Apostila Básica De HTML, PHP, MYSQL

Autor: Daniel Garcia Teixeira
Curso: PHP Básico
Duração Aproximada: 9h
Contatos: danielgt@cefetrn.br

HTML

1 - HIERARQUIA DE ELEMENTOS

A estrutura básicas de uma página HTML é mostrada na **listagem 1.1**. Observe que a construção de páginas exigir o uso de marcadores chamados de “TAGS”. Veja agora o uso deles na **listagem 1.1**

Listagem 1.1

```
<html>
     <head>
          <title>COLOQUE AQUI O TÍTULO DA PÁGINA</title>
     </head>
     <body>
          DAQUI EM DIANTE Você DESENVOLVE SUA PÁGINA
     </body>
</html>
```

Fim da Listagem 1.1

Com certeza você observou que sempre usei os TAGS, fazendo demarcação, ou seja, eles sempre **estarão ANTES DE ALGUMA COISA** E **APÓS ALGUMA COISA.**
Exemplo: **<title>EDITORA ERICA</title>**
Para facilitar deve se identar o código quando se abre uma TAG como dá pra ver na listagem acima. Ao abrir o <head> os itens dentro dele ficaram com um recuo a frente para facilitar o entendimento do código.
A seguir vemos na listagem 1.2 outras TAGS que são explicadas no quadro seguinte.

Listagem 1.2

```
<html>
<head><title>Melhorando Minha Home Page</title></head>
<! Início do Corpo da Página>
     <body>
          <h1>Este é o título Principal</h1>
          <h2>Este é o título Secundário</h2>
          <h3> Estou adorando criar páginas</h3>
          <hr>
          <p>Este é o 1º Primeiro Paragráfo <br> Esta é 2ª
          Linha  do 1º Paragráfo
          <p>Com este recurso inicio um paragráfo<br> E Com
          este recurso quebro uma linha
          <hr>
<! Fim do Corpo da Página>
     </body>
</html>
```

Fim da Listagem 1.2

CONCEITO DOS TAGS USADOS

TAG	O QUE FAZ
<html>	Marca o início e o fim do documento HTML. Com ele você inicia e finaliza a construção da página Web.
<head>	Marca o início e do fim do cabeçalho, ou seja, a área onde serão descritos cabeçalhos e o título da página
<title>	Marca o início e o fim do título do cabeçalho. O título da página aparecerá na barra superior do browser.
<body>	Marca o início e o fim do do corpo da página
<!>	Insere comentários nas páginas
<hn>	Marca um título. Sendo que n representa um valor que pode ser de 1 a 6, o tamanho muda de forma decrescente, ou seja, o número 1 é o maior tamanho do título.
<hr>	Insere uma linha horizontal
<p>	Marca um parágrafo e acrescenta uma linha em branco.
 	Insere uma quebra de linha

2 - TAGS DE ALINHAMENTO

Assim, como num documento comum, há necessidade de melhorar a aparência do documento, e a primeira providência a tomar é cuidar do alinhamento do texto. O Alinhamento padrão que vem configurado nos navegadores é à esquerda. Para entender isto, observe a **listagem 1.3**.

Listagem 1.3

```
<html>
<head>
     <title>TAGS para Alinhamentos</title>
</head>
     <body>
          <h4 align=center>Título Centralizado</h4>
          <h4 align=right>Título À Direita</h4>
          <h4 align=left> Título À Esquerda<h4>
          <hr>
          <p align=center> Parágrafo ao Centro
          <p align=right>Parágrafo a direita
          <p align=left>Paragráfo a esquerda
          <p align=justify>Este é um texto justificado. Na
linguagem HTML temos vários tipos de alinhamentos que você
poderá aplicar em sua página. Nesta parte do livro,veremos
como alinhar linhas, parágrafos e cabeçalhos.
          <br><br>
<hr width=50% align=center>
          <blockquote>Texto com mais margem</blockquote>
          <blockquote><blockquote>Tem com mais margem
ainda</blockquote></blockquote>
     </body>
</html>
```

Fim da Listagem 1.3

CONCEITO DOS TAGS USADOS

TAG	O QUE FAZ
	Alinha o trecho (texto, imagem ou tabela ao centro>
align=center, right, left ou justify	Atribuídos dentro do tag <p> que marca o início de um parágrafo modificam o alinhamento do título. Center= alinha ao centro Right = alinha a direita Left = alinha a esquerda Justify = faz a justificação do parágrafo.
<blockquote>	Adiciona uma margem de cerca de um centímetro
<hr>	Atributos Size = define a altura da linha. Exemplo: <hr size=50> Width = define a largura da linha horizontal. Exemplo: <hr width=200>ou <hr width=50%> Noshade = desenha a linha sem a sombra para dar o efeito de três dimensões. Exemplo: <hr noshade>

