Sumário

Prefácio

Este material[1] foi produzido para servir de apoio a disciplina Novas Tecnologias Web e Internet do curso técnico a distância em Informática da diretoria de educação a distância do IFCE. Seu principal objetivo é ser um guia que contém todo o conteúdo que será abordado durante a disciplina, mas que necessita ser complementado com outros materiais como vídeo aulas, livros, sites indicados, além dos mecanismos de aprendizagem disponíveis na ferramenta (email, fórum, etc), bem como os tutores a distância e formadores responsáveis da disciplina.

Esperamos que todos os estudantes possam aproveitar ao máximo o conteúdo abordado e tenham bom aproveitamento durante a disciplina.

Um grande abraço a todos!

Aula 1

Introdução ao HTML

Triste época! É mais fácil desintegrar um átomo do que um preconceito.
Albert Einstein

Objetivos

- Identificar as principais *tags* utilizadas no HTML; ■< .mine
- Conhecer a estrutura básica de uma página;
- Aplicar *tags* em páginas *web* básicas
- Construir páginas *web* com formulários =======
- Conhecer a estrutura básica de uma página;
- Aplicar *tags* em páginas *web* básicas;
- Construir páginas *web* com formulários. ■> .r45

1.1 Páginas web

Antes de começar o estudo sobre qualquer linguagem de programação para *web*, é necessário entender o funcionamento de um navegador. Os *browsers*, como Internet Explorer, Mozilla Firefox, Safari e outros, nada mais são do que visualizadores de documentos HTML, ou seja, recebem arquivos HTML provenientes de um servidor web, interpretam e exibem o resultado para o usuário na forma de hipertexto, imagens e tudo mais que pode ser visualizado através dos navegadores.

O HTML (*Hypertext Markup Language*) é exatamente a maneira como as páginas web são descritas. É uma pseudo-linguagem, pois trata-se apenas de uma linguagem de marcações (também chamadas de *tags*) colocadas ao longo do conteúdo do documento. Estas marcações nada mais são do que formatações sobre a forma como o conteúdo será exibido pelo navegador, tais como fontes, cores, tamanho, tabelas e demais elementos presentes na página.

Um site é um conjunto de páginas web que, por sua vez, são apenas arquivos HTML. Cada página contém diversas *tags* que são interpretadas pelos navegadores que exibem o conteúdo da maneira usual. Os navegadores não exibem o conteúdo do documento HTML (fonte) diretamente, mas sim sua interpretação visual. Caso o usuário solicite, o código fonte da página pode ser exibido pelo navegador. Para o Firefox ou Internet Explorer, basta selecionar Exibir|Codificação que é mostrado o código fonte da página.

Abaixo, temos o primeiro exemplo de arquivo em HTML. Para visualizá-lo, basta salvar o conteúdo mostrado na figura 1.1 em um arquivo com a extensão .htm ou .html e abrir com qualquer navegador instalado no computador.

Figura 1.1: *Hello world* na versão HTML.

```
<html>
<body>
<h1>Hello, World</h1>
<p>Ola Mundo</p>
</body>
</html>
```

Este pequeno documento de exemplo é composto por 4 *tags*: <html>, <body>, <h1> e . As marcações do documento HTML são colocadas mescladas ao conteúdo. Cada marcação é interpretada pelo navegador para exibir o conteúdo para o usuário da maneira indicada.

A primeira *tag* (<html>) inicia e finaliza qualquer documento HTML. Todas as outras *tags* devem estar entre tags <html></html>. Praticamente todas as *tags* necessitam ser fechadas com uma *tag* de mesmo nome acrescida de / no início.

O que está entre <body> e </body> delimita o que é chamado de corpo do documento, é a parte visível ao usuário. Apenas o que está entre *tags* <body> é exibido na tela do navegador ao usuário.

A *tag* <h1> formata o texto "Hello, World"como um cabeçalho de nível 1 (*heading*). Há ainda outros cinco níveis formando os cabeçalhos <h2>, <h3>, <h4>, <h5> e <h6>. Sendo <h1> o de maior destaque e <h6> o menos destacado.

Por fim, a *tag* delimita um parágrafo do conteúdo. O fechamento desta *tag* é opcional. Assim, pode-se remover o sem nenhum prejuízo à exibição.

1.2 Ferramentas

Para visualizar arquivos simples em HTML não é preciso nenhum software específico. Basta um editor de texto simples e o navegador para visualizar o resultado. Entretanto, a medida que a quantidade de páginas e a complexidade aumentam, ferramentas mais adequadas podem ser utilizadas para criar as páginas HTML de um site e facilitar o trabalho do desenvolvedor *web*.

Alguns exemplos de editores HTML livres são o Bluefish e o NVu. Do lado dos proprietários, temos o Dreamweaver, Cold Fusion e também o Visual Studio. Qualquer destas ferramentas atende as necessidades do que será abordado neste material, entretanto, as ferramentas livres são preferíveis devido a facilidade de acesso.

1.3 Atributos

As *tags* são complementadas através de atributos que são formados de pares de nome de *tag* e valor atribuído. Seguem a seguinte sintaxe: nome=valor. Eles são utilizados sempre na *tag* inicial e nunca no fechamento. Vejamos:

```
<a href="http://www.google.com.br">Buscador</a>
```

O atributo `href` indica o endereço do hiperlink. O texto entre as *tags* é a parte exibida para o usuário, no caso, "Buscador". Cada *tag* possui atributos específicos. Para uma referência detalhada sobre os atributos, pode-se consultar a descrição completa de cada um deles no endereço da **W3 Schools**.

A *tag* <font> é utilizada para alterar parâmetros relacionados à exibição de fontes no navegador. Entretanto, o uso desta *tag* não é mais recomendado no HTML 4 e foi totalmente removida no HTML 5. Atualmente, a recomendação é formatar as fontes através de folhas de estilo CSS (*Cascading Style Sheets*). Assim mesmo, ainda é bom um exemplo para demonstrar o uso dos atributos. Na figura 1.2,

observa-se que, mesmo utilizando as folhas de estilo CSS (citadas anteriormente) o resultado obtido é o mesmo.

Figura 1.2: Exemplo de utilização da tag <font>

```
<p>
<font size="5" face="arial" color="red">
Este paragrafo esta em Arial, tamanho 5 e cor vermelha.
</font>
</p>

<p>
<font size="3" face="verdana" color="blue">
Este paragrafo esta em Arial, tamanho 3 e cor azul.
</font>
</p>
```

Figura 1.3: Exemplo de formatação de com CSS

```
<p style="font-family:arial;color:red;font-size:20px;">
Este paragrafo usa fonte Arial, tamanho 20 pixels e cor vermelha.
</font>
</p>
<p style="font-family:verdana;color:#10A030;font-size:30px;">
Este paragrafo usa fonte Verdana, tamanho 30 pixels e cor especificada em RGB
</font>
</p>
```

1.4 *Hiperlinks*, imagens e tabelas

Dificilmente encontraremos um *site* na Internet, por mais simples que seja, sem encontrar *hiperlinks*, que é a essência do HTML. As imagens também estão sempre presentes e as tabelas são fundamentais para organizar e dar uma melhor organização ao conteúdo. Dominar o uso destas *tags* é fundamental para o desenvolvimento de páginas de qualquer tipo.

Os conhecidos *hiperlinks* são palavras ou imagens que, ao serem clicadas, levam a uma nova página ou a uma nova seção da mesma página. Ao mover o ponteiro do mouse sobre um link, o ponteiro se altera, indicando a presença de um *hiperlink*. A *tag* <a> é utilizada para criação de links em uma página. O formato básico é:

```
<a href="url">Texto do link</a>
```

O atributo `href` especifica o destino do link. Apenas o texto entre as *tags* é exibido no navegador. Outro atributo bastante utilizado é `target`. O exemplo a seguir abre a página em uma nova janela:

```
<a href="http://www.google.com" target="_blank">Buscador</a>
```

Para utilizar imagens nas páginas web é necessário utilizar a *tag* <img> que não necessita ser fechada. O atributo `src` é utilizado para indicar o caminho para o arquivo que será exibido. Caso o arquivo esteja no mesmo diretório da página, basta indicar o nome. Os principais formatos utilizados são: JPG, GIF e PNG. O valor do atributo `alt` é exibido quando o mouse passa sobre a figura. Abaixo é mostrado um exemplo do uso de <img>:

```
<img src="boat.gif" alt="Imagem de um barco">
```

Também é possível alterar o tamanho de exibição da imagem no navegador através dos atributos `heigth` e `width`, conforme exemplo a seguir. Caso estes atributos não sejam atribuídos a imagem é mostrada em seu tamanho real,sem ampliações ou reduções.

```
<img src="pulpit.jpg"alt="Pulpit rock"width="304"height="228"
```

Para utilizar imagens como hyperlinks, basta substituir o texto do link entre as *tags* por uma *tag* <img>. Assim, a imagem aparecerá no navegador como link ao invés de texto.

As tabelas são definidas com a *tag* <table>. Cada tabela possui linhas (<tr>) e células (<td>). O conteúdo da tabela fica dentro das células. Pode conter texto, links, imagens, listas ou até mesmo outras tabelas. A figura 1.4 exemplifica o uso destas *tags*:

Figura 1.4: Exemplo de tabela

```
<table border="1">
<tr>
<td>linha 1, celula 1</td>
<td>linha 1, celula 2</td>
</tr>
<tr>
<td>linha 2, celula 1</td>
<td>linha 2, celula 2</td>
</tr>
</table>
```

1.5 Formulários

Formulários são utilizados para passar informações de uma página a outra ou do usuário para o servidor, por exemplo. Um formulário deve conter elementos de entrada de dados como caixas de texto, *checkbox*, botões de envio, listas de seleção e outros elementos disponíveis. Para criar um formulário, utiliza-se a *tag* <form> que possui diversos atributos como `action` e `name`. Para acrescentar elementos ao formulário, basta acrescentar uma *tag* <input> entre as *tags* <form>, segundo o exemplo a seguir.

Figura 1.5: Exemplo de formulário com vários elementos

```
<form name="pesquisa">
Primeiro Nome: <input type="text" name="firstname"><br>
Sobrenome: <input type="text" name="lastname">
<input type="radio" name="sex" value="masculino">Masculino<br>
<input type="radio" name="sex" value="feminino">Feminino<br>
Meios de Transporte que Utiliza:<br>
<input type="checkbox" name="vehicle" value="Bicicleta">Bicicleta<br>
<input type="checkbox" name="vehicle" value="Carro">Carro
</form>
```

No exemplo anterior, o formulário é definido com o nome `pesquisa`, conforme o atributo presente na *tag* <form>. Em seguida, estão presentes 6 elementos, sendo duas caixas de texto, dois botões de rádio e dois *checkboxes*, respectivamente.

Nas caixas de texto, o usuário pode inserir texto livremente. Os botões de rádio servem para escolher uma opção entre várias disponíveis. Já o *checkbox*, deve ser utilizado para selecionar uma opção única dentre várias disponíveis.

O formulário da figura 1.5 ainda está incompleto. Faltam alguns elementos para torná-lo funcional. Para que isto aconteça, é necessário que seja utilizado o atributo `action` da tag <form>. Este atributo indica o arquivo ou *script* que irá processar os dados passados pelo formulário.

Normalmente, os *scripts* são processados no lado do servidor. O cliente (navegador) apenas envia as informações para um servidor na Internet ou localmente para que este realize todo o processamento (cálculos, armazenamento, otimização, etc) da informação e devolva uma resposta ao cliente.

Na figura 1.6, o valor de `action=html_form_action.php` indica que este *script* em linguagem PHP realizará o processamento do formulário. Também foi acrescentado o botão `submit` para enviar os dados do formulário para o arquivo indicado em `action` para processamento.

Figura 1.6: Formulário com botão de envio e `action`

```
<form name="pesquisa" action="html_form_action.php" method="get">
Primeiro Nome: <input type="text" name="firstname"><br>
```

```
Sobrenome: <input type="text" name="lastname">
<input type="radio" name="sex" value="masculino">Masculino<br>
<input type="radio" name="sex" value="feminino">Feminino<br>
Meios de Transporte que Utiliza:<br>
<input type="checkbox" name="vehicle" value="Bicicleta"> Bicicleta<br>
<input type="checkbox" name="vehicle" value="Carro"> Carro<br>
<input type="submit" value="Enviar">
</form>
```

1.6 Resumo

Na seção 1.1 foi apresentado o conceito de página *web* e seus principais usos. Na seção 1.2 foram apresentadas as principais ferramentas que podem ser utilizadas pelo *webdesigner* para desenvolver suas páginas. Já na seção 1.3 são apresentados as principais *tags* juntamente com seus respectivos atributos e também uma referência completa no *site* da **W3 *Schools***. As *tags* mais utilizadas em qualquer página são apresentadas na seção 1.4. E na seção 1.5 são apresentadas as *tags* associadas com formulários juntamente com exemplos.

1.7 Exercícios

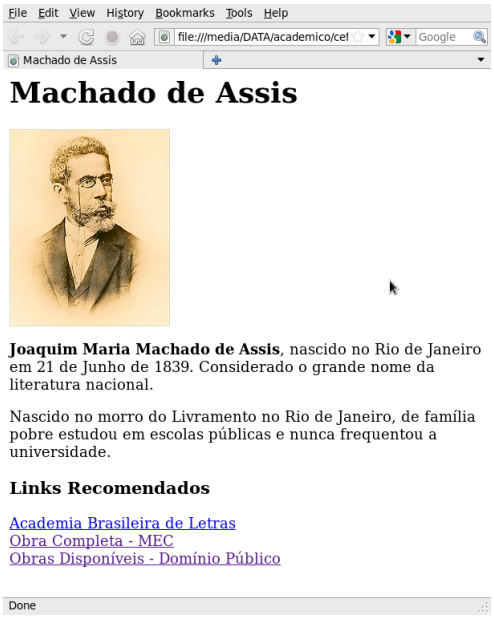
1. Utilizando uma das ferramentas indicadas na seção 1.2, crie uma página web pessoal (eu.html) contendo alguns de seus dados pessoais, como nome, cidade onde nasceu, data de nascimento e informações similares. Utilize tags <p> para inserir um ou mais parágrafos com essas informações.

2. No exercício do ítem anterior, acrescente uma foto sua ao arquivo.

3. Ainda com relação ao exercício 1, acrescente, no final do arquivo, links para três sites externos com um texto em destaque **Links Recomendados**.

4. Ao final dos exercícios anteriores sua página deve ter uma aparência semelhante a figura 1.1.

Figura 1.1: Screenshot de uma página em HTML



5. Crie um arquivo html (tabela.html) que contenha as informações mostradas na tabela 1.1. Utilize a tag <table> e demais tags relacionadas (<tr>, <td>, etc).

Tabela 1.1: Lista de preços

ítem	descrição	quantidade
1	Notebook	2
2	Processador	10
3	Mouse	5
4	Monitor	11

6. As tabelas podem ser utilizadas para organizar informações em uma página como pedido no exercício 5. Outra utilidade das tabelas é que permitem organizar a disposição dos elementos de uma página. O código HTML, mostrado na figura 1.7, exemplifica essa idéia. Salve o código em um arquivo e visualize o resultado no navegador.

Figura 1.7: Exemplo do uso de tabelas para organizar os elementos da página

```
1 <html>
2 <head>
3 <title>Machado de Assis</title>
4 </head>
5 <body>
```

```
<h1>Machado de Assis</h1>
<table>
<tr>
<td><img src="../figs/machado.jpg" width="200" height="246"></td>
<td><p><b>Joaquim Maria Machado de Assis</b>, nascido no Rio de Janeiro
em 21 de Junho de 1839.<br>
Considerado o grande nome da literatura nacional.</p>
<p>Nascido no morro do Livramento no Rio de Janeiro, de familia pobre,
estudou em escolas publicas e nunca frequentou a universidade.</p>
</td>
</tr>
<tr>
<td> </td>
<td><h3>Links Recomendados</h3>
<a href="www.machadodeassis.org.br/">Academia Brasileira de Letras</a><br>
<a href="http://machado.mec.gov.br/">Obra Completa - MEC</a><br>
</td>
</tr>
</table>
</body>
</html>
```

7. Modifique o código mostrada na figura 1.7 para que a imagem seja exibida no lado direito ao invés do lado esquerdo. Pesquise na internet como modificar a cor de fundo de uma tabela e aplique nesta página.

8. Crie uma página para o cadastramento (form.html) de um usuário na intranet de uma empresa. O formulário deve conter os campos: nome completo, setor (compras, vendas ou financeiro), nome do usuário e senha.

9. Dada a tag <body> mostrada abaixo, pesquise para que servem os atributos BGCOLOR, TEXT, LINK, ALINK e VLINK. Aplique-os na página do exercício 1.

```
<body bgcolor="#rrggbb" text="#rrggbb" link="#rrggbb"
alink="#rrggbb" VLINK="#rrggbb">
```

10. Diferencie os termos **páginas estáticas** e **páginas dinâmicas** relacionados a construção de *sites* em geral.

Aula 2

Começando em PHP

"Estamos irrevogavelmente em um caminho que nos levará
às estrelas. A não ser que, por uma monstruosa capitulação ao egoísmo
e à estupidez, acabemos nos destruindo."
Carl Sagan

Objetivos

- Conhecer a dinâmica de funcionamento do PHP;
- Conhecer os comandos básicos da linguagem;
- Conhecer os tipos de dados, operadores e funções utilizadas.

2.1 Mais um acrônimo

Os iniciantes em desenvolvimento para *web* ou em tecnologia da informação não devem se assustar com a quantidade de acrônimos (siglas) existentes. ASP, CGI, SOAP, XML e HTTP são alguns deles. A lista pode ser acrescida de vários outros ítens e pode parecer interminável assustando os menos experientes no assunto. Para uma formação completa em desenvolvimento de *sites* de qualidade e, principalmente com funcionalidade, é indispensável acrescentar na lista a linguagem de *scripts* PHP.