3 - FORMATAÇÃO DE ESTILOS

Muito bem caro estudante, perceba que a cada exemplo sua página vai melhorando ainda mais sua aparência. Neste exemplo trabalharemos com a formatação das letras bem como cor, tamanho de fonte, estilo, e etc..
Para entender isto, observe a **listagem 1.4.**

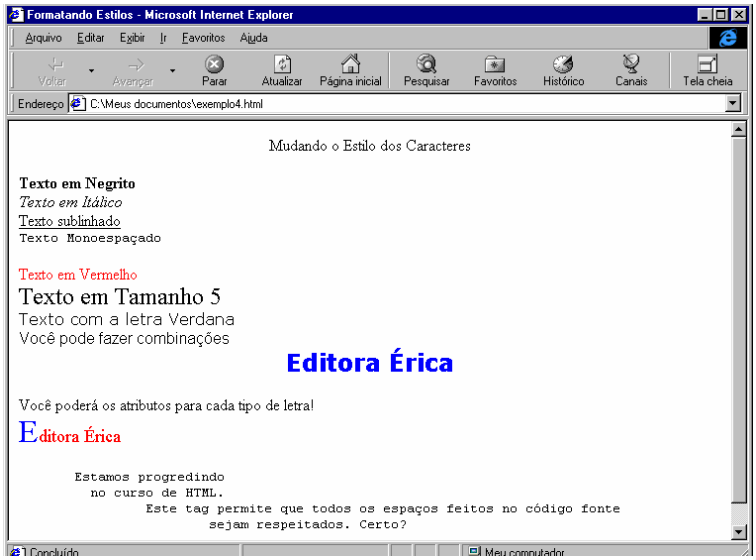
Listagem 1.4

```
<html>
      <head>
         <title>Formatando Estilos</title>
      </head>
      <body>
      <center>Mudando o Estilo dos Caracteres</center>
      <p>
      <b>Texto em Negrito</b><br>
      <i>Texto em Itálico</i><br>
      <u>Texto sublinhado</u><br>
      <tt>Texto Monoespaçado</tt><br>
      <br><font color=red>Texto em Vermelho</font>
      <br><font size=5>Texto em Tamanho 5</font>
      <br><font    face=verdana>Texto    com    a    letra
Verdana</font>
      <br><font    face=arial    black>Você    pode    fazer
combinações</font>
      <br><center>
      <font   color=blue   face=verdana   size=5><b>Editora
Érica</b></font></center>
      <br>Você poderá os atributos para cada tipo de letra!
      <br>
```

```
      <font   color=blue   size=6>E</font><font   color=red
size=4>ditora Érica</font>
      <br>
      <pre>
       Estamos progredindo
         no curso de HTML.
                Este  tag  permite  que  todos  os  espaços
feitos no código fonte
                        sejam respeitados. Certo?
      </pre>
      </body>
 </html>
```

Fim da Listagem 1.4

Veja a figura do resultado no navegador.



CONCEITO DOS TAGS USADOS

TAG	O QUE FAZ
<b>	Coloca o texto em negrito
<i>	Coloca o texto em itálico
<u>	Coloca o texto sublinhado
<tt>	Coloca o texto em fonte monoespaçada. (fonte Courier,como máquina de escrever)
	Modifica a formatação do texto. Atributos: **Size** = define o tamanho da letra. Ex: Texto **Face** = define o estilo da letra. Ex: Texto **Color** = define a cor da letra. Ex:Texto
<pre>	Marca um trecho formatado com fonte monoespaçada. A formatação com espaços e entrada de parágrafos é respeitada.
<basefont>	Modifica a formatação padrão do texto. Ex: <basefont size=5>

TABELA DE CORES

Você percebeu que as cores a fonte obedecem o idioma inglês, no entanto, as cores da fonte podem ser adicionados através do nome ou de seus respectivos códigos. Então para você ficar mais feliz, relacionei aqui algumas cores para colorir e diversificar sua home page. Veja a figura a seguir.

Também é possível setar cor para toda pagina na TAG <body> como descrito na tabela a baixo.

CONCEITO DOS TAGS USADOS

TAG	O QUE FAZ
<body>	Marca o início e o fim do corpo da página Atributos **Bgcolor** = define a cor do fundo da página **Text** = define a cor do texto padrão da página **Background** = permite inserir uma imagem como fundo da página

4 - IMAGENS

Neste ponto veremos os TAGS que permitem a inserção de imagens em sua home page.
Antes de iniciar o estudo sobre imagens, quero lembrar-lhe que uma imagem só poderá ser exibida no browser, se ela estiver na mesma pasta, ou então, você deverá apontar o caminho onde está.
Outro fato é que, você poderá escolher as imagens que desejar, basta substituir o nome da imagem que está no exemplo pela a que você escolheu.
Para entender isto, observe a **listagem 1.5**.

Listagem 1.5

```
<html>
     <head><title>Imagens</title></head>
     <body>
     <h1>Agora estamos trabalhando com imagens</h1>
     <h3> Esta é sua primeira imagem. Sorria?</h3>
     <img src=fig.jpg align=left>Veja se consegue enxergar,
     as seguintes imagens?<br>
          O rosto de um velho<br>
          Uma moça<br>
          Um cachorro<br>
          Um outro velho<br>
          Um pássaro<br>
          E algo mais<br>
     </body>
</html>
```

Fim da Listagem 1.5

Veja a figura do resultado no navegador.

CONCEITO DOS TAGS USADOS

TAG	O QUE FAZ
<img>	Insere uma imagem **Atributos** **src** indica o nome da imagem a ser carregado. **align=middle** centraliza o base do texto com o centro da imagem **align=left** faz a imagem flutuar a esquerda enquanto o texto circunda imagem à direita. **align=top** alinha o texto no topo **align=right** faz a imagem flutuar a direita enquanto o texto circunda imagem à esquerda. **alt="n"** indica o texto para ser exibido quando o navegador não exibe a imagem. Sendo que n é o título que identifique a imagem. Exemplo: ***<img src=fig.jp alt="Esta é uma imagem legal">***

5 - TABELAS

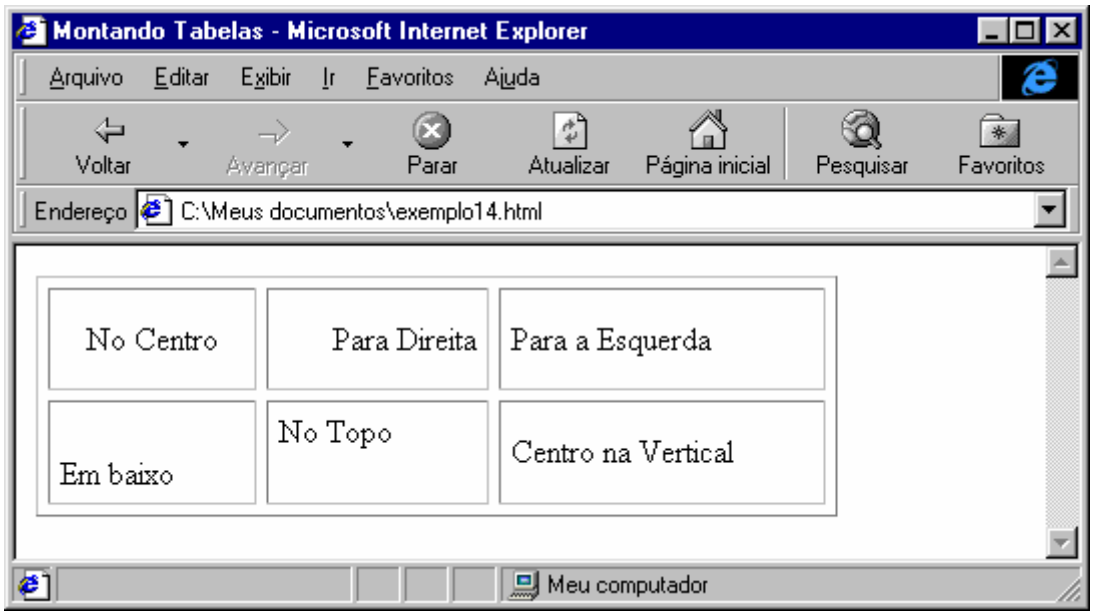
Nesta etapa conheceremos os TAGS responsáveis pela construção de tabelas. Utilizaremos a tabela como recurso para a definição dos layouts de nossas páginas. Você pode também controlar as dimensões de sua tabela. Bem como os elementos dentro da tabela também podem ser alinhados da mesma que um parágrafo. Veja mais um exemplo através da listagem **1.6**