O PHP é uma linguagem de programação voltada para *web* bastante conhecida. Trata-se da única linguagem de *scripts* baseada em servidor que possui código aberto. É bastante flexível e fácil de aprender. Pesquisas mostram que milhões de *websites* utilizam a linguagem PHP como base para suas aplicações. A razão para isso é a grande capacidade do PHP, pois é uma linguagem poderosa, fácil de usar e aprender. Além de ser extremamente robusta e escalável, a linguagem PHP pode ser utlizada em aplicações de alta demanda com boas respostas. Possui suporte a diversos bancos de dados, a XML, permite a criação de *frameworks* do próprio programador e ainda excelente documentação. Como se todas estas vantagens não fossem suficientes ainda é gratuita.

2.2 O ambiente de desenvolvimento

Para que o PHP funcione, é necessário combiná-lo com um servidor *web*, tipicamente o Apache. As requisições a *scripts* são feitas pelo usuário ou pela aplicação são recebidas pelo servidor *web* e manipuladas pelo interpretador PHP. O resultado obtido depois da execução é devolvido ao servidor web que transmite a informação para o cliente (navegador). O *script* é o programa propriamente dito.

É onde está escrito o código da aplicação que funciona na *internet*. Nele, é possível fazer cálculos, processar entradas do usuário, interagir com bancos de dados, ler e escrever em arquivos e tudo mais que uma linguagem de programação é capaz de realizar.

As atividades propostas neste material foram implementadas utilizando o PHP, juntamente com o servidor *web* Apache, no sistema operacional Linux. Essa é a combinação de ferramentas mais comum para utilizar o PHP. Entretanto, existem versões tanto do PHP quanto do Apache para outros sistemas operacionais, como Windows e MacOS. Instruções detalhadas sobre a instalação e preparação do ambiente de desenvolvimento para cada plataforma podem ser encontradas no manual do PHP disponível no endereço:

http://www.php.net/manual/pt_BR/install.php

.

2.3 Iniciando

A maneira mais simples de programar em PHP é embutir o código da linguagem dentro de um arquivo contendo *tags* HTML. O código embutido nas páginas é executado quando a página é carregada.

Os comandos da linguagem PHP devem estar sempre entre *tags* específicas indicando ao Apache que os comandos entre estas *tags* devem ser processados pelo interpretador PHP como pode ser observado no código 2.1.

Figura 2.1: *Tags* da linguagem PHP

```
1 <?php
2
3 comandos PHP
4
5 ?>
```

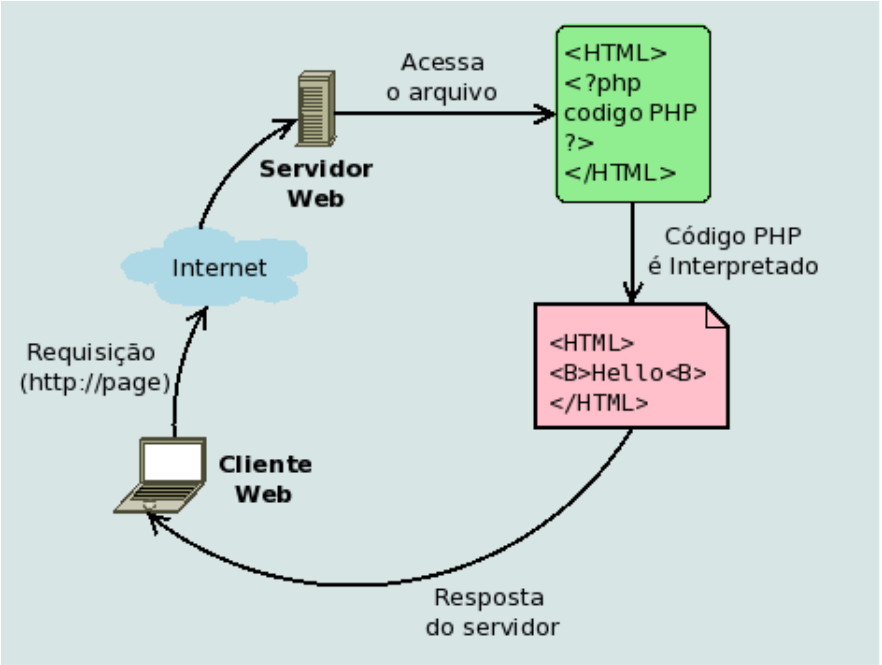
Um exemplo simples de como os comandos PHP são colocados junto com HTML é mostrado em 2.2. Na linha 7, a *tag* `<b></b>` é utilizada diretamente no corpo do documento. Já na linha 12, a saída de `echo()` já contém a mesma *tag* `<b></b>` com a mesma funcionalidade descrita na linha 7, porém misturando PHP e HTML.

Figura 2.2: Código PHP e HTML mesclados.

```
1  <html>
2  <head>
3  <title>Matrix</title>
4  </head>
5  <body>
6
7  <b>Agente</b>: Quem voce pensa que e?
8  <br>
9
10 <?php
11 // saida mostrada
12 echo '<b>Neo</b>: Eu sou Neo, mas me chamam de o escolhido.';
13 ?>
14
15 </body>
16 </html>
```

Executando-se o código mostrado em 2.2 e abrindo o código HTML lido pelo navegador é possível observar que as tags PHP não estão presentes. O que aconteceu? Quando a página é requisitada ao servidor web (Apache) pelo cliente *web* (navegador) esta é interpretada pelo interpretador PHP presente no servidor, que, por sua vez, repassa o resultado na forma de um arquivo HTML que é visualizado pelo navegador. Este processo está ilustrado na figura 2.1. É importante notar que a figura ilustra o caso típico de um servidor que está sendo acessado através de uma rede interna ou

Figura 2.1: Interação entre cliente e servidor *web* com suporte a PHP.



Fonte: Autores.

da própria internet, conhecido como um ambiente de **produção**. O que quer dizer que os elementos estão interagindo em um sistema real, em pleno funcionamento.

Em oposição a um sistema de produção, temos os sitemas de teste ou desenvolvimento em que, normalmente, todos os elementos citados anteriormente funcionam em uma única máquina, conforme ilustrado na figura 2.2. Estes sistemas são utilizados para testar modificações antes de serem colocadas em funcionamento no servidor de produção e também para aprendizagem e estudos. Na figura 2.2, são mostrados dois processos: o cliente (navegador) e o servidor web. A troca de informações entre eles se dá através da interface de rede local ao invés da internet, que normalmente é usada em um sistema de produção. Desta maneira, não é necessário um servidor externo para aprendizagem ou testes, pois todos os processos envolvidos (navegador e servidor *web*) estão rodando na mesma máquina.

Para inserir comentários ao longo do código fonte em PHP, existem duas opções: comentários de apenas uma linha ou comentários de várias linhas. Para comentários de uma linha basta acrescentar //. Já os comentários de várias linhas são iniciados com /* e finalizados com */. As linhas em branco são ignoradas, assim como qualquer caractere fora das *tags*. Os comentários seguem o mesmo padrão da linguagem C, que aliás é a base do PHP. Nos exemplos mostrados neste material, pode-se observar muitas semelhanças entre a sintaxe utilizada em ambas as linguagens.

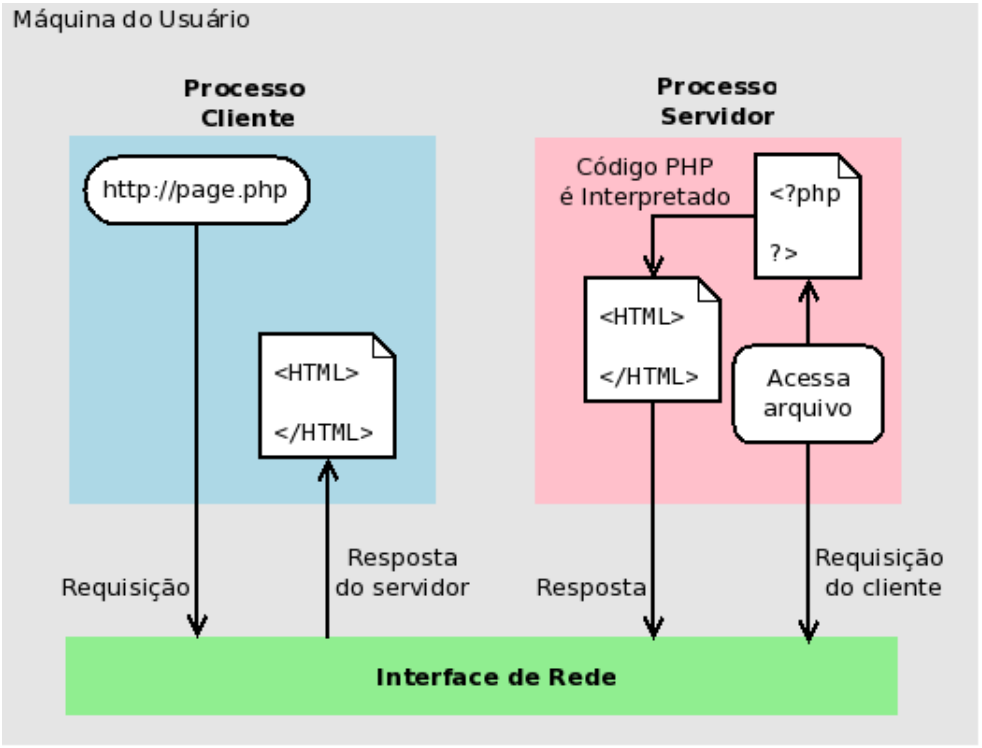
Figura 2.3: Comentários em PHP

```
1 <?php
2
3 // comentario de linha unica
4
5 /* comentario
6 de multiplas
7 linhas */
8 ?>
```

2.4 Variáveis

Assim como qualquer outra linguagem de programação, o PHP também utiliza-se de variáveis para armazenar dados durante a execução de um *script*. Seu conteúdo pode mudar ao longo da execução

Figura 2.2: Sistema de desenvolvimento típico utilizando apenas uma máquina.



Fonte: Autores.

do *script*. As variáveis podem ser comparadas e o resultado das comparações pode ser utilizado para disparar ações específicas como em qualquer outra linguagem.

O PHP suporta uma grande quantidade de tipos de variáveis: inteiros, ponto flutuante, strings e matrizes. Na grande maioria das linguagens, é fundamental especificar o tipo de variável que está sendo utilizada logo no início do programa. O que é conhecido como declaração de variáveis. No caso da linguagem PHP, o próprio interpretador determina o tipo de variável através do contexto em que é utilizada. Inclusive, uma variável pode ser tratada inicialmente como inteiro e durante a execução passar a ser tratada como *string*. Isto traz bastante flexibilidade no uso de variáveis e conforto para o programador, já que ele não precisa preocupar-se com a declaração das variáveis.

Cada variável deve ser identificada por um nome. O nome de uma variável é precedido por $ e seguido por uma letra ou _. São exemplos de nomes de variáveis válidos: $popeye, $one e $INCOME. Já os nomes `$48hrs` e `$123` não são válidos. Os nomes também são sensíveis ao caso, logo, $me é diferente de $ME.

No código 2.4, é mostrado um exemplo simples de utilização de variáveis em PHP. As variáveis $name, $rank e $serialNumber são definidas como string e como número, mas são utilizadas como string na chamada da função `echo()`. Assim, como `printf()`, `echo()` é utilizada para exibir dados na saída padrão. Na chamada da função `echo()`, ainda foram incluídas tags HTML que serão processadas pelo navegador.

Figura 2.4: Variáveis em PHP

```
1  <html>
2  <head>
3  <title>Matrix</title>
4  </head>
5  <body>
6
7  Agente: Quem voce pensa que e?
8  <br>
9
10 <?php
11 // define as variaveis
12 $name = 'Neo';
```

```
$rank = 'Anomalia';
$serialNumber = 1;

// imprime saida
echo "Neo: Eu sou <b>$name</b>, o <b>$rank</b>.
Qual meu numero de serie, <b>$serialNumber</b>.";
?>

</body>
</html>
```

Para atribuir valor a uma variável, basta utilizar o sinal = a direita do nome da variável seguido do valor que deseja-se atribuir, conforme mostrado a seguir.

Figura 2.5: Atribuição em PHP

```
<?php

$old = 1;
$age = $old + 15;
$angle1 = $angle2 = $angle3 = 60;

?>
```

Os principais tipos de variáveis disponíveis em PHP são:

- Booleano (bool);
- Inteiro (int);
- Ponto flutuante (float);
- *String*;
- Vetores (arrays).

Para especificar uma variável como boolena, basta atribuir as palavras-chave TRUE ou FALSE. É o tipo de dado mais simples. Expressa um valor verdade. Também são considerados falsos o valor 0 (inteiro), 0.0 (ponto flutuante), string vazia e um array sem elementos.

Os números inteiros são compostos pelo conjunto $Z = \{... - 2, -1, 0, 1, 2, ...\}$. Podem ser especificados em base decimal, hexadecimal ou octal. Em octal é necessário preceder o número com 0. Já em hexadecimal é necessário preceder por $0x$.

Os números em ponto flutuante são os chamados números reais. Utiliza-se o ponto para separar a parte inteira da parte não inteira ($a = 1.234). Também pode-se utilizar a notação científica ($a = 1.23e4 ou $b = 7E-10).

Uma *string* é especificada utilizando-se aspas simples ($nome = 'maria') ou aspas duplas ($palavra = "word"). Porém, no caso de aspas duplas, se uma variável é colocada entre as aspas, o seu valor é substituído.

Um *array*, em PHP, é considerado um mapa ordenado, ou seja, é um tipo que relaciona valores para chaves. Este tipo é otimizado de várias maneiras, assim, é possível usá-lo como um array real, ou uma lista (vetor), *hashtable* (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente o que mais a criatividade do programador alcançar.

Um array contém um certo número de pares, separados por vírgula, `chave=>valor`. A chave pode ser tanto um inteiro quanto uma string. No código 2.6 é mostrado um exemplo de utilização deste tipo.

Figura 2.6: Atribuição de arrays em PHP

```
<?php
$arr = array("foo" => "bar", 12 => true);

echo $arr["foo"]; // bar
```

```
5 echo $arr[12];     // 1
6 /$
7 ?>
```

2.5 Operadores

A melhor maneira de se familiarizar com os operadores disponíveis em uma linguagem é utilizando-os. A seguir, seguem exemplos de utilização de diversos operadores suportados pelo PHP:

Figura 2.7: Principais operadores em PHP

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <?php
6 // atribui valor a quantity
7 $quantity = 1000;
8 // atribui valor a preco original e preco corrente
9 $origPrice = 100;
10 $currPrice = 25;
11 // calcula a diferenca de precos
12 $diffPrice = $currPrice - $origPrice;
13 // calcula a percetagem
14 $diffPricePercent = (($currPrice - $origPrice) * 100)/$origPrice;
15 //£
16 ?>
17 <table border="1" cellpadding="5" cellspacing="0">
18 <tr>
19 <td>Quantidade</td>
20 <td>Preco de custo</td>
21 <td>Preco atual</td>
22 <td>Alteracao absoluta</td>
23 <td>Alteracao percentual</td>
24 </tr>
25 <tr>
26 <td><?php echo $quantity ?></td>
27 <td><?php echo $origPrice ?></td>
28 <td><?php echo $currPrice ?></td>
29 <td><?php echo $diffPrice ?></td>
30 <td><?php echo $diffPricePercent ?>Percentual</td>
31 </tr>
32 </table>
33 </body>
34 </html>
```

O PHP também disponibiliza operadores bastante úteis para realizar operações com *strings*. A concatenação de *strings* é feita com o operador (.). O código 2.8 exemplifica a concatenação utilizando esse operador.

Figura 2.8: Outros operadores úteis em PHP

```
1 <?php
2 // algumas strings
3 $a = 'os';
4 $b = 'jogos';
5 $c = 'iniciam';
6 $d = 'agora';
7
8 // os valores sao combinados
9 // e retornado 'os jogos iniciam agora<br>'
10 $statement = $a.' '.$b.' '.$c.' '.$d.'<br />';
11 print $statement;
12
```

```
// outra variacao 'iniciam os jogos agora!'
$command = $c.' '.$a.' '.$b.' '.$d.'!';
print $command;
?>
```

2.6 Formulários

O que torna o PHP uma linguagem para web é sua característica própria de receber entradas do usuário a partir de um formulário de uma página web e converter os dados passados em variáveis.

O uso de formulários é a maneira mais natural e simples de interagir com páginas *web*, pois realizam a interação entre clientes e servidores através de todo tipo de aplicações bastante conhecida na *web*. Aplicações de comércio eletrônico e transações bancárias são alguns dos exemplos típicos na qual os de formulários para passam informações. O PHP é bastante ágil no recebimento das informações passadas pelos clientes (navegadores) e as processa no servidor *web*.

A popularização da internet tornou os navegadores a ferramenta mais prática para os usuários interagirem com as aplicações disponibilizadas pelas empresas e instituições. Também flexibilizou o acesso aos aplicativos, dispensando a necessidade de distribuição de arquivos executáveis, já que as aplicações estão disponíveis em sites. A manutenção é centralizada tirando do usuário o trabalho de atualizar o software instalado em sua máquina.

A seguir, temos o primeiro exemplo de uma aplicação em que são passados dados através de um formulário para ser processado por um *script* em PHP. O exemplo é composto basicamente de dois *scripts*: o primeiro contém o formulário HTML (form.html) e o segundo contém a lógica de processamento (message.php).

Figura 2.9: form.html

```
<html>
<head></head>
<body>
<form action="message.php" method="post">
Entre sua mensagem: <input type="text" name="msg" size="30">
<input type="submit" value="Enviar">
</form>
</body>
</html>
```

O atributo **action** da tag `<form>` especifica o nome do *script* do lado do servidor (*server-side script*) que será utilizado no processamento dos dados passados no formulário. O segundo atributo `method` especifica a maneira como os dados serão passados. O HTTP dispõe basicamente de dois métodos básicos para trocar informações com formulários: `GET` e `POST`. Mais detalhes sobre esses métodos são apresentados na seção 2.6.1.

Figura 2.10: message.php