Listagem 1.6

```
<html>
   <head><title>Montando Tabelas</title></head>
     <body>
           <table     border=1     width=400     height=120
           cellpadding=5 cellspacing=5>
             <tr>
                 <td align=center>No Centro</td>
                <td align=right>Para Direita</td>
                 <td align=left>Para a Esquerda</td>
             </tr>
             <tr>
                 <td valign=bottom>Em baixo</td>
                <td valign=top>No Topo</td>
                 <td valign=middle>Centro na Vertical</td>
             </tr>
          </table>
      </body>
</html>
```

Fim da Listagem 1.6

Veja a figura do resultado no navegador.



Cor de Fundo
Você pode ainda adicionar cores no fundo da tabela ou apenas de uma célula que deseja. Lembra do atributo **bgcolor**, então é com este mesmo que você pode adicionar um cor padrão para tabela ou para a célula.

Veja o exemplo:
<table bgcolor=blue>
O resultado seria assim: Toda Tabela teria um preenchimento azul

Veja mais este exemplo:
<table>
<tr>
<td bgcolor=beige>Bege</td>
<td bgcolor=red> Vermelho<td>
O resultado seria assim: As células teriam cores diferentes

CONCEITO DOS TAGS USADOS

TAG	O QUE FAZ
<table>	Marca o início e o fim de uma tabela **Atributos** **Width** define a largura da tabela ou da célula **Height** define a altura da tabela ou da célula **Cellpadding** define a margem dentro das células **Cellspacing** define o espaço entre as células **Bgcolor** define a cor de fundo da tabela ou da célula
<tr>	Marca o início e o fim de uma linha
<td>	Marca o início e o fim de uma célula
border=”n”	Coloca uma borda na tabela, onde n é o valor em pixels da borda
align=left	Alinha o conteúdo da célula a esquerda na horizontal
align=right	Alinha o conteúdo da célula a direita na horizontal
align=center	Alinha o conteúdo da célula ao centro na horizontal
valign=top	Alinha o conteúdo da célula no topo (vertical)
valign=bottom	Alinha o conteúdo da célula na base da célula (vertical)
valign=middle	Alinha o conteúdo da célula no centro na vertica

Observação: O atributo de alinhamento ALIGN, faz o alinhamento na horizontal.
O atributo de alinhamento VALIGN, faz o alinhamento na vertical.

6 - LINKS

Estamos produzindo páginas de hipertextos, ou seja, textos que podem fazer ligações com outros textos, ligando páginas entre si e a WEB. Então, este pontos nós chamamos de links ou hyperlinks, âncoras de hipertexto, todos com a mesma função, de através de um único clique sobre a frase ou imagem conduzir a algum lugar no site ou na WEB.
O Tag responsável é o <A>, onde dentro deste, através de um atributo coloco a referência, ou seja, A URL (**Uniform Resource Locator**). Como podemos ver na listagem 1.7.

Listagem 1.7

```
<html>
     <head><title>Estudando Links</title></head>
     <body>
          <a href=exem1.html>Exemplo1</a>
          <a href=exem2.html>Exemplo2</a>
          <a href=exem3.html><img src=bola.gif></a>Imagens
    </body>
</html>
```

Fim da Listagem 1.7

CONCEITO DOS TAGS USADOS

TAG	**O QUE FAZ**
<a>	Marca o início e o fim de um link Atributos **href** = define a endereço do link. **Text** = define a cor do texto padrão da página **Background** = permite inserir uma imagem como fundo da página

7 - Formulários HTML

Uma das formas de se enviar dos para uma pagina é atravez dos formulários. E como todo elemento de uma pagina o formulário é montado atravez de TAGS. Devendo começar com a TAG:

```
<form name="" action="" method="" enctype="">
      (textos e elementos do form)
</form>
```

Onde temos:

- name: o identificador do formulário. Utilizado principalmente em Scripts client-side (JavaScript);
- action: nome do script que receberá os dados do formulário ao ser submetido. Mais à frente estão abordadas as maneiras de tratar esses dados recebidos;
- method: método de envio dos dados: get ou post;

- enctype: formato em que os dados serão enviados. O default é urlencoded. Se for utilizado um elemento do tipo upload de arquivo (file) é preciso utilizar o tipo multipart/form-data.