```
<html>
<head></head>
<body>
<?php
// recebe dados do formulario
$input = $_POST['msg'];
// usa os dados
echo "Voce disse: <i>$input</i>";
?>
</body>
</html>
/$
```

Quando os dados são passados (submetidos) através de `form.html`, o *script* `message.php` lê e exibe os dados segundo a lógica implementada. Assim, para cada formulário submetido a um (*script*)

PHP os pares valor-variável estão disponíveis para uso no *script* através de uma variável *container* especial, `$_POST`.

Obviamente, o PHP também suporta o método `GET`. No formulário, basta alterar o atributo `method` para `GET` ao invés de `POST`. E para acessar o valor, utilizar o *container* `$_GET`.

2.6.1 Métodos HTTP

O *HyperText Transfer Protocol* (Protocolo de transferência de hipertexto - HTTP) é o protocolo de comunicação utilizado para a troca de dados entre um navegador e um servidor web. Ele é acionado quando o usuário clica em um endereço no seu navegador. A troca de informações entre os dois processos, cliente e servidor, é feita por algum método disponibilizado pelo HTTP. Os principais métodos associados à transferência de dados de formulários são: o método GET e o método POST.

GET

O método `GET` é acionado por meio de um formulário HTML através do atributo `method=get` incluída em uma *tag* <form>. Por meio desse método, os dados constantes no formulário são primeiramente transmitidos ao processo servidor e este, por sua vez, armazena os dados temporariamente numa variável de contexto denominada QUERY_STRING (ver seção 2.6.2).

Quando um formulário HTML utiliza o método GET, o fluxo de dados é separado do endereço URL que chama o *script* através de um ponto de interrogação (?). Esta forma de endereçamento e separação pode ser observada no campo de endereços do navegador do usuário, logo após o formulário ter sido enviado, como mostrado a seguir:

```
http://www.meusite.com/meuscript.cgi?nome=Maria&id=123
```

POST

O método `POST` é selecionado de forma similar ao `GET` no formulário, através do atributo `method`. O `POST` faz com que os dados do formulário sejam diretamente transmitidos ao endereço que constar no atributo `action`. Um *script*, chamado por `action`, precisa extrair os dados através da entrada padrão (*standart input*) para poder obter os dados transmitidos pelo formulário. O conteúdo das variáveis do formulário não fica exposto no campo de endereço do navegador (o que é considerado uma forma insegura de trocar informações na internet, por exemplo).

Normalmente, as aplicações *web* utilizam este método para trafegar os dados, justamente devido a esse fragilidade do método `GET`.

2.6.2 *Query string*

Uma maneira alternativa de passar dados de uma página para outra é através do próprio campo de endereço do navegador (URL). Esta forma de passar dados entre as páginas é conhecida como *Querystring*. Está estruturada da seguinte forma:

```
http://servidor/pagina.html?var1=dado1&var2=dado2&var3=dado3
```

Após o sinal de ? inicia-se a sequência de nomes de variáveis seguido do sinal de = e os respectivos valores das variáveis. Não pode haver espaço em branco nesta sequência. Para valores de strings que contenham espaços em branco é necessário substituí-los pelo sinal +. Os pares variável/valor são separados &.

A listagem 2.11 exemplifica o uso de *querystring*. Salve o conteúdo em um arquivo PHP de nome `qse.php` no diretório padrão do Apache e digite no campo de endereço do navegador a seguinte URL:

```
http://localhost/qse.php?nome=ana&idade=12
```

Figura 2.11: Exemplo de *script* que utiliza *query string*

```
1 <?php
2 $nome = $_REQUEST['nome'];
3 $idade = $_REQUEST['idade'];
```

```
4
5 echo $nome . " tem " . $idade . " anos de idade";
6 ?>
```

2.7 Comparando

Para agregar um pouco de inteligência aos scripts é necessário utilizar declarações condicionais, ou seja, fazer com que o script desempenhe ações baseadas no resultado de um teste de comparação.

Alguns operadores específicos para comparação podem ser utilizados. No código 2.12 são mostrados alguns deles:

Figura 2.12: Operadores de comparação

```
1  <?php
2  /* define duas variaveis */
3  $str = '10';
4  $int = 10;
5
6  /* retorna true, desde que ambas contenham o mesmo valor */
7  $result = ($str == $int);
8  print "resultado e $result<br>";
9
10 /* retorna false, desde que as variaveis nao sejam do mesmo tipo, mesmo contendo o mesmo
11 $result = ($str === $int);
12 print "resultado e $result<br>";
13
14 /* returna true, desde que as variaveis tenham mesmo valor e tipo */
15 $anotherInt = 10;
16 $result = ($anotherInt === $int);
17 print "resultado e $result";
18 /$
19 ?>
```

Também estão disponíveis no PHP os operadores lógicos: AND, OR, XOR e NOT. O uso destes operadores é ilustrado no código 2.13:

Figura 2.13: Operadores lógicos em PHP

```
1  <?php
2  /* definicao de variaveis */
3  $auth = 1;
4  $status = 1;
5  $role = 4;
6
7  /* AND logico returna true se todas as condicoes sao true */
8  // returna true
9  $result = (($auth == 1) && ($status != 0));
10 print "resultado e $result<br>";
11
12 /* OR logico returna true se alguma condicao e true */
13 // returna true
14 $result = (($status == 1) || ($role <= 2));
15 print "resultado e $result<br>";
16
17 /* NOT logico returna true se a condicao e false e vice-versa */
18 // returna false
19 $result = !($status == 1);
20 print "result is $result<br>";
21
22 // XOR logico returna true se nenhuma das duas condicoes sao true, ou
23 // returna false se ambas condicoes sao true
24 $result = (($status == 1) xor ($auth == 1));
25 print "resultado e $result<br>";
```

2.8 Controle de fluxo

Em PHP a forma simples de controle de fluxo é através da declaração `if`. Seu argumento é uma expressão condicional, ou seja, que retorna TRUE ou FALSE. O exemplo a seguir, mostra o uso de `if` juntamente com um formulário que irá realizar o processamento de acordo com a idade do usuário.

Figura 2.14: exemplo de if

```
<html>
<head></head>
<body>
<form action="ageist.php" method="post">
Digite sua idade: <input name="age" size="2">
</form>
</body>
</html>
```

O código 2.15 mostra o *script* que processa o formulário anterior:

Figura 2.15: ageist.php

```
<html>
<head></head>
<body>
<?php
// retrieve form data
$age = $_POST['age'];
// verificacao do valor
if ($age >= 18)
{
     echo 'voce pode tirar carteira de motorista!';
}
if ($age < 18)
{
     echo "voce nao pode tentar tirar carteira de motorista ainda";
}
?>
</body>
</html>
```

Além da estrutura tradicional `if-else` o PHP dispõe ainda de uma estrutura especial: `if-elseif-else`. As listagens 2.16 e 2.17 exemplificam seu uso.

Figura 2.16: Formulário

```
<html>
<head></head>
<body>
<h2>Today's Special</h2>
<p>
<form method="get" action="cooking.php">
<select name="day">
<option value="1">Segunda/Quarta
<option value="2">Terca/Quinta
<option value="3">Sexta/Domingo
<option value="4">Sabado
</select>
<input type="submit" value="Enviar">
</form>
</body>
</html>
```

Figura 2.17: cooking.php

```
<html>
<head></head>
```

```
<body>
<?php
// pega selecao do formulario
$day = $_GET['day'];
// checa a opcao e escolhe o item apropriado
if ($day == 1) {
    $special = 'Frango ao molho';
    }
elseif ($day == 2) {
    $special = 'Sopa francesa';
    }
elseif ($day == 3) {
    $special = 'Pure de batata e carne';
    }
else {
    $special = 'Peixe e batatas';
}
?>
<h2>O cardapio de hoje e:</h2>
<?php echo $special; ?>
</body>
</html>
```

Os chamados *loops* também estão disponíveis no PHP. O mais simples deles é o `while` que executa ações repetidas vezes até que a condição contida no argumento seja válida (TRUE). A condição é avaliada no início de cada interação. Dois exemplos simples e que produzem a mesma saída são mostrados a seguir:

Figura 2.18: Um *loop* simples

```
<?php
// exemplo 1
$i = 1;
while ($i <= 10)
{
    echo $i++;
}
// exemplo 2
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
/$
```

O *loop* mais complexo é o `for`. Seu funcionamento é semelhante ao homônimo na linguagem C. Compõe-se de três expressões. A primeira é a inicialização da variável de controle do laço. A segunda é a condição de parada. Enquanto esta condição for válida, os comandos contidos no `for` se repetem. A terceira é o incremento da variável de controle para o próximo ciclo. A código 2.19 ilustra a utilização do `for` através de vários exemplos:

Figura 2.19: Exemplos de laços `for`

```
<?php
// exemplo 1
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
// exemplo 2
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
```

```
}
// exemplo 3
$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}
// exemplo 4
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
/$
```

Outra estrutura de controle bastante utilizada em PHP é o `include`, código 2.20. Esta estrutura inclui e executa um determinado arquivo dentro de outro. Os arquivos são incluídos especificando-se seu caminho. Caso o arquivo não seja encontrado, um *warning* é emitido para o usuário. Se especificado apenas o nome, o arquivo é procurado no diretório corrente.

Um estrutura similar é o `require`. A diferença básica é que em caso de erro (arquivo não encontrado, por exemplo) o *script* que originou a chamada `require` não continua, pois um erro fatal é emitido.

Figura 2.20: Exemplo de como usar o comando `include()`

```
vars.php
<?php
$cor = 'verde';
$fruta = 'laranja';
?>

teste.php
<?php
echo "Uma $cor $fruta"; // Uma verde laranja
include("vars.php");
echo "Uma $cor $fruta"; // Uma laranja verde
?>
```

2.9 Funções

Durante o desenvolvimento de uma determinada aplicação, ou mesmo dentro de um mesmo *script*, é comum haver a necessidade de repetição de partes do código. Esta tarefa é cansativa e dificulta a manutenção do código de aplicações mais extensas. As linguagens de programação modernas, disponibilizam formas de realizar o **reaproveitamento** de código. Uma maneira de fazer isso é utilizando funções.

De maneira simplificada, pode-se dizer que uma função é um subprograma que executa diversas instruções sempre que é chamada. Uma função pode ou não retornar valores.

A criação (definição) de uma função em PHP é bastante simples, conforme mostrado a seguir:

```
function nomefuncao(arg1, arg2, arg3)
{
comandos
}
```

O nome da função não pode coincidir com nenhuma palavra reservada do PHP (como nomes de funções já existentes na linguagem). A definição de funções pode estar em qualquer parte do *script*, mesmo após a sua chamada. Abaixo, temos um exemplo simples de uma função do usuário que mostra o quadrado dos números de 1 a 10.

```php
<?php
echo "FUNCAO DO USUARIO";
quadrado();
echo "FIM";
function quadrado()
{
        for($i=0;$i<10;$i++)
                echo "O quadrado de " . $i . " e " . "($i*$i)" . "<br>";
}
?>
```

O fragmento de código 2.9 mostra a definição e o uso da função `quadrado()` criada pelo desenvolvedor. É interessante observar que não é passado nenhum valor (argumento) para a função, ou seja, nenhuma informação entre parênteses. Outro detalhe que deve ser observado é que esta função não retorna valores, ou seja, na chamada da função, simplesmente o código é executado e o resultado substituído no local do *script* onde a função foi chamada.

Em muitas situações é necessário enviar um ou mais valores para uso da função. Assim, uma função pode não ter argumentos, como no exemplo anterior, ou ter vários argumentos declarados para uso. Para isso, basta informar quais os argumentos na declaração da função, conforme exemplo 2.9:

```php
<?php
function area($b,$h)
{
        $area = $b * $h;
        printf("Area do quadrado e: %s m2",$area);
}
echo "Area do Retangulo";
area(2,4);
echo "FIM";
?>
```

A função `area()` recebe dois argumentos, $b e $h, que representam a base e a altura de um retângulo. A função recebe esses argumentos e os utiliza para calcular a área do retângulo que é o objetivo da função. Para criar funções que retornem valores, basta utilizar a função `return`, conforme mostrado em 2.9.

```php
<?php
function area($b,$h)
{
        $a = $b * $h;
        return $a;
}
echo "Area do Retangulo";
$base = 2;
$altura = 3;
$area = area($base,$altura);
printf("A area do retangulo de base %f e altura %f e %f",$base,$altura,$area);
?>
```

2.10 Resumo

Nesta aula foram apresentados os conceitos iniciais e exemplos básicos para utilização da linguagem PHP. Na seção 2.1 é mostrado que o PHP é mais um acrônimo para compor o dicionário dos programadores e desenvolvedores *web*. Na seção 2.2 é demonstrada como acontece a interação entre os elementos (navegador, servidor *web* e PHP) envolvidos. Já na seção 2.3 são apresentados os primeiros comandos da linguagem e exemplos de uso. Os tipos de dados disponíveis na linguagem e os operadores são apresentados nas seções 2.4 e 2.5. O uso de formulários é introduzido na seção 2.6 bem como os principais métodos HTTP utilizados e sua recomendação de uso. Diversos exemplos

de comparação de variáveis e controle de fluxo são mostrados nas seções 2.7 e 2.8, respectivamente. Finalmente, o uso de funções é demonstrado na seção 2.9.

2.11 Exercícios

1. Qual o método HTTP recomendado para utilização em aplicações *web*? Justifique.

2. Execute os códigos mostrados no capítulo 2. Releia as explicações dadas e tente entender o funcionamento dos *scripts*.

3. Escreva um formulário que contenha dois campos: Nota 1 e Nota 2. Acrescente um botão de envio. Aponte o formuluário para enviar os dados para media.php, mostrado a seguir. Execute o *script* 2.21 e observe seu funcionamento.

Figura 2.21: media.php

```php
<?php
// media.php
$n1 = $_GET[nota1];
$n2 = $_POST[nota2];

$media = ($n1 + $n2)/2;
printf("A media das notas e:%s",$media);
?>
```

4. Sobre o exercício anterior, responda:

   - A variável `$media` é do tipo inteiro (int) ou ponto flutuante (float)?
   - Como modificar a exibição do valor da média para apenas uma casa decimal?
   - Faça as adequações necessárias para utilizar o método `POST`.

5. Para calcular a área de um retângulo basta multiplicar o valor da base (`b`) pela altura (`h`). Crie um arquivo em HTML que contenha um formulário que solicite ao usuário os valores de base e altura de um retângulo e um botão de envio. Em seguida, escreva um *script* para calcular a área do retângulo.
   (**Lembrete:** O formulário deve apontar para o *script* que realizará o processamento)

6. Repita o exercício anterior acrescentando o cálculo do perímetro do retângulo, ou seja, a soma da medida de todos os lados.

7. A fórmula para calcular a área de uma circunferência é: $A = \pi * r^2$. Elabore uma aplicação (formulário e *script*) que solicite ao usuário o valor do raio de uma circunferência e exiba o valor da área.

8. Dado o código em PHP a seguir, faça o que se pede:

   - Pesquise o que fazem as funções `substr()` e `strlen()`;
   - Execute o *script* e verifique a resposta apresentada;
   - Refaça o *script* utilizando um `while` ao invés do `for`.

Figura 2.22: countchar.php

```php
<?php
$text="Bem vindo ao desenvolvimento em PHP";
$searchchar="e";
$count="0";

for($i=0; $i<strlen($text); $i=$i+1)
{
    if(substr($text,$i,1)==$searchchar)
        $count=$count+1;
```

```
10 }
11
12 echo $count;
13 ?>
```

9. Modifique o código mostrado em 2.11 de forma que caso não sejam passados os valores das variáveis `nome` e `idade`, seja exibida uma mensagem explicativa ao usuário. Caso contrário, a mensagem informando o nome e a idade é mostrada.

10. Faça uma pesquisa para determinar a utilidade da função `phpinfo()`. Em seguida, escreva um *script* que utilize esta função, execute-o e veja o resultado.

11. Na questão anterior, o que acontece quando o valor passado em nome é ”ana silva”ao invés de ”ana”? Como evitar o truncamento dos dados ao utilizar *query strings*?

12. Crie um *script* que exiba informações sobre a máquina em que está sendo executado o sistema, como nome do servidor, endereço IP, a porta do servidor *web*.
(**Sugestão**: Pesquise sobre o *container* `$_SERVER`, semelhante a `$_GET` e `$_POST`).

Aula 3

Aprofundando em PHP

Educai as crianças, para que não seja necessário punir os homens.
Pitágoras

Objetivos

- Compreender e aplicar o conceito de *cookies*;
- Compreender e aplicar o conceito de sessões;
- Conhecer exemplos de aplicação de *cookies* e sessões em conjunto com banco de dados MySQL.

3.1 Por que usar *cookies* ou sessões?

No capítulo anterior, foi feita uma apresentação geral da linguagem PHP e sua dinâmica básica de funcionamento através do uso de formulários e dos *arrays* `$_GET` e `$_POST`, também conhecidos como *arrays* superglobais. Porém, o uso dos valores contidos nesses *arrays* existe apenas para o *script* no qual os dados foram diretamente repassados. E na maioria das vezes, os valores das variáveis devem permanecer existindo durante a execução de vários *scripts*. É o caso de aplicações acessíveis apenas a certos usuários, ou seja, que necessitam de autenticação, por exemplo. Os *cookies* podem ser utilizados para esta tarefa, também conhecida como de persistência dos dados. O PHP oferece ainda o recurso de sessões que é uma alternativa mais moderna do que os *cookies*.

Quando se acessa uma página web, a comunicação entre o navegador e o servidor web é feita através do protocolo HTTP (*Hypertext Transfer Protocol*). A questão é que o HTTP não armazena informações de estado, ou seja, os dados enviados para atender uma requisição do cliente não são mantidos quando uma nova requisição é feita, inviabilizando o armazenamento de informações enviadas anteriormente.

Por exemplo, se dois usuários acessam a página teste.html não é possível para o servidor determinar qual dos usuários realizou o acesso. As requisições são tratadas independentemente. Simplesmente, o Apache atende as duas requisições. Assim, mecanismos como *cookies* e sessões são fundamentais para o desenvolvimento de aplicações como comércio eletrônico, exibição de anúncios e personalização de páginas em geral. São as chamadas páginas dinâmicas. Diferentemente das páginas estáticas que utilizam puramente o HTML, ou seja, o mesmo conteúdo é sempre exibido para qualquer usuário que requisite a página.

Tabela 3.1: Principais parâmetros dos *cookies*

Parâmetro	Descrição
nome	Indica o nome que será usado como referência
valor	Valor atribuído ao cookie. Se vazio o cookie é excluído
validade	Define o tempo de validade do cookie no formato de tempo UNIX
caminho	Caminho no servidor para o qual o cookie estará disponível
domínio	Domínio para o qual o cookie está disponível
seguro	Valor (0 ou 1) que indica se o cookie é seguro

3.2 *Cookies*

Cookies são apenas arquivos texto armazenados localmente na máquina do cliente web para posteriormente serem recuperados pelo servidor. São compostos por um nome, que serve como referência e um valor associado a esse nome. Qualquer aplicação que necessite compartilhar dados entre diferentes páginas pode ser implementada utilizando os *cookies*.

Um exemplo muito simples do uso de *cookies* é a contagem de acessos. Supondo que um usuário acesse um *site* pela primeira vez e que este *site* tenha um *cookie* é definido com o nome `numero_acessos` e valor inicial 1, no próximo acesso, o valor é incrementado. Assim, a cada acesso o servidor resgata esse valor na máquina do cliente e pode utilizar a informação para exibir algo como:

```
Este é seu acesso número 2
```

Outro exemplo que pode ser citado é quando são feitas buscas em um *site*. O termo pesquisado pode ser armazenado em um *cookie* de forma que em acessos futuros o termo pesquisado já seja preenchido no campo do formulário.

Os *cookies* também possuem uma característica interessante chamada de validade, ou seja, o tempo que permanecerá armazenado na máquina do usuário. A validade de um *cookie* pode chegar a dias ou pode ser configurado para ter tempo de vida de apenas alguns minutos.

Existe uma discussão em relação à falta de privacidade dos *cookies*. Isso ocorre porque, muitas vezes, o usuário pode estar tendo informações gravadas em *cookies* e estas informações serem utilizadas por empresas para rastrear os sites pela qual o usuário navegou, por exemplo. Por isso, os navegadores permitem ao usuário habilitar ou desabilitar o armazenamento de *cookies* na máquina. Entretanto, ao desabilitar este recurso, dificilmente pode-se obter os mesmos recursos disponibilizados pelo site. Provalmente, não será possível realizar o *login* ou registrar a cidade em que o usuário se encontra e coisas do tipo.

3.2.1 Enviando *cookies*

O PHP disponibiliza a função `setcookie()` para enviar *cookies* para a máquina do usuário. Esta função possui dupla funcionalidade: tanto serve para definir como para excluir um *cookie*. A sintaxe básica da função é a seguinte:

```
bool setcookie(string nome [, string valor [,int validade [, string caminho [,
                    string domínio [, int seguro]]]]])
```

Para definir um *cookie*, é necessário atribuir um valor. Caso contrário, o *cookie* é excluído. Assim, a definição de um *cookie* é feita da seguinte forma:

```
setcookie ("nome", "Ana")
```

Enquanto a exclusão deve ser realizada assim:

```
setcookie ("nome").
```

Para criar um *cookie* válido por dois dias, com o auxílio da função `time()`,

```
setcookie ("nome", "Ana",time()+172800).
```

O valor de 172800 é equivalente a quantidade de segundos em 48 horas. Acrescentadando esse valor ao tempo atual retornado por `time()` temos a validade do *cookie* definida para o período de 2 dias.

Um detalhe importante quando se está utilizando *cookies* é que seu envio deve ocorrer antes que qualquer tag HTML. Caso o *cookie* seja enviado após as tags, uma mensagem de erro é exibida informando que a função `setcookie()` foi chamada após o envio dos cabeçcalhos. Assim, o valor não é armazenado nem o *cookie* é criado.

3.2.2 Mais um superglobal

A função *setcookie()* realiza a tarefa de enviar um *cookie* para o computador do usuário. Agora, é preciso saber como utilizar os valores armazenados. É importante lembrar que os *cookies* não poderão ser utilizados na mesma página que os criou. Apenas na próxima requisição é que será possível utilizá-los.

A maneira recomendada de acessar os *cookies* é através do array superglobal `$_COOKIE`, juntamente com a chave associativa desse *array*. Por exemplo, se o *cookie* chamado "nome"é definido,

```
setcookie ("nome", "Ana");
```

Na próxima requisição de página, o valor poderia ser acessado atribuindo-se o valor do *cookie* para uma variável, como mostrado a seguir.

```
$var = $_COOKIE["nome"];
```

Outra maneira de acessar o valor de um *cookie* é através da ativação diretiva *register_globals* no arquivo de configuração do PHP (`php.ini`). Nesse caso, o próprio nome do *cookie* é convertido em uma variável. No exemplo anterior, o valor do *cookie* estaria acessível através da variável `$nome`.

Apesar da facilidade, esta prática não é mais recomendada pelos desenvolvedores desde a versão 4.2.0 do PHP. O uso desta configuração permite que códigos inseguros sejam interpretados pelo PHP. O que obviamente não é desejável. Em Usando a diretiva Register Globals é mostrado os riscos que o uso desta diretiva trazem. Tanto que a partir da versão 6.0.0 ela será totalmente abolida.

3.2.3 Autenticando

Uma utilização bastante comum dos *cookies* é a autenticação de usuários. A autenticação é uma maneira de se certificar de que somente usuários autorizados por senha possam acessar determinadas áreas de um site ou mesmo todo o site. A autenticação com *cookies* pode ser dividida em três etapas:

- Página de *login* para receber os dados do usuário, verificar o cadastro e criar os *cookies* que permitem a utilização da página;
- Rotina de validação para ser utilizada nas páginas restritas;
- Página de *logout* para excluir os *cookies* criados inicialmente.

Para realizar a autenticação dos usuários, é necessário ainda uma ferramenta auxiliar para armazenamento dos dados dos usuários que terão acesso ao *site*, tais como nome, senha, email, endereço e qualquer outra informação que se queira armazenar. Para isso, o mais recomendado é a utilização de um servidor de banco de dados. Para exemplificar, utilizaremos o MySQL, bastante popular e adequado para as necessidades desta disciplina.

SGBD

Os bancos de dados são fundamentais para as aplicações *web*, pois são responsáveis pelo armazenamento de todo tipo de informação de maneira persistente. Para utilizar um SGBD (Sistema Gerenciador de Bancos de Dados), o primeiro passo a ser dado é a criação das tabelas que serão utilizadas. Na seção 3.4, o uso dos bancos de dados será abordado em mais detalhes.

Para o caso da autenticação do usuário apenas uma tabela é necessária que deverá armazenar, no mínimo, o nome do usuário e a senha. Os demais dados são opcionais e dependem da complexidade e do nível de segurança exigido pela aplicação. A listagem 3.1 exibe o arquivo em liguagem SQL (*Structured Query Language*) que cria a tabela `usuarios` e o comando `insert` para inserir um registro na tabela.

Figura 3.1: usuarios.sql

```
CREATE TABLE usuarios
(
        username varchar(10) NOT NULL,
        senha varchar(10) NOT NULL,
        nome varchar(80) NOT NULL,
        email varchar(80) NOT NULL,
        primary key(username)
);
INSERT INTO usuarios values ('ana','teste','Ana Silva','ana@dominio.com.br');
```

Para facilitar o entendimento do código, a listagem 3.4 mostra um arquivo de *include* (arquivo com código em PHP para ser anexado em outras páginas) para realizar a conexão com o servidor de banco de dados MySQL. Sempre que for realizado algum acesso ao banco de dados, será necessário chamar este *include* (`conecta_mysql.inc`).

Esses são os elementos fundamentais para ter acesso ao MySQL: a tabela e o *script* de conexão. A primeira armazena os dados dos usuários que se pretende acessar e o *script* realiza a conexão com o MySQL. Qualquer aplicação *web* a ser desenvolvida necessitará destes elementos.

Figura 3.2: conecta_mysql.sql

```
<?php
// configuracoes do banco de dados
$servidor = "localhost";
$usuario_bd = "ana";
$senha_bd = "123456";
$banco = "bd";
$con = mysql_connect($servidor, $usuario_bd, $senha_bd);
mysql_select_db ($banco);
//£
?>
```

Login

Passando a parte da autenticação propriamente dita é necessário criar um formulário inicial que será exibido ao usuário solicitando seu *username* e senha. As informações são verificadas e, no caso de estarem corretas os respectivos *cookies* são criados.

Nessa etapa, temos a criação de apenas dois arquivos: login.html e login.php. O primeiro contém o formulário inicial que será exibido para o usuário e o segundo realiza a lógica da autenticação.

Figura 3.3: conecta_mysql.sql

```
<html>
<body>
<form method="POST" action="login.php">
  <p align="center">Nome de usuario: <input type="text" name="username" size="10">
  <p align="center">Senha: <input type="password" name="senha" size="10">
  <p align="center"><input type="submit" value="Enviar" name="enviar">
</form>
```

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

Figura 3.4: login.php

```
<?php
// obtem os valores digitados
$username = $_POST["username"];
$senha = $_POST["senha"];
// acesso ao banco de dados
include "conecta_mysql.inc";
$resultado = mysql_query("SELECT * FROM usuarios where username='$username'");
$linhas = mysql_num_rows ($resultado);
if($linhas==0)  // testa se a consulta retornou algum registro
{
        echo "<html><body>";
        echo "<p align=\"center\">Usuario nao encontrado!</p>";
        echo "<p align=\"center\"><a href=\"login.html\">Voltar</a></p>";
        echo "</body></html>";
}
else
{
        if ($senha != mysql_result($resultado, 0, "senha")) // confere senha
        {
                echo "<html><body>";
                echo "<p align=\"center\">A senha esta incorreta!</p>";
                echo "<p align=\"center\"><a href=\"login.html\">Voltar</a></p>";
                echo "</body></html>";
        }
        else   // usuario e senha corretos. Criando os cookies
    {
        setcookie("nome_usuario", $username);
        setcookie("senha_usuario", $senha);
        // direciona para a pagina inicial dos usuarios cadastrados
        header ("Location: index.php");
    }
}
mysql_close($con);
?>
```

Validação

Depois de efetuado procedimento de login/autenticação do usuário, os *cookies* já encontram-se armazenados na máquina do cliente e disponíveis para uso. A cada nova requisição de página é necessário fazer a validação, ou seja, verificar se os *cookies* ainda são válidos e se as informações do usuário ainda estão corretas. Isso deve ser feito para evitar que um usuário mal intencionado e que não tem acesso a determinadas áreas do site digite o endereço diretamente no navegador e tenha acesso a página. Para isso, basta criar um *include* `valida_cookies.inc` que deve ser chamado pelas páginas protegidas.

Figura 3.5: `valida_cookies.inc`

```
<?php
if(IsSet($_COOKIE["nome_usuario"]))
    $nome_usuario = $_COOKIE["nome_usuario"];
if(IsSet($_COOKIE["senha_usuario"]))
    $senha_usuario = $_COOKIE["senha_usuario"];

if(!(empty($nome_usuario) OR empty($senha_usuario)))
{
    include "conecta_mysql.inc";
        $resultado = mysql_query("SELECT * FROM usuarios WHERE username='$nome_usuario'"
        if(mysql_num_rows($resultado)==1)
```

```
12          {
13                  if($senha_usuario != mysql_result($resultado,0,"senha"))
14                  {
15                  setcookie("nome_usuario");
16                  setcookie("senha_usuario");
17                  echo "Voce nao efetuou o LOGIN!";
18                  exit;
19          }
20      }
21      else
22      {
23          setcookie("nome_usuario");
24          setcookie("senha_usuario");
25          echo "Voce nao efetuou o LOGIN!";
26          exit;
27      }
28  }
29  else
30  {
31      echo "Voce nao efetuou o LOGIN!";
32      exit;
33  }
34  mysql_close($con);
35  ?>
```

Observe que nas primeiras linhas da listagem 3.5 a função `isset()` testa a existência dos *cookies* antes de obter seus valores. Assim, evita-se a exibição de mensagens de erro ao tentar acessar elementos que não existem. Caso seja verificado a inexistência dos *cookies*, uma mensagem de erro é exibida (linha 31) e o *script* é finalizado. Se o nome do usuário ou senha não forem vazios (linha 7), é realizada uma busca no banco de dados pelo usuário especificado e em seguida a senha é comparada. Se a senha estiver errada, os *cookies* são destruídos. Caso contrário, a conexão é encerrada e a página está validada.

Logout

O último passo consiste em construir um *script* para *logout* que é importante que exista para evitar que quando o usuário continua navegando por outros sites, mas não mais necessite utilizar o *site* que estava logado. O que é fundamental em máquinas que são utilizadas por vários usuários. A principal ação deste *script* é destruir os *cookies* criados durante o *login* e direcionar o usuário para a página de login novamente. A listagem 3.6 exemplifica estas ações. Basicamente são utilizadas as funções `setcookie()` que serve tanto para criar como para destruir *cookies* e a função `header()` para direcionar o usuário para a página login.html.

Figura 3.6: logout.php

```
1  <?php
2          setcookie("nome_usuario");
3          setcookie("senha_usuario");
4          header ("Location: login.html");
5  ?>
```

3.3 Sessões

Uma sessão é o tempo na qual o usuário permanece navegando em um determinado *site*. Ao iniciar a navegação, uma sessão é aberta e diversas variáveis podem ser registradas e acessadas durante o tempo de duração da sessão, ou seja, enquanto a sessão estiver **aberta** ou ainda a sessão for **válida**.

Da mesma forma que os *cookies*, as sessões permitem que os dados armazenados nas variáveis sejam compartilhados entre as páginas. Porém, existem algumas diferenças entre *cookies* e sessões. Os primeiros armazenam dados na própria máquina do usuário podendo continuar existindo mesmo

quando a navegação é finalizada. Já as variáveis de sessão são armazenadas no servidor e são destruídas ao sair do site ou quando o navegador é fechado.

A primeira informação importante relacionada a uma sessão é justamente o seu *id* utilizado como identificador da sessão. Para utilizar os dados armazenados em uma sessão é necessário conhecer esse número. O compartilhamento deste valor pode ser feito através de *cookies* ou através da própria URL.

O primeiro método só poderá ser utilizado se os *cookies* estiverem habilitados na máquina. Como nem sempre isto acontece, o segundo método pode ser utilizado.

O uso da URL é mais sofisticado. A partir da versão 4.2.0 do PHP, está habilitada a opção `enable-trans-sid` que permite que a identificação da sessão seja enviada transparentemente para as páginas, não sendo necessário o uso de *query strings*.

A principal função utilizada para o uso de sessões em PHP é `session_start()`. Serve tanto para criar uma nova sessão quanto para ter acesso às variáveis de uma sessão existente.

3.3.1 Registrando variáveis

Registrar uma variável em uma sessão significa disponibilizar esta variável para qualquer página enquanto a sessão for válida. A maneira mais segura de registrar variáveis é utilizando o array superglobal `$_SESSION`.

Na listagem 3.7 é mostrado um exemplo de como realizar a inicialização e o registro de algumas variáveis. Já na listagem 3.8, os dados registrados na listagem anterior são lidos. A primeira função utilizada em ambas as páginas é `session_start()`. Depois, as variáveis de sessão `nome`, `sobrenome` e `idade` são inicializadas e um link para a página seguinte é mostrado para o usuário. Na página seguinte, os dados são restaurados com `session_start()` e os valores das variáveis são acessados com o array `$_SESSION`.

Figura 3.7: writes.php

```
1 <?php
2 session_start();
3 echo 'Escrevendo os dados da sessao';
4 $_SESSION['nome'] = 'Ana';
5 $_SESSION['sobrenome'] = 'Maria';
6 $_SESSION['idade'] = 23;
7
8 printf("echo '<br><a href="readses.php?'. SID .'">Nova pagina</a>';
9 ?>
```

Figura 3.8: reads.php

```
1 <?php
2 session_start();
3 $nome = $_SESSION['nome'];
4 $sobrenome = $_SESSION['sobrenome'];
5 $idade = $_SESSION['idade'];
6
7 printf("Ola sou %s %s e tenho %d anos de idade<br>",$nome,$sobrenome,$idade);
8 ?>
```

Quando uma variável não é mais útil, pode-se eliminá-la com a função `unset($_SESSION['nome'])`. Assim, a variável de sessão especificada deixa de existir.

3.4 Bancos de Dados

A grande maioria das aplicações necessita de persistência dos dados, ou seja, mesmo depois que o sistema é desligado, os dados devem permanecer até que o sistema seja religado. Entretanto, sabe-se que, depois que um processo é finalizado, todas as variáveis e demais informações relacionadas a ele são removidas da memória. Daí vem a necessidade de se utilizar arquivos para garantir a persistência dos dados.

No caso de aplicações que compartilham informações, apenas o uso de arquivos certamente acarretará em inconsistência dos dados, por isso, é necessário utilizar uma ferramenta auxiliar: um SGBD. Trata-se de um conjunto de aplicativos responsáveis pelo gerenciamento das informações em uma base de dados. O principal objetivo é livrar a aplicação cliente da responsabilidade de gerenciar o acesso, manipulação e organização dos dados. O SGBD disponibiliza uma interface para que os seus clientes possam incluir, alterar ou consultar dados.

Exitem diversos gerenciadores de bancos de dados, alguns mais famosos e voltados para grandes volumes de dados como o ORACLE. Há ainda alternativas mais populares para aplicações menores como SQL Server, PostgreSQL e MySQL. Praticamente todos os gerenciadores de bancos de dados citados possuem versões para a *web* ou foram criados especificamente para esta finalidade, como o MySQL.

As informações em uma base de dados estão distribídas em tabelas. Cada uma das tabelas possui um nome pelo qual ela é referenciada e diversos campos (colunas). Na listagem 3.1, temos os comandos da linguagem SQL que criam a tabela de nome **usuarios** com quatro campos (username, senha, nome e email). Cada campo armazenará as informações associadas ao nome do campo.

Uma base pode conter diversas tabelas para armazenar os dados. De acordo com o tamanho e a complexidade da aplicação, uma base de dados pode conter outros elementos mais sofisticados que facilitam o uso e melhoram o desempenho. Para a grande maioria das aplicações, gerenciadores como o PostgreSQL e o MySQL atendem perfeitamente às necessidades.

3.4.1 Criando tabelas

Antes de qualquer ação ser realizada é necessário criar uma base de dados e nesta base de dados criar as tabelas. Cada campo de uma tabela deve receber um atributo informando o tipo de dado que será armazenado. Considerando o MySQL como gerenciador de banco de dados, existem vários tipos de dados disponíveis. Na tabela 3.2, são mostrados alguns deles. Para uma lista completa dos tipos de dados, basta consultar o manual do usuário.

Tabela 3.2: Alguns tipos de dados disponíveis no MySQL

Tipo	**Descrição**
`TINYINT`	Inteiros de 1 *byte*
`SMALLINT`	Inteiros de 2 *bytes*
`MEDIUMINT`	Inteiros de 3 *bytes*
`INT`	Inteiros de 4 *bytes*
`VARCHAR`	Texto de até 256 caracteres
`DATE`	Datas em formatos diversos
`FLOAT`	Números em ponto flutuante de até de 4 *bytes*

3.4.2 Tabela exemplo

A tabela 3.3 mostra uma amostra dos campos que a tabela de exemplo **contatos** deve possuir para armazenamento das informações. Observe que deve haver uma relação entre os campos que são necessários (tabela 3.3) e os tipos de dados disponíveis (tabela 3.2).

Tabela 3.3: Tipos de dados usados na tabela contatos

Nome do Campo	**Tipo**	**Tamanho**	**Descrição**
id	`INT`	6	Identificador único para cada registro
nome	`VARCHAR`	15	Nome do contato
fone	`VARCHAR`	10	Telefone
email	`VARCHAR`	25	Email
nascimento	`DATE`		Data de aniversario

Para entender melhor a criação de tabelas, vejamos a listagem 3.9 exemplifica a utilização da linguagem SQL (*Structured Query Language*). Os comandos necessários para criação da tabela **contatos** segundo os campos mostrados na tabela 3.2 são mostrados. Logo na linha 1 é utilizado o comando `CREATE TABLE` que informa ao MySQL que a *query* a seguir é para criação de uma tabela e seu nome será **contatos**. Entre parênteses temos a lista dos campos que irão compor a tabela, seguidos dos respectivos tipos de dados. É interessante notar que no campo `id` (linha 3) além da diretiva `NOT NULL`, que informa que não serão aceitos registros com esse campo sem um valor, temos a diretiva `AUTO_INCREMENT`, que automaticamente incrementa o valor deste campo para novos registros. Na linha 8, é informado ainda qual o campo que será utilizado como chave primária da tabela, ou seja, o campo de valor único que é utilizado pelo MySQL para otimização e busca de dados.

Figura 3.9: contatos.sql

```
1 CREATE TABLE contatos
2 (
3         id int(6) NOT NULL AUTO_INCREMENT,
4         nome VARCHAR(15) NOT NULL,
5         fone VARCHAR(10) NOT NULL,
6         email VARCHAR(25) NOT NULL,
7         nascimento DATE NOT NULL,
8         PRIMARY KEY id,
9 );
```

3.5 Resumo

O foco desta aula foi a abordagem dos conceitos de persistência de dados (*cookies*, sessões e bancos de dados) em conjunto com a linguagem PHP. Na seção 3.1 é mostrada a importância do uso de *cookies* ou sessões. Na seção 3.2 é feito o aprofundamento deste recurso. Na seção 3.2.1 o uso dos *cookies* é exemplificado. Na seção 3.2.2 os principais *arrays* superglobais disponibilizados pela linguagem são apresentados. As sessões são apresentadas na seção 3.3 e a integração com o banco de dados MySQL na seção 3.4.

3.6 Exercícios

1. Quais as principais diferenças entre *cookies* e sessões? São funcionalidades equivalentes?

2. Qual a informação repassada pelos *arrays* conhecidos como super globais mais utilizados pela linguagem PHP?

   - `$_SERVER`
   - `$_GET`
   - `$_POST`
   - `$_SESSION`
   - `$_ENV`

3. O que seria exibido como saída do fragmento de código 3.10? Modifique o código para obter o mesmo resultado utilizando o *array* superglobal `$_ENV` ao invés da função `getenv()`.

Figura 3.10: Exemplo de uso da função `getenv()`

```php
<?php
// exemplo de uso da funcao getenv() do PHP
$usuario = getenv('USER');
$host = getenv('HOSTNAME');
$ip = getenv('REMOTE_ADDR');
printf("Bom dia %s",$usuario);
printf("Voce esta conectado ao host %s de IP %s",$host,$ip);
?>
```

4. Dado o fragmento de código a seguir, explique o que é executado em cada linha de código mostrada?.

Figura 3.11: Exemplo de uso de sessões

```php
<?php
// inicia a sessao
session_start();
// atribui valor as variaveis de sessao
$_SESSION['cor']='azul';
$_SESSION['tamanho']='medio';
$_SESSION['formato']='circular';
print "Feito";
//£
?>
```

5. Supondo que o código em 3.11 é executado corretamente, o que seria exibido após a execução do seguinte código?

Figura 3.12: Continuação de exemplo de uso de sessões

```php
<?php
// inicia a sessao
session_start();
// exibe as variaveis da sessao, a atribuicao e feita na pagina anterior
echo "A cor e ".$_SESSION['cor']."<br>";
echo "O tamanho e ".$_SESSION['tamanho']."<br>";
echo "O formato e ".$_SESSION['formato']."<br>";
/$
?>
```

6. Retirando as linhas 5, 6 e 7 do código 3.12 e acrescentando `print_r($_SESSION);` o que é exibido como saída?

7. Como alterar os valores das variáveis de sessão do código 3.12 (cor, tamanho e formato) para vermelho, grande e retangular? Escreva as linhas de código.

8. Explique em detalhes qual a relação entre as funções `unset()` e `destroy()`.

9. O *script* a seguir exemplifica como realizar a entrada de dados na tabela Contatos, citada em 3.4.2. A primeira parte trata da conexão ao MySQL propriamente dita. Em seguida, o comando SQL (*query*) é escrita e repassada ao MySQL. Sobre este *script*, responda:

   - O função `mysql_connect()` poderia ser substituída pela função `mysql_pconnect()`? Qual a diferença básica entre elas?
   - Qual o papel da função `mysql_query()`? Qual a saída do *script* em caso de inserção correta dos dados?
   - Explique em que circunstância a linha 16 é executada?

Figura 3.13: Exemplo de código para inserir dados em uma tabela

```
1  <?php
2  // usuario para conexao com banco
3  $username="username";
4  // senha para conexao com banco
5  $password="password";
6  // nome da base de dados
7  $database="agenda";
8  // conexao com o aplicativo servidor de banco de dados
9  mysql_pconnect(localhost,$username,$password);
10 // seleciona o banco de dados
11 @mysql_select_db($database) or die( "Impossivel selecionar a base");
12 // query
13 $query = "INSERT INTO contacts VALUES ('','Joao','01234567890',
14 'joaosilva@net.com','1976/11/06')";
15 // tratamento de erro
16 if(!$result = mysql_query($query))
17         echo mysql_error();
18 else
19         echo mysql_affected_rows($result)."dados inseridos com sucesso";
20 ?>
```

Aula 4

Estudo de caso: controle de finanças empresarial

"A vida é uma peça de teatro que não permite ensaios.
Por isso, cante, ria, dance, chore e viva intensamente cada momento de sua vida, antes que a cortina se feche e a peça termine sem aplausos"
Charles Chaplin

Objetivos

- Conhecer uma aplicação *web* completa;
- Ter uma referência para o desenvolvimento de outras aplicações de mesmo porte.

4.1 Determinando os objetivos da aplicação

Neste capítulo, vamos aprender a criar um sistema para controle de finanças de uma empresa. Este sistema será desenvolvido em PHP a partir dos diversos conteúdos que foram abordados ao longo deste material. Vamos verificar, na prática, a criação de formulários, manutenção de informações em *cookies* e seções, autenticação, transações com banco de dados, dentre outros assuntos. Dessa forma, espera-se que vocês possam praticar os conhecimentos adquiridos, tornando-se aptos a desenvolver sistemas próprios para as mais diversas aplicações.

Antes de iniciar a construção do sistema, o bom programador sempre dedica um período para estudo e planejamento das atividades a serem desenvolvidas. Nesta etapa, é interessante que se entenda bem todas as funcionalidades do sistema e que eventuais questionamentos sejam discutidos. O objetivo desta etapa é que não existam dúvidas e, consequentemente, todos os requisitos apontados durante a fase de modelagem sejam cumpridos.

A ideia básica do sistema que vamos desenvolver é controlar o orçamento mensal de uma empresa, permitindo ao administrador cadastrar o dinheiro que recebe (receitas) e o dinheiro que gasta (despesas). O sistema deve exibir também o saldo de cada mês, ou seja, a diferença entre as receitas e despesas. Dessa forma, esta ferramenta permitirá ao administrador visualizar melhor quanto sobra no caixa da empresa no final do mês, ou se ela está gastando mais do que ganha (saldo negativo).

Podemos classificar as receitas e as despesas em FIXAS e VARIÁVEIS, de acordo com o seu tipo. Para entender melhor essa divisão, vejamos a seguir o significado e alguns exemplos sobre cada um dos tipos.

- **RECEITAS FIXAS -** É aquele dinheiro que a empresa recebe todos os meses. Exemplos: direitos autorais, contratos de prestação de serviços, aluguéis (caso sua empresa tenha algum bem e esteja alugando-o), dentre outros.
- **RECEITAS VARIÁVEIS -** É aquele dinheiro que não se recebe todos os meses. Exemplos: serviços prestados eventualmente, premiações, venda de produtos, dentre outros.
- **DESPESAS FIXAS -** São aquelas que a empresa tem que pagar todos os meses. Exemplos: água, luz, telefone, funcionários do quadro efetivo, dentre outros.
- **DESPESAS VARIÁVEIS -** São aquelas que não são pagas todos os meses. Exemplos: manutenção de veículos, reformas no prédio, consertos de equipamentos, pagamento de funcionários temporários, dentre outros.

Para melhor contextualizar estes termos, vamos imaginar que uma empresa de aluguéis de imóveis possui sua fonte de renda fixa oriunda de contratos de locação de 20 casas no valor de R$ 500,00. Isso significa que, mensalmente, a empresa tem garantido o recebimento de R$ 10.000,00. No entanto, em um mês específico, a empresa alugou uma de suas casas de praia para um grupo de jovens durante o final de semana e, como consequência, recebeu R$ 800,00 a mais. Podemos classificar o capital derivado dos contratos como *receitas fixas*, pois todos os meses este dinheiro é recebido. No entanto, o valor referente a casa de praia não é certo, portanto deve ser alocado como *receitas variáveis*. Em relação às despesas, vamos supor que a empresa teve os seguintes gastos no mês de outubro:

1. R$ 800,00 do aluguel de seu escritório.
2. R$ 300,00 do pagamento de condomínio.
3. R$ 4.947,00 com pagamento dos funcionários de seu quadro efetivo.
4. R$ 592,00 da conta de energia elétrica.
5. R$ 118,00 da conta de água.
6. R$ 437,00 da conta de telefone e internet.
7. R$ 278,00 com combustível de seus veículos operacionais.
8. R$ 127,00 de manutenção de veículo.
9. R$ 299,00 com toner da impressora.

Entre essas despesas, podemos classificar as sete primeiras como *despesas fixas*, pois independente do mês, elas irão sempre ocorrer. Obviamente o valor dessas contas pode mudar, mas mensalmente algum dinheiro será destinado a elas. As duas últimas seriam *despesas variáveis*, visto que não é todo mês que o veículo necessita de manutenção ou que é preciso comprar novo toner para a impressora.

De forma simplificada, no mês de outubro esta empresa recebeu R$ 10.800,00 (contratos + locação da casa de praia) e teve despesas de R$7.898,00 (somatório das *despesas fixas* e *despesas variáveis*). Fazendo a diferença entre o total de receitas e o total de despesas, podemos obter o saldo final do mês:

SALDO = RECEITAS - DESPESAS

SALDO = R$ 10.800,00 - R$7.898,00

SALDO = R$ 2.902,00

Portanto, neste mês, esta empresa conseguiu pagar todas as suas contas e ainda sobrou R$ 2.902,00. Este saldo corresponde a aproximadamente 26% do total de receitas obtidas no mês de

outubro. Se este procedimento for realizado em todos os meses, é possível construir gráficos que permitam ao administrador observar os períodos do ano nos quais a empresa tem maior lucratividade ou que sofre prejuízos. Um exemplo de gráfico interessante seria apresentar o saldo, em termos de porcentagens, para cada mês. Para calcular esta informação, podemos utilizar a seguinte expressão:

$$SALDO(\%) = (\frac{RECEITAS - DESPESAS}{RECEITAS}) * 100. \quad (4.1)$$

Para criarmos uma aplicação em PHP que realize esse controle de finanças, primeiramente precisamos criar uma base de dados para armazenar as receitas e as despesas. Na seção 4.2, vamos descrever os procedimentos necessários para realizar essa atividade.

4.2 Criando a base de dados

Como já sabemos, a base de dados é responsável pelo armazenamento das informações. Quando o usuário do sistema cadastrar uma despesa, ou receita, estes dados terão que ser organizados de forma que seja possível recuperar a informação. Antigamente, era comum realizarmos esse procedimento por meio de formulários de papel, preenchendo campos com diversas informações. Com isso, sempre que estivéssemos buscando algo sobre algum assunto, recorríamos ao devido formulário e recuperávamos a informação. Quando não encontrávamos a informação desejada, ou por não existir o devido campo no formulário ou por simplesmente o usuário não haver preenchido, comumente o nosso trabalho era prejudicado. Daí sempre vinha em mente a angústia por não ter colocado esse campo ou não ter obrigado o usuário a preenchê-lo.

Em sistemas computacionais, a preocupação de se projetar o sistema para que exista um processamento de dados com organização e desempenho adequados deve ser um dos pilares da boa programação. Imagine o trabalho e a lentidão que seria para uma secretária obter informações de um aluno em centenas de formulários de papel cujos dados dos professores estivessem misturados com os dos alunos, sem qualquer ordem. Com certeza, isso tornaria o sistema lento e com total desagrado aos usuários. A eficiência do sistema, seja computacional ou manual, está diretamente relacionada à forma que a informação está organizada e, para realizar isso de forma coerente, o primeiro passo é identificar as entidades principais do nosso sistema.

Claramente podemos observar que o nosso sistema terá usuários. Daí vem a questão, quem serão os usuários? Qualquer pessoa que acesse o site pela internet? Ou vamos tornar o acesso restrito apenas a pessoas cadastradas? Quem poderá cadastrar outros usuários? No nosso caso, vamos considerar que o sistema será restrito a pessoas cadastradas e quem poderá cadastrar será somente o usuário administrador. Então, identificamos a nossa primeira entidade: usuários. Precisamos ter informações sobre os usuários. Pensando no que seria útil ao nosso sistema, seguem alguns dados sobre os usuários que devem ser armazenados:

- Nome de usuário para acesso ao sistema, conhecido por *login*;
- Senha para acesso ao sistema;
- Nome completo;
- Sexo;
- Identidade;
- CPF;
- Data de nascimento;
- Estado civil;
- Função que exerce na empresa;
- Email;

- Telefone para contato;
- Perfil (padrão/administrador).

Observe que todos os itens listados acima são referentes aos usuários. Não faz sentido algum misturarmos com dados de outras entidades como, por exemplo, despesas. Se estivéssemos elaborando um formulário de papel para cadastramento dessas informações de usuários, ele seria semelhante ao expresso na figura 4.1.

Figura 4.1: Exemplo de ficha para cadastramento de usuários.

CADASTRAMENTO DE USUÁRIOS

Dados Pessoais	**(*) Campos Obrigatórios**
Nome completo*	
CPF*	
Identidade	
Sexo*	( ) Feminino ( ) Masculino
Data de Nascimento (dd/mm/aaaa)*	
Estado civil*	( ) Solteiro ( ) Casado ( ) Separado ( ) Divorciado ( ) Viúvo ( ) União Estável
Função que exerce na empresa*	
E-mail*	
Telefone para contato*	
Administrativo	
Perfil*	( ) Padrão ( ) Administrador
Usuário administrador*	
Data e hora de cadastro*	

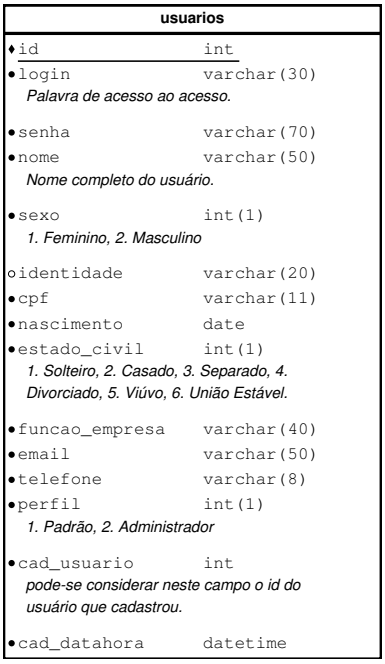
Analisando o formulário da figura 4.1 podemos perceber que todos os campos, exceto *identidade*, estão sinalizados com um asterisco (*). Isso significa que seu conteúdo é algo indispensável para o funcionamento coerente do sistema e, consequentemente, o usuário é obrigado a preenchê-lo. A informação relacionada à *identidade* tem a função apenas de complementar o cadastro. Quando falamos de registros armazenados em bancos de dados eletrônicos, também podemos obrigar o preenchimento de campos, sendo de responsabilidade do analista decidir o que é informação fundamental ou não.

Ainda analisando a figura 4.1, podemos observar a existência de campos com teor puramente administrativo. Trata-se de informações que o usuário não deve preencher, mas sim o sistema. No caso de formulários de papel, algum responsável irá cadastrar esses campos, sinalizando o perfil do novo usuário, data e hora de cadastro e o usuário que o atendeu. Essas informações são úteis em caso de problemas, como cadastramentos ilegais, dentre outros. O campo *perfil*, em nosso caso, também terá um papel bastante importante por informar se o novo usuário terá permissões para cadastrar outros novos usuários.

Na construção de nosso sistema PHP, cada entidade identificada deverá ser convertida em uma tabela do banco de dados MySQL. No caso dos *usuários*, vamos ter uma tabela como descrita a seguir:

Como discutido anteriormente, o campo *identidade* aparece na figura 4.2 representado por uma circunferência sem preenchimento. Isso mostra que esse campo foi pensado pelo analista como opcional. Os campos *sexo*, *estado_civil*, *perfil* e *cad_usuario* são modelados como inteiro, quando aparentemente deveriam ser do tipo *string*. Isso acontece porque, em algumas ocasiões, é mais prático armazenar números correspondentes que a própria *string*. No caso do *sexo*, *estado_civil* e *perfil* veremos, nas seções seguintes, como definir números para identificar a opção do usuário. Já para o

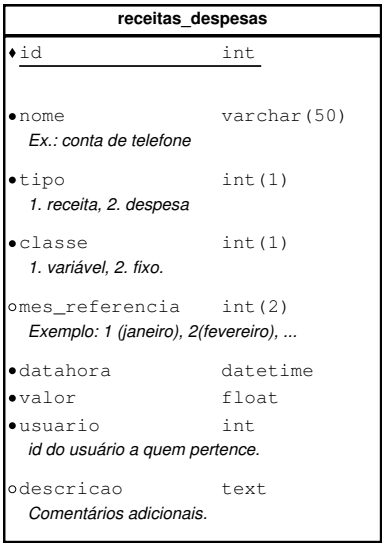
Figura 4.2: Tabela de usuários.



campo *cad_usuario*, vamos gravar o *id* do usuário que cadastrou. Caso seja necessário recuperar essa informação, basta fazer uma consulta na tabela de *usuários* filtrando pelo número de *id*.

Tendo como base o mesmo raciocínio empregado para a construção da tabela de *usuários*, vamos apresentar a outra tabela necessária ao nosso sistema.

Figura 4.3: Tabela de receitas e despesas.



Para que possamos dar início a criação de nossas tabelas, é necessário inicialmente realizarmos a criação da base de dados *exemplo_ntwi* na qual as informações serão armazenadas. No MySQL, podemos realizar este procedimento por meio do seu utilitário, como vemos na seguinte instrução:

Figura 4.1: código SQL para criação da base de dados *exemplo_ntwi*.

```
1 CREATE DATABASE exemplo_ntwi;
```

A tabela que irá armazenar os usuários do sistema, mostrada na figura 4.2, pode ser criada da seguinte forma:

Figura 4.2: código SQL para criação da tabela *usuarios*.

```
1 CREATE TABLE exemplo_ntwi.usuarios
```

```
(
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY COMMENT 'Chave primaria.',
        login VARCHAR(30) NOT NULL,
        senha VARCHAR(30) NOT NULL,
        nome VARCHAR(50) NOT NULL,
        sexo INT NOT NULL COMMENT '1. Feminino; 2. Masculino.',
        identidade VARCHAR(20) NULL COMMENT 'Apenas numeros.',
        cpf VARCHAR(11) NOT NULL COMMENT 'Apenas numeros.',
        nascimento DATE NOT NULL,
        estado_civil INT NOT NULL COMMENT '1. Solteiro; 2.Casado; 3.Separado;
        4. Divorciado; 5. Viuvo; 6. Uniao estavel.',
        funcao_empresa VARCHAR(40) NOT NULL,
        email VARCHAR(50) NOT NULL,
        telefone VARCHAR(8) NOT NULL COMMENT 'Apenas numeros.',
        perfil INT NOT NULL COMMENT '1. Padrao; 2. Administrador.',
        cad_usuario INT NOT NULL COMMENT 'Id do usuario que efetuou o cadastro.',
        cad_datahora DATETIME NOT NULL COMMENT 'Data e hora de efetivacao do cadastro.',
        UNIQUE (login, identidade, cpf)
);
```

Ao executar o código 4.2, podemos observar que a tabela é criada sem qualquer registro. No entanto, é necessário que o sistema tenha pelo menos um usuário. Dessa forma, vamos inserir um usuário chamado *admin*, com perfil administrador e senha *admin*. Será este usuário que efetuará o primeiro acesso e será responsável por iniciar o funcionamento do sistema. Por se tratar de uma senha padrão, é recomendado que o usuário administrador a substitua no primeiro processo de *login*.

Figura 4.3: código SQL para inserir o primeiro usuário do sistema.

```
INSERT INTO exemplo_ntwi.usuarios
(
        id, login, senha, nome, sexo, identidade, cpf, nascimento,
        estado_civil, funcao_empresa, email, telefone, perfil, cad_usuario,
        cad_datahora
)
VALUES
(
NULL, 'admin', 'admin', 'Administrador Padrao', '2', NULL,
'00000000000', '2011-08-09', '1', 'Administracao', 'admin@minhaempresa.com.br',
'00000000', '2', '1', '2011-08-09 17:44:54'
);
```

Seguindo os mesmos procedimentos empregados para criação da tabela de *usuarios*, podemos construir a tabela *receitas_despesas*, mostrada na figura 4.3, por meio do seguinte código SQL:

Figura 4.4: Código SQL para criação da tabela *receitas_despesas*.

```
CREATE TABLE exemplo_ntwi.receitas/despesas
(
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY COMMENT 'Chave primaria.',
        nome VARCHAR( 50 ) NOT NULL COMMENT 'Ex.: conta de telefone.',
        tipo INT( 1 ) NOT NULL COMMENT '1. Receita; 2. Despesa.',
        classe INT( 1 ) NOT NULL COMMENT '1. variavel; 2. Fixo.',
        datahora DATETIME NOT NULL,
        valor FLOAT NOT NULL,
        usuario INT NOT NULL COMMENT 'Id do usuario a quem pertence.',
        descricao TEXT NULL COMMENT 'Comentarios adicionais.'
);
```

O aplicativo PhpMyAdmin representa uma ferramenta bastante interessante para criação e gerenciamento de base de dados MySQL. Por meio do uso deste aplicativo, podemos criar visualmente bases de dados, tabelas, bem como realizar operações de inserção, exclusão, dentre outras.

4.3 *Include* de acesso ao banco de dados

Para que se possa trocar informações com a base de dados, é necessário primeiramente estabelecer uma comunicação com o banco. Como em várias páginas haverá essa necessidade, o ideal seria criarmos um *include* de acesso ao banco, para evitar a repetição de códigos e a manutenção do sistema. Esse *include* será nomeado como *conectar_banco.inc*, e será chamado por todas as páginas que precisem acessar o banco de dados.

Como o nosso objetivo neste estudo de caso é apresentar uma aplicação construída a partir dos conhecimentos abordados neste material, o acesso será realizado por meio da comando *mysql_connect* referente ao SGBD que estamos utilizando. Se você não quiser empregar comandos dependentes da base de dados, você pode fazer uso de classes de abstração para ter mais flexibilidade. Dessa forma, é possível mudar o SGBD do sistema com o mínimo de modificações no arquivo fonte. Um exemplo de biblioteca para implementação desse procedimento pode ser consultada em http://pear.php.net/db.

Figura 4.5: `conecta_mysql.sql`

```
<?php
// configuracoes do banco de dados
$servidor = "localhost";
$usuario_bd = "ana";
$senha_bd = "123456";
$banco = "exemplo_ntwi";
$con = mysql_connect($servidor, $usuario_bd, $senha_bd);
mysql_select_db ($banco);
//£
?>
```

As variáveis *$servidor*, *$usuario_bd*, *$senha_bd* e *$banco* devem ser alteradas de acordo com as configurações do seu servidor de dados.

4.4 Autenticação do usuário com sessões

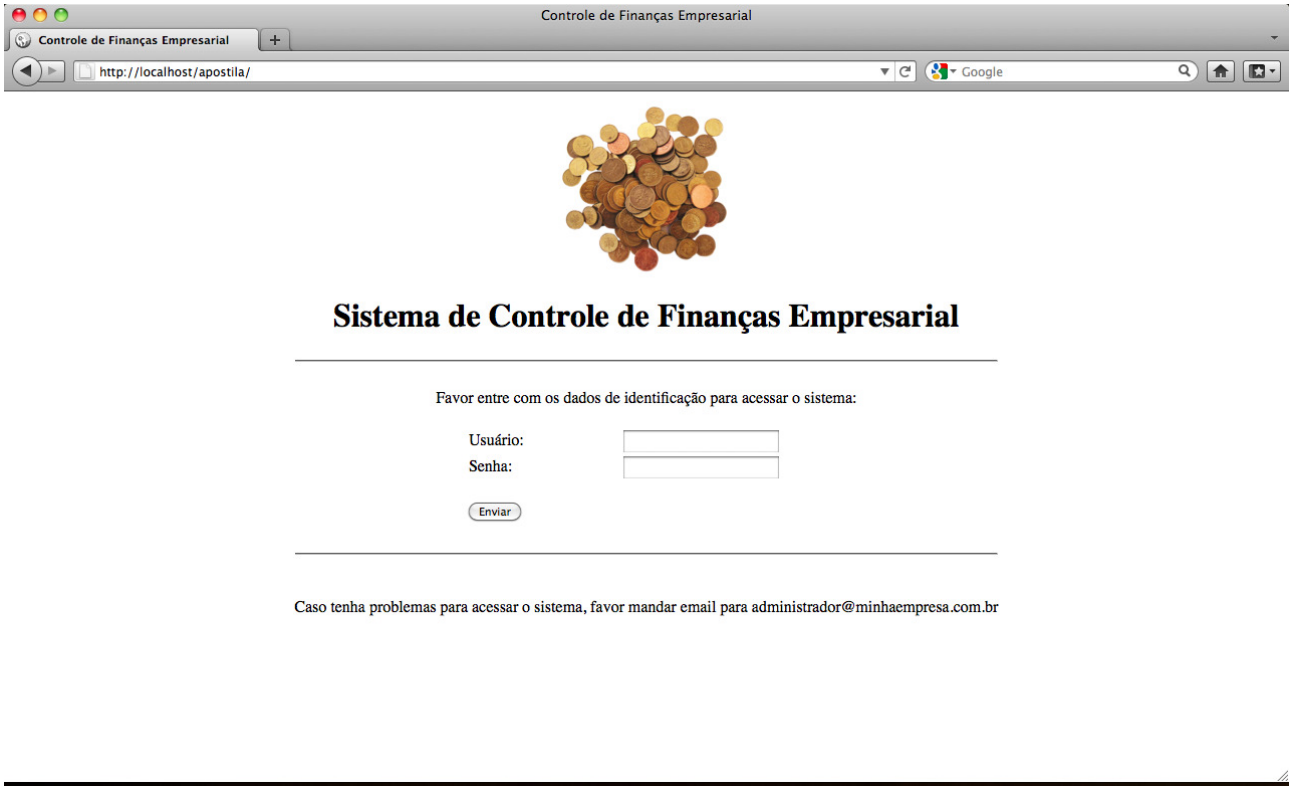
No capítulo 3 deste documento, aprendemos a construir controles de autenticação utilizando *cookies*. Nesta seção, vamos empregar uma nova abordagem por meio da utilização de sessões. Como vimos na seção 3.3, sessões são mecanismos que permitem que informações sejam registradas e utilizadas por várias páginas. Elas permitem armazenar tipos de dados mais complexos que os *cookies* e, além disso, há quem defenda que são mais seguras, pois os dados ficam armazenados no servidor e são destruídos quando o navegador é fechado.

A primeira página de nosso sistema será a de identificação do usuário. Ela será formada por um formulário HTML com os campos "Usuário" e "Senha", conforme apresentado na figura 4.4. Para construirmos a tela mostrada na figura 4.4, podemos utilizar o código HTML exibido em 4.6.

Figura 4.6: index.html

```
<html>
<head><title>Controle de Finan&ccedil;as</title></head>
<body>
<form method="POST" action="login.php">
  <center>
  <img src="receita_despesa.jpg" width="30%" height="30%"/>
  <h1>$$$ Sistema de Controle de Financas $$$</h1>
  <hr width="700px"  /><br />
  Favor entre com os dados de identificacao para acessar o sistema:
  <br /><br />
  <table>
        <tr>
                <td width="150px">Usuario: </td>
                <td width="200px"><input type="text" name="username" size="20">
                </td>
        </tr>
```

Figura 4.4: Tela de identificação dos usuários (*login*.




```
17          <tr>
18                  <td width="150px">Senha: </td>
19                  <td width="200px"><input type="password" name="senha" size="20">
20                  </td>
21          </tr>
22          <tr>
23                  <td><br /><input type="submit" value="Enviar" name="enviar">
24                  </td>
25          </tr>
26    </table>
27    <br /> <hr width="700px"  /><br />
28    <p>Caso tenha problemas para acessar o sistema, favor enviar
29    email para administrador@minhaempresa.com.br</p>
30    </center>
31 </form>
32 </body>
33 </html>
```

Conforme estudado na aula 3 deste material, um sistema de usuário/senha é formado por três partes básicas:

- Login - entrada do usuário no site. Nessa etapa, pode ser registrada uma sessões ou armazenados os dados do usuário em *cookies*;
- Autenticação - verificação da existência e validade da sessão ou *cookies*;
- Logout - saída do sistema. Nessa etapa a sessão é destruída ou os *cookies* são excluídos.

4.4.1 *Login*

Por meio do código HTML da página *index.php*, podemos notar que, ao preencher os campos necessários e clicar no botão *Enviar*, os dados dos usuários são submetidos a um programa chamado *login.php*. Esse algoritmo irá obter os dados enviados acessando o *array* superglobal `$_POST`, conforme estudado no capítulo 2. Em seguida, uma conexão com a base de dados deve ser aberta para

verificar a existência do usuário e a validade da senha. Veremos, portanto, como seria o código do programa *login.php* na Listagem 4.7.

Figura 4.7: login.php

```
1  <?php
2  // éobtm os valores digitados
3  $username = $_POST["username"];
4  // md5 - evitar que a senha do usuario seja armazenada limpa no banco.
5  $senha = md5($_POST["senha"]);
6  // acesso ao banco de dados
7  include "conecta_mysql.inc";
8  $resultado = mysql_query("SELECT * FROM usuarios where login='$username'");
9  $linhas = mysql_num_rows ($resultado);
10 if($linhas==0)  // testa se a consulta retornou algum registro
11 {
12         echo "<html><body>";
13         echo "<p align=\"center\">áUsurio ãno encontrado!</p>";
14         echo "<p align=\"center\"><a href=\"index.html\">Voltar</a></p>";
15         echo "</body></html>";
16 }
17 else
18 {
19         if ($senha != mysql_result($resultado, 0, "senha")) // confere senha
20         {
21                 echo "<html><body>";
22                 echo "<p align=\"center\">A senha áest incorreta!</p>";
23                 echo "<p align=\"center\"><a href=\"index.html\">Voltar</a></p>";
24                 echo "</body></html>";
25         }
26         else   // áusurio e senha corretos. Vamos gravar as çõinformaes na ãsesso.
27     {
28         $id = mysql_result($resultado, 0, "id");         // id do usuario.
29         $perfil = mysql_result($resultado, 0, "perfil"); // perfil do usuario.
30                 //Iniciar ãSesso.
31                 session_start();
32         $_SESSION['nome_usuario'] = $username;
33         $_SESSION['senha_usuario'] = $senha;
34         $_SESSION['perfil_usuario'] = $perfil;
35         $_SESSION['id_usuario'] = $id;
36        // direciona para a ápgina inicial do sistema.
37     header ("Location: principal.php");
38    }
39 }
40 //Encerrar ãconexo com o banco de dados.
41 mysql_close($con);
42 ?>
```

Vamos observar o que acontece nas linhas 3 e 5. Nestas linhas, os dados referentes ao *login* e a senha fornecidos pelo usuário são capturadas pelo nosso *script*. Como o método utilizado foi o POST, então os dados temos que utilizar o *array* superglobal $_POST. No entanto, se analisarmos atentamente as duas linhas, vamos perceber a presença de uma função, chamada *md5* sendo aplicada na informação de senha. O que isso significa? Calma, não se assustem! Essa função realiza um processamento necessário para esconder a senha do cliente que, como vocês sabem, tem que ser protegida. Por meio dessa função, cria-se uma palavra código relacionada a senha. Por exemplo, se a senha do usuário for 123456, ela será tratada por nosso sistema e armazenada na base de dados, como será visto adiante, como 81dc9bdb52d04dc20036dbd8313ed055. Ou seja, bem diferente da senha original garantindo, por consequência, melhores condições de segurança.

A linha 7 realiza a inclusão do código fonte 4.5. Logo após a conexão, realiza-se as operações para obter as informações necessárias para autenticação na tabela de usuários. Caso a linha 8 retorne algum registro, significa que o *login* do usuário em questão foi encontrado e o procedimento de autenticação, discutido no capítulo 3, pode ser iniciado. Então compara-se, por meio da linha 19, a senha fornecida com a senha armazenada na base de dados e, caso sejam iguais, as informações

de *login*, nome, senha e código do usuário (id) podem ser armazenados na sessão. Com o sucesso do procedimento de autenticação, a linha 37 direciona para a página principal do sistema. Caso o *login* ou a senha fornecida não estejam corretos, o código HTML das linhas 12 a 15 e 21 a 24 são devolvidos para o navegador.

4.4.2 Autenticação

Cada vez que uma página restrita do seu sistema tiver que ser acessada é necessário passar por um processo chamado de autenticação. O objetivo desse procedimento é evitar que qualquer usuário esperto digite, no campo de endereço do navegador, a URL de uma página restrita sem antes ter efetuado o *login*. O código 4.8, chamado valida_sessao.inc, valida a existência coerente da sessão e é empregado em nossa aplicação para a realização desta atividade.

Figura 4.8: `valida_sessao.inc`

```
1  <?php
2  session_start();
3
4  if(isset($_SESSION["nome_usuario"]))
5      $nome_usuario = $_SESSION["nome_usuario"];
6  if(isset($_SESSION["senha_usuario"]))
7      $senha_usuario = $_SESSION["senha_usuario"];
8
9  if(!(empty($nome_usuario) OR empty($senha_usuario)))
10 {
11     include "conecta_mysql.inc";
12         $resultado = mysql_query("SELECT * FROM usuarios WHERE login='$nome_usuario'");
13         if(mysql_num_rows($resultado)==1)
14         {
15                 if($senha_usuario != mysql_result($resultado,0,"senha"))
16                 {
17                 unset ($_SESSION['nome_usuario']);
18                 unset ($_SESSION['senha_usuario']);
19                 unset ($_SESSION['perfil_usuario']);
20                 echo "Voce nao efetuou o LOGIN!" . "<br />";
21                 echo "<p align=\"center\"><a href=\"index.html\">Voltar</a></p>";
22                 exit;
23         }
24     }
25     else
26     {
27         unset ($_SESSION['nome_usuario']);
28         unset ($_SESSION['senha_usuario']);
29         unset ($_SESSION['perfil_usuario']);
30         echo "Voce nao efetuou o LOGIN!";
31         echo "<p align=\"center\"><a href=\"index.html\">Voltar</a></p>";
32         exit;
33     }
34 }
35 else
36 {
37         echo "Voce nao efetuou o LOGIN!";
38         echo "<p align=\"center\"><a href=\"index.html\">Voltar</a></p>";
39         exit;
40 }
41 mysql_close($con);
42 ?>
```

Como podemos observar, a extensão dada para este arquivo é `inc`. Isto acontece porque este documento será incluído em outros vários arquivos para verificar se o usuário está autenticado, daí a padronização. No entanto, como estudado, o seu conteúdo não passa de códigos `PHP`.

Observe nas linhas 20 e 21, 30 e 31, 37 e 38 que se a senha fornecida estiver errada, o *login* não estiver cadastrado na base de dados ou se o *login* ou a senha não existirem na sessão, o usuário é

convidado a ir para página inicial do sistema.

4.4.3 *Logout*

O procedimento de *logout* consiste em o usuário deixar o sistema, apagando todas as suas informações gravadas na sessão. Este processo é bastante importante para evitar acessos indevidos e operações não autorizadas. Muitas vezes, nós usuários, esquecemos de sair corretamente do nosso email ou do sistema de compra virtual que estávamos acessado e, caso alguém mal intencionado use o computador logo em seguinda, isso pode trazer muitas dores de cabeça. Nosso sistema deve oferecer aos nossos usuários a opção de efetuar o *logout* de forma apropriada. Para isso, foi elaborado o código 4.9. Vamos estuda-lo agora.

Figura 4.9: `logout.php`

```
1 <?php
2 session_start();
3 $_SESSION = array();
4 session_destroy();
5 header ("Location: index.html");
6 ?> $
```

O *script* `logout.php` não faz nada mais que destruir a sessão construída pelo *script* `login.php`. Como os dados deixam de existir, quando o usuário tenta acessar páginas que possuem conteúdo restrito, o arquivo valida_sessao.php detecta e direciona o usuário para a página inicial do sistema.

4.5 Página principal

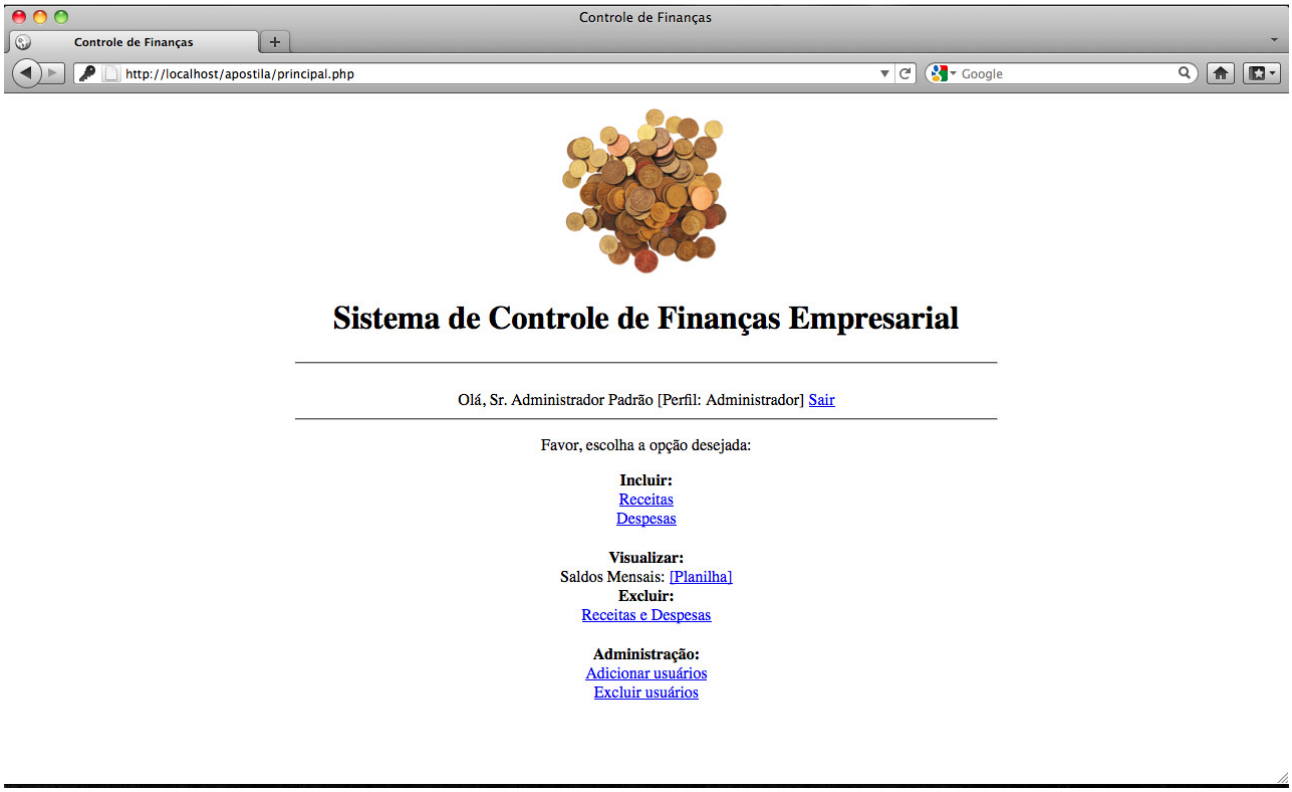
Vamos agora apresentar a tela principal de nosso sistema. Lembrando que esta tela é restrita apenas para os usuários cadastrados e que, se todos os procedimentos detalhados anteriormente tiverem sido executados, usuários não autenticados não poderão acessá-la. A figura 4.5 exibe como essa *interface* está construída, apresentando todas suas funções. Vamos, portanto, cuidadosamente estudá-la.

Figura 4.5: Arquivo `principal.php`. Tela principal do sistema para usuários comuns.

Observando a parte superior da tela, podemos observar duas informações sobre o usuário que está acessando. A primeira é o nome completo do usuário, cadastrado durante sua inscrição no site. A

outra corresponde ao seu perfil. Vamos lembrar que no início de nossa aula, colocamos que o nosso sistema terá dois perfis diferente de usuários. Vocês lembram? Isso mesmo.. administrador e padrão. No caso, é mostrado que o usuário "Daniel Silva Ferreira" possui seu perfil definido como **padrão**. No entanto, se observamos a tela apresentada pela figura 4.6, vamos verificar que novas opções surgem quando o usuário em questão possui perfil **administrador**.

Figura 4.6: Arquivo `principal.php`. Tela principal do sistema para usuários administradores.



As atividades de gerenciamento de usuários é restrita apenas a administradores do sistema e, portanto, usuários **padrão** não podem executá-la. Caso, algum usuário esperto que não esteja definido como administrador tende acessar diretamente as páginas para gerenciamento de usuários, o *script valida_sessao.php* irá bloqueá-lo. O código fonte apresentado em 4.10 correspondente à página principal de nosso sistema.

Figura 4.10: `principal.php`

```
1  <?php
2  include "valida_sessao.inc";
3  include "conecta_mysql.inc";
4  $nome_usuario = $_SESSION["nome_usuario"];
5  $perfil_usuario = $_SESSION["perfil_usuario"];
6  $resultado = mysql_query("SELECT * FROM usuarios WHERE login='$nome_usuario'");
7
8  $sexo = mysql_result($resultado,0,"sexo");
9  $nome = mysql_result($resultado,0,"nome");
10
11 mysql_close($con);
12
13 switch ($sexo) {
14         case 1:
15                 $saud = "áOl, Sra. " . $nome;
16                 break;
17         case 2:
18                 $saud = "áOl, Sr. " . $nome;
19                 break;
20 }
21
22 switch ($perfil_usuario) {
```

```
23          case 1:
24                  $perfil = "ãPadro";
25                  break;
26          case 2:
27                  $perfil = "Administrador";
28                  break;
29  }
30  ?>
31  <html>
32  <head><title>Controle de çFinanas</title></head>
33  <body>
34  <form method="POST" action="login.php">
35    <center>
36    <img src="coins.jpg" width="15%"/>
37    <h1>Sistema de Controle de çFinanas Empresarial</h1>
38    <hr width="700px"  /><br />
39    <?php echo $saud . " " . "[Perfil: ".$perfil."]";?> <a href="logout.php">Sair</a>
40    <hr width="700px"  />
41    <p>Favor, escolha a çãopo desejada: </p>
42    <b>Incluir: </b> <br />
43    <a href="receitas_despesas.php?t=1">Receitas</a> <br />
44    <a href="receitas_despesas.php?t=2">Despesas</a> <br /> <br />
45    <b>Visualizar: </b> <br />
46    Saldos Mensais: <a href="saldosMensaisPlan.php">[Planilha]</a>
47    <br />
48    <b>Excluir: </b> <br />
49    <a href="excluirReceitasDespesas.php">Receitas e Despesas</a> <br /> <br />
50    <?php
51          if ($perfil_usuario==2){ ?>
52                  <b>çãAdministrao: </b> <br />
53                          <a href="addUsuarios.php">Adicionar áusurios</a> <br />
54                          <a href="delUsuarios.php">Excluir áusurios</a> <br /> <br />
55    <?php } ?>
56    </center>
57  </form>
58  </body>
59  </html>
```

Analisando as linhas 50 a 55 podemos observar o código PHP necessário para habilitar as opções de gerencimento de usuários. Na estrutura relacional construída, se o perfil do usuário for igual a 2 (número referente ao perfil administrador), os *links* para cadastramento e remoção de usuários são exibidos. Notamos ainda que não é necessário a construção de dois documentos para sistemas com esta funcionalidade. Nossa página principal é a mesma para os dois perfis, apenas seleciona opções de acordo com o usuário.

4.6 Cadastro de Receitas e Despesas

A figura 4.7 apresenta os formulários necessários para cadastrar receitas e despesas em nosso sistema. Ao clicar em **receitas**, na página principal, a linha 43 do código exibido na figura 4.10 direciona o usuário para o arquivo `receitas_despesas.php` com uma variável `t` configurada com valor 1. Essa variável será responsável em informar ao *script* `receitas_despesas.php` que a operação de cadastro será referente a uma receita. Caso o usuário deseje cadastrar uma despesa, a linha 44 do código da página principal irá direcionar o usuário para o mesmo arquivo `receitas_despesas.php`, com a diferença da variável `t` enviada ser de valor 2. Então, vocês podem se perguntar: por que não são construídas duas páginas diferentes de cadastro, uma para receita e outra para despesa? A resposta é simples. Imagine como seria o código fonte dessas duas páginas. Se você tiver imaginado corretamente, verá que seus fontes serão bastante iguais. Ou seja, eles irão possuir códigos fontes semelhantes e uma mesma organização. Dessa forma, é preferível economizar códigos, fazendo apenas um documento e alterando entre as particularidades de cada ação por meio da linguagem de programação. As vantagens disso é que, com apenas um documento, fica mais fácil administrar, os

códigos ficam sem redundância e consequentemente tornará o processo de manutenção mais simples.

Figura 4.7: Arquivo `receitas_despesas.php`.

(a) Formulário para cadastro de receitas.

(b) Formulário para cadastro de despesas.

O campo do navegador referente a URL apresenta, após o carácter `?`, a atribuição `t=1` ou `t=2` para os casos de cadastro de receita e despesa, respectivamente. Como essa informação está sendo transmitida diretamente na URL, qual dos métodos está sendo utilizado: `GET` ou `POST`? Se você não souber, favor estude novamente a aula 2. Vamos relembrar. Quando a informação está disponível para o servidor por meio da URL, utiliza-se o método `GET`. Logo, para que as páginas exibidas na figura 4.7 sejam confeccionadas, essa informação tem que ser obtida. Vamos, portanto, analisar a figura 4.11 que apresenta o código responsável pela montagem dessas telas.

Figura 4.11: `receitas_despesas.php`

```
1  <?php
2  include "valida_sessao.inc";
3  //include "conecta_mysql.inc";
4  $nome_usuario = $_SESSION["nome_usuario"];
5
6  //$resultado = mysql_query("SELECT id FROM usuarios WHERE login='$nome_usuario'");
7  //$id = mysql_result($resultado,0,"id");
8  //mysql_close($con);
9
10 $t = $_GET['t'];
11
12 switch ($t) {
13         case 1:
14                 $tipo = "receita";
15                 break;
16         case 2:
17                 $tipo = "despesa";
18                 break; \\ $
19 }
20
21 ?>
22 <html>
23 <head><title>Controle de çFinanas</title></head>
24 <body>
25  <center>
26   <img src="coins.jpg" width="15%"/>
27   <h1>Sistema de Controle de çFinanas Empresarial</h1>
28   <hr width="700px"  /><br />á
29   Formulrio para cadastro de <?php echo $tipo ?> (* óObrigatrio)<br /><br />
30   <form method="post" action="gravar.php" name='fCadRecDes'>
31         <table>
32           <tr>
33             <td width="130px">Nome*: </td>
```

```
34              <td width="200px"><input size="80" type="text" name="nome"></td>
35            </tr>
36            <tr>
37              <td width="130px">Classe*: </td>
38                  <td width="200px">
39                    <input type="radio" name="classe" value="1" checked>áVarivel
40                    <input type="radio" name="classe" value="2" onclick="">Fixa
41                  </td>
42            </tr>
43            <tr>
44              <td width="130px">êMs de êreferncia*: </td>
45                  <td width="200px">
46                    <select name="mesRef">
47                      <option value="1">Janeiro</option>
48                      <option value="2">Fevereiro</option>
49                      <option value="3">çMaro</option>
50                      <option value="4">Abril</option>
51                      <option value="5">Maio</option>
52                      <option value="6">Junho</option>
53                      <option value="7">Julho</option>
54                      <option value="8">Agosto</option>
55                      <option value="9">Setembro</option>
56                      <option value="10">Outubro</option>
57                      <option value="11">Novembro</option>
58                      <option value="12">Dezembro</option>
59                    </select>
60                  </td>
61            </tr>
62            <tr>
63              <td width="130px">Valor* (R$): </td>
64              <td width="200px">
65                <input size="10" type="text" name="valor"> (formato (xx.xx))
66              </td>
67            </tr>
68            <tr>
69              <td width="130px">çãDescrio: </td>
70              <td width="200px">
71                <textarea rows="7" cols="69" name="descricao"></textarea>
72              </td>
73            </tr>
74            <tr>
75              <td width="130px">
76                    <input type="button" value="Voltar" name="voltar"
77                           onclick="javascript:history.back()">
78              </td>
79              <td width="200px">
80                    <input type="reset" value="Limpar">
81                    <input type="submit" value="Salvar">
82                    <input type="hidden" name="t"
83                       value="<?php echo $t?>">
84             </td>
85           </tr>
86    </table>
87   </form>
88 </center>
89 </body>
90 </html>
```

A linha 10 do código apresentado na figura 4.11 atribui a uma variável PHP `$t` o valor transmitido por meio da URL. Nas linhas 12 a 19 existe o tratamento desta variável. Neste procedimento, a variável `$tipo` é configurada como receita ou despesa dependendo do valor. Esta informação é a única que muda entre os dois casos, o que justifica a existência de uma página para esta atividade. Imagine o volume de códigos replicados que iríamos ter tivéssemos dois arquivos para realizar esse cadastro.

Outro detalhe importante está colocado nas linhas 82 e 83. Observe que existe um `input` do tipo `hidden` (type="hidden"). Este tipo significa que este botão não será visível ao usuário. Daí, vem aquela perguntinha... se o botão é invisível, entende-se que ele não terá utilidade para o usuário, logo, para que ele foi definido? A questão é bem simples. Apesar de não aparecer, ele é muito útil para o sistema. É por meio dele que vamos enviar a informação de tipo (receita ou despesa) para ser gravada no banco de dados. Como aprendemos na aula 3, os dados do formulário HTML são transmitidos para um *script* `PHP` que irá registrar na base. Neste caso, se não utilizássemos este recurso, ficaria muito complicado enviar esta informação, receita ou despesa, para o *script* de gravação.

Vamos agora conhecer para onde as informações digitadas nas telas da figura 4.7 são enviadas. Trata-se do *script* `gravar.php` apresentado na figura 4.12. O estudo cuidadoso deste *script* é muito importante para entendimento de nosso sistema. Observem as questões de validação, funções de data e hora do `PHP` e código `SQL` necessário para efetuar a escrita.

Figura 4.12: `gravar.php`

```
1  <?php
2  include "valida_sessao.inc";
3  include "conecta_mysql.inc";
4  // Obtem o usuario que efetuou o login
5  $nome_usuario = $_SESSION["nome_usuario"];
6  // Obtem informacoes digitadas
7  $t = $_POST['t'];
8  $nome = $_POST['nome'];
9  $classe = $_POST['classe'];
10 $mesRef = $_POST['mesRef'];
11 $valor = $_POST['valor'];
12 $descricao = $_POST['descricao'];
13
14 // Validacao dos campos nome e valor.
15 if(empty($nome) or empty($valor)){
16         $erro=1;
17         $msg ="Por favor, preencha todos os campos obrigatorios.";
18 }elseif ((substr_count($valor, '.')!=1) or (!is_numeric($valor))){
19         $erro=1;
20         $msg ="Digitar o campo valor apenas com numeros e no formato (xx.xx).";
21 }else{
22         // Tratamento da Descricao
23         if (empty($descricao)) $descricao = NULL;
24         // Id do usuario que efetuou o login
25         $resultado =
26                 mysql_query("SELECT id FROM usuarios WHERE login='$nome_usuario'");
27         $idUsuario = mysql_result($resultado,0,"id");
28         // Data e Hora
29         $datahora= date("Y-m-d H:i:s");
30         //Formatar o valor para duas casas decimais.
31         $valor = number_format($valor, 2, '.', '');
32         //Comandos SQL para insercao na base de dados.
33         $comandoSQL =
34         "insert into `apostila_ntwi`.`receitas_despesas`
35         (`nome`,`tipo`,`classe`,`mes_referencia`,`datahora`,`valor`,`usuario`,`descricao
36         values";
37         $comandoSQL .= " ('$nome','$t','$classe','$mesRef','$datahora','$valor','$idUsua
38         $resultado = mysql_query($comandoSQL) or
39                 die('Erro fatal durante operacao com base de dados');
40         $msg ="Inclusao realizada com sucesso.";
41 }
42 mysql_close($con);
43 ?>
44 <html>
45 <head><title>Controle de Finan&ccedil;as</title></head>
46 <body>
47   <center>
48         <img src="receita_despesa.jpg" width="30%" height="30%"/>
```

```
49          <h1>$$$ Sistema de Controle de Financas $$$</h1>
50          <hr width="700px"/><br />
51          <?php
52                  echo "<p>".$msg."</p>";
53          ?>
54          <p><a href='principal.php'>Voltar</a></p>
55    </center>
56 </body>
57 </html>
```

Vamos iniciar nossa discussão com as instruções apresentadas pelas linhas 5 a 12. Observe que o *login* do usuário é obtido por meio da sessão. Lembre-se que essa informação ficou gravada na sessão durante o procedimento de autenticação. Em seguida, os dados referentes aos demais campos do formulário são obtidos pelo *array* superglobal `POST`.

A linha 15 inicia um procedimento conhecido como validação. Nesta etapa, temos que verificar se os campos do formulário foram preenchidos de acordo com o tipo de dados esperado. Isto é, precisamos verificar se no campo `valor`, por exemplo, foram digitados apenas números e não letras. Esta etapa é bastante importante porque protege a base de dados de eventuais problemas e códigos maliciosos. Não podemos construir aplicações para web sem que exista esse cuidado com os dados que o usuário está digitando. Em nosso formulário, apenas dois campos, `nome` e `valor`, possuem uma entrada não controlada. Dessa forma, precisamos realizar validação, visto que não dar para ter certeza que a informação será sempre digitada no sistema de forma correta. Os demais campos possuem entrada controlada, ou seja, o usuário escolhe o seu valor dentre algumas opções. Como não existe a possibilidade de digitação, a entrada dessa informação sempre estará com sua definição correta de tipo.

A função `empty()`, empregada na linha 15, verifica se um determinado campo não está vazio. Como em nosso formulário de cadastro os campos `nome` e `valor` são de preenchimento obrigatório, então eles não podem estar vazios. Observe que, para o campo `descrição`, esta verificação não é efetuada, pois trata-se de uma informação opcional. Os demais campos não precisam passar por este procedimento, devido ao fato de seus valores partirem de uma escolha do usuário e, portanto, sempre vai haver um valor de tipo conhecido.

Se tudo ocorrer bem durante a execução da linha 15, é a hora de verificar se o `valor` foi digitado corretamente. Um erro comum é o usuário digitar números com vírgula ou ponto. Por exemplo, 200, 12 ou 200.12. Você pode construir o sistema para receber valores das duas formas. Se isto for feito, você torna o seu site mais robusto e mais cômodo ao usuário. Também devemos tomar cuidados com os números que eventualmente são colocados sem casas decimais como, por exemplo, 200. No nosso caso, vamos construir essa verificação da forma mais simples, ou seja, todos os valores tem que ser obrigatoriamente digitados com duas casas decimais e com ponto (.). Para validar isso, a instrução da linha 18 é executada. A função `substr_count()` verificar se o número digitado possui uma ocorrência do carácter ponto (.). A instrução é complementada pela função `is_numeric()` responsável em analisar se o conteúdo digitado de fato correspondente a um número. Ambas funções estão ligadas por meio do operador lógico `and`, significando que a expressão será verdadeira somente se as ambas funções forem verdadeiras.

Se algum erro ocorrer nas verificações descritas anteriormente, uma mensagem é emitida para o usuário. Em caso de sucesso, nosso *script* pode formular o código `SQL` e finalmente efetuar a escrita dos dados na base. Este procedimento ocorre entre as linhas 22 e 35. Estudando a linha 22, inicialmente, veremos que o valor `NULL` é atribuído a descrição, caso o usuário não tenha preenchido no formulário. A linha 28 obtém a data e hora do sistema no momento da execução. Ter registrado na base o momento exato que os registros são gravados é importante para uma boa organização dos dados e principalmente em questões de segurança, pois pode oferecer pistas valiosas em caso de gravações não autorizadas.

A linha 30 tem a missão de formatar o número recebido para duas casas decimais. Mas isso já não foi feito na linha 18 com a função `substr_count()`? A resposta é não. Por meio da função `substr_count()` é possível validar se o valor de entrada tem um ponto, no entanto, se o usuário digitar 123.456 (três casas decimais), o uso apenas dessa função não é suficiente. A função `number_format()`,

como exposto na linha 30, força o número a ter duas casas decimais para poder ser escrito de forma padronizada no banco de dados. Lembre-se que padronizar, além de tornar seu código mais elegante, facilita a tarefa posterior de manutenção.

Após todas essas verificações quanto aos dados fornecidos pelo usuário, podemos finalmente montar a *query* `SQL` para gravação das informações. Se essas questões tiverem sido corretamente implementadas, vamos eliminar muitos dos problemas que poderíamos ter com dados fora do padrão na base de dados. Mas, se mesmo com tudo isso, algo passar, a função `die()` que compõe a instrução da linha 34 leva a um erro fatal de escrita e comunica a mensagem passada como parâmetro para o usuário. Se esta função não for empregada, o sistema pode simplesmente travar ou aparecer uma mensagem típica de erro do banco `MySQL` nada elegante para o usuário.

4.7 Visualização dos dados

Ao acessar a opção `planilha` da página principal, figura 4.5, o usuário será encaminhado para a tela de visualização das receitas e despesas cadastradas. No entanto, inicialmente é necessário escolher o mês de referência desses dados. Lembre-se que esta informação faz sentido somente para as receitas e despesas `variáveis`, pois as `fixas` são contabilizadas independente do mês. Então, por meio da tela exibida na figura 4.8, podemos escolher o mês do qual queremos acessar os dados. Vamos analisar.

Figura 4.8: Arquivo `saldosMensaisPlan.php`.

Uma vez selecionado o mês, o usuário será encaminhado para a visualização dos dados correspondentes ao mês escolhido. Esta tela está organizada em forma de planilha e são mostradas tanto as despesas fixas e variáveis quanto as receitas. No final da página, podemos observar um resumo do mês, onde observamos o saldo (diferença entre o total de receitas e total de despesas). Por questões de apresentação, a tela de visualização dos dados (exemplificada pelo mês de novembro) foi dividida nas figuras 4.9a e 4.9b.

Observe que a organização dos dados em forma de planilha fornece condições claras para interpretação dos dados. O código necessário para geração desta página está descrito na figura 4.13. Observe que a geração das tabelas ocorre de forma dinâmica com base nos resultados da execução das *query* `SQL`. Para facilitar o entendimento, foram realizadas quatro consultas independentes ao banco de dados, que podem ser estudadas nas linhas 9 a 18. Que tal o desafio de tentar fazer a mesma planilha com apenas uma única *query*? Divirtam-se.

Figura 4.9: `saldosMensaisPlan.php`. Parte superior.

(a) Parte superior.

(b) Parte inferior.

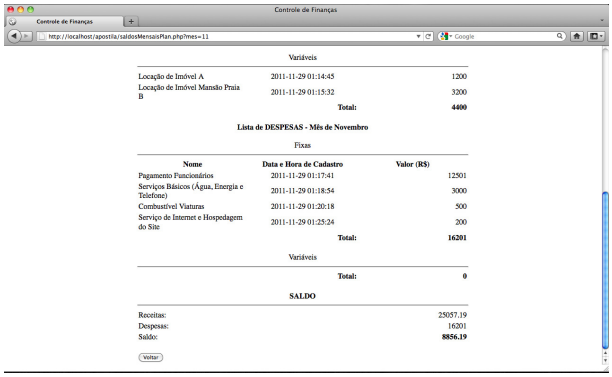


Figura 4.13: `saldosMensaisPlan.php`

```
1  <?php
2  include "valida_sessao.inc";
3  include "conecta_mysql.inc";
4  $nome_usuario = $_SESSION["nome_usuario"];
5  $id_usuario = $_SESSION["id_usuario"];
6  $mes = $_GET['mes'];
7  $meses = array ("Janeiro","Fevereiro","çMaro","Abril","Maio","Junho","Julho",
8          "Agosto","Setembro","Outubro","Novembro","Dezembro");
9  $resRecVar = mysql_query("SELECT * FROM receitas_despesas
10         WHERE classe=1 and mes_referencia=$mes and tipo=1 and
11         usuario=$id_usuario");
12 $resDesVar = mysql_query("SELECT * FROM receitas_despesas
13         WHERE classe=1 and mes_referencia=$mes and tipo=2 and
14         usuario=$id_usuario");
15 $resRecFix = mysql_query("SELECT * FROM receitas_despesas
16         WHERE classe=2 and tipo=1 and usuario=$id_usuario");
17 $resDesFix = mysql_query("SELECT * FROM receitas_despesas
18         WHERE classe=2 and tipo=2 and usuario=$id_usuario");
19 // Valores Totais Receitas e Despesas
20 $recVarTotal = 0; $recFixTotal = 0; $desVarTotal = 0; $desFixTotal = 0;
21 mysql_close($con);
22 ?>
23 <html>
24 <head><title>Controle de çFinanas</title></head>
25 <body>
26 <form method="GET" name="fmes" action="saldosMensaisPlan.php">
27   <center>
28   <img src="coins.jpg" width="15%"/>
29   <h1>Sistema de Controle de çFinanas Empresarial</h1>
30   <hr width="700px"  /><br />
31   <p>Favor, escolha o mes que deseja visualizar:
32         <select name="mes">
33                 <option value="1" onclick="javascript:document.fmes.submit();">
34                         Janeiro</option>
35                 <option value="2" onclick="javascript:document.fmes.submit();">
36                         Fevereiro</option>
37                 <option value="3" onclick="javascript:document.fmes.submit();">
38                         Marco</option>
39                 <option value="4" onclick="javascript:document.fmes.submit();">
40                         Abril</option>
41                 <option value="5" onclick="javascript:document.fmes.submit();">
42                         Maio</option>
43                 <option value="6" onclick="javascript:document.fmes.submit();">
44                         Junho</option>
45                 <option value="7" onclick="javascript:document.fmes.submit();">
46                         Julho</option>
```

```
47                  <option value="8" onclick="javascript:document.fmes.submit();">
48                          Agosto</option>
49                  <option value="9" onclick="javascript:document.fmes.submit();">
50                          Setembro</option>
51                  <option value="10" onclick="javascript:document.fmes.submit();">
52                          Outubro</option>
53                  <option value="11" onclick="javascript:document.fmes.submit();">
54                          Novembro</option>
55                  <option value="12" onclick="javascript:document.fmes.submit();">
56                          Dezembro</option>
57          </select>
58    </p>
59 <?php if (isset($mes))
60 {
61 ?>
62 <b>Lista de RECEITAS - Mes de <?php echo $meses[$mes-1]?></b><br><br>
63 Fixas
64          <hr width="700px"  />
65          <table width=700px border=0px>
66                  <th> Nome </th><th> Data e Hora de Cadastro </th><th> Valor (R$)</th>
67                  <?php
68                  while($linha = mysql_fetch_array($resRecFix, MYSQL_ASSOC))
69                  {
70                          echo "<tr>";
71                          echo "<td align='left' width=33%>" . $linha["nome"] . "</td>";
72                          echo "<td align='center' width=33%>" . $linha["datahora"] . "</t
73                          echo "<td align='right' width=33%>" . $linha["valor"] . "</td>";
74                          echo "</tr>";
75                          // Incrementar o valor total
76                          $recFixTotal = $recFixTotal +  $linha["valor"];
77                  }
78                  ?>
79                  <tr>
80                          <td width=33%></td><td align='right' width=33%><b>Total: </b></t
81                  </tr>
82          </table><br>
83          Variaveis
84          <hr width="700px"  />
85          <table width=700px border=0px>
86                  <?php
87                  while($linha = mysql_fetch_array($resRecVar, MYSQL_ASSOC))
88                  {
89                          echo "<tr>";
90                          echo "<td align='left' width=33%>" . $linha["nome"] . "</td>";
91                          echo "<td align='center' width=33%>" . $linha["datahora"] . "</t
92                          echo "<td align='right' width=33%>" . $linha["valor"] . "</td>";
93                          echo "</tr>";
94                          // Incrementar o valor total
95                          $recVarTotal = $recVarTotal +  $linha["valor"];
96                  } ?>
97                  <tr>
98                          <td width=33%></td><td align='right' width=33%><b>Total: </b></t
99                  </tr>
100         </table><br />
101         <b>Lista de DESPESAS - Mes de <?php echo $meses[$mes-1]?></b><br /><br />
102         Fixas
103         <hr width="700px"  />
104         <table width=700px border=0px>
105                 <th> Nome </th><th> Data e Hora de Cadastro </th><th> Valor (R$)</th>
106                 <?php
107                 while($linha = mysql_fetch_array($resDesFix, MYSQL_ASSOC))
108                 {
109                         echo "<tr>";
110                         echo "<td align='left' width=33%>" . $linha["nome"] . "</td>";
111                         echo "<td align='center' width=33%>" . $linha["datahora"] . "</t
```

```
112                         echo "<td align='right' width=33%>" . $linha["valor"] . "</td>";
113                         echo "</tr>";
114                         // Incrementar o valor total
115                         $desFixTotal = $desFixTotal +  $linha["valor"];
116                 } ?>
117                 <tr>
118                         <td width=33%></td><td align='right' width=33%><b>Total: </b></t
119                 </tr>
120         </table><br />
121         Variaveis
122         <hr width="700px"  />
123         <table width=700px border=0px>
124                 <?php
125                 while($linha = mysql_fetch_array($resDesVar, MYSQL_ASSOC))
126                 {
127                         echo "<tr>";
128                         echo "<td align='center' width=33%>" . $linha["nome"] . "</td>";
129                         echo "<td align='center' width=33%>" . $linha["datahora"] . "</t
130                         echo "<td align='center' width=33%>" . $linha["valor"] . "</td>"
131                         echo "</tr>";
132                         // Incrementar o valor total
133                         $desVarTotal = $desVarTotal +  $linha["valor"];
134                 } ?>
135                 <tr>
136                         <td width=33%></td><td align='right' width=33%><b>Total: </b></t
137                 </tr>
138         </table><br />
139         <b>SALDO</b>
140         <hr width="700px"  />
141         <table width=700px border=0px>
142                 <tr>
143                         <td width="50%">Receitas: </td>
144                         <td align="right" width="50%"><?php echo ($recFixTotal+$recVarTo
145                 </tr>
146                 <tr>
147                         <td width="50%">Despesas: </td>
148                         <td align="right" width="50%"><?php echo ($desFixTotal+$desVarTo
149                 </tr>
150                 <tr>
151                         <td width="50%">Saldo: </td>
152                         <td align="right" width="50%">
153                                 <b><?php echo ($recFixTotal+$recVarTotal)-($desFixTotal+
154                 </tr>
155                 <tr>
156                         <td>
157                                 <input type="button" onClick="location.href='principal.p
158                         </td>
159                         <td>
160                         </td>
161                 </tr>
162         </table>
163
164 <?php
165 }
166 ?>
167 </center>
168 </form>
169 </body>
170 </html>
```

Vamos estudar agora algumas particularidades do código fonte exibido na figura 4.13. Inicialmente vamos atentar para o fato de que esse mesmo código é responsável em gerar as telas das figuras 4.8 e 4.9. O controle é efetuado por meio da variável PHP `$mes`. Quando algum valor é atribuído a ela, por meio do `select` de escolha do mês, os dados são capturados no banco de acordo com o número

correspondente. Essa informação é transmitida ao *script* por meio do evento `onclick` configurado nas opções do `select`. Quando nenhum valor está contido em `$mes`, significa que nenhum mês foi escolhido, portanto, apenas o `select` inicial é apresentado.

4.8 Remover receitas e despesas

Ao selecionar esta opção na página principal, o usuário deve ser direcionado para uma página onde ele possa selecionar e remover receitas e despesas cadastradas. Com base no que foi exposto nas seções anteriores, tente construir o documento `excluirReceitasDespesas.php` para realizar esta atividade. Atente para a forma de organização dos dados. Lembre-se que o usuário deve conseguir selecionar de forma simples e precisa o registro que deseja remover. Utilize os fontes anteriores como exemplo e mãos à obra.

4.9 Administração dos usuários

Esta ação é restrita apenas para usuários cujo perfil está definido como administrador. Nesta parte do sistema, o usuário pode tanto adicionar novos usuários, como deletar existentes. Fundamentado nas questões discutidas anteriormente e na modelagem da base de dados exposta no início da aula, desenvolva os documentos `addUsuarios.php` e `delUsuarios.php` agregando ao sistema. Tente seguir o mesmo *layout* adotado. Utilize os fontes já estudados como exemplos e não esqueçam que, em programação, praticar é essencial para o aprendizado.