Cada elemento do formulário deve possuir um nome que irá identificá-lo no momento em que o script indicado no ACTION for tratar os dados.

AS TAGS

Muitos elementos de um formulário html são definidos pela tag <input>. Cada tipo de elemento possui parâmetros próprios, mas todos possuem pelo menos dois parâmetros em comum: type, que define o tipo de elemento, e name, que como já foi dito define o nome daquele elemento.

- **Campo de Texto**

  <input type="text" name="" value="" size="" maxlength="">

O campo mais comum em formulários. Exibe na tela um campo para entrada de texto com apenas uma linha.

Parâmetros:

Value – o valor pré-definido do elemento, que aparecerá quando a página for carregada;

Size – O tamanho do elemento na tela, em caracteres;

Maxlength – O tamanho máximo do texto contido no elemento, em caracteres;

- **Campo de Texto com Máscara**

  <input type="password" name="" value="" size="" maxlength="">

Tipo de campo semelhante ao anterior, com a diferença que neste caso os dados digitados são substituídos por asteriscos, e por isso são os mais recomendados para campos que devam conter senhas. Os parâmetros são os mesmo que para o campo de texto.

- **Campo de Texto oculto**

  <input type="hidden" name="" value="" size="" maxlength="">

Tipo de campo semelhante ao anterior, com a diferença que neste caso os dados não são visualizados na pagina. Por isso são recomendados para campos de controle que devam conter informações que devem ser reenviadas ao servidor. Os parâmetros são os mesmo que para o campo de texto.

- **Checkbox**

  <input type="checkbox" name="" value="" checked>

Utilizado para campos de múltipla escolha, onde o usuário pode marcar mais de uma opção.

Parâmetros:

Value – o valor que será enviado ao servidor quando o formulário for submetido, no caso do campo estar marcado

Checked – O estado inicial do elemento. Quando presente, o elemento já aparece marcado;

- **Radio Button**

  <input type="radio" name="" value="" checked>

Utilizado para campos de múltipla escolha, onde o usuário pode marcar apenas uma opção. Para agrupar vários elementos deste tipo, fazendo com que eles sejam exclusivos, basta atribuir o mesmo nome a todos do grupo. Os parâmetros são os mesmo que para o Checkbox.

- **Button**

```
<input type="button" name="" value="">
```

Utilizado normalmente para ativar funções de scripts client-side (JavaScript, por exemplo). Sem essa utilização, não produz efeito algum.
Parâmetros:
Value – o texto que aparecerá no corpo do botão.

- **Submit Button**

```
<input type="submit" name="" value="">
```

Utilizado para enviar os dados do formulário para o script descrito na seção "action" da definição do formulário. Mesmos parâmetros do Button.

- **Reset Button**

```
<input type="reset" name="" value="">
```

Utilizado para fazer todos os campos do formulário retornem ao valor original, quando a página foi carregada. Bastante utilizado como botão "limpar", mas na realidade só restaura os valores iniciais do formulario . Mesmos parâmetros do Button.

- **TextArea**

```
<textarea cols="" rows="" name="" wrap="">texto</textarea>
```

Exibe na tela uma caixa de texto, com o tamanho definido pelos parâmetros "cols" e "rows".
Parâmetros:
Cols – número de colunas do campo, em caracteres;
Rows– número de linhas do campo, em caracteres;
Wrap – Maneira como são tratadas as quebras de linha automáticas. O valor soft faz com que o texto "quebre" somente na tela, sendo enviado para o servidor o texto da maneira como foi digitado; O valor "hard" faz com que seja enviado para o servidor da maneira como o texto aparece na tela, com todas as quebras de linhas inseridas automaticamente; o valor "off" faz com que o texto não quebre na tela e nem quando enviado ao servidor.
Value – O elemento do tipo textarea não possui o parâmetro "value". O valor pré-definido do campo é o texto que fica entre as tags <textarea> e </textarea>.

- **Select**

```
<select name="" size="" multiple>
        <option value="">texto</option>
</select>
```

Se o parâmetro "size" tiver o valor 1 e não houver o parâmetro "multiple", exibe na tela uma "combo box". Caso contrário, exibe na tela uma "select list".
Parâmetros:
Size – número de linhas exibidas. Default: 1;

Multiple – parâmetro que, se presente, permite que sejam selecionadas duas ou mais linhas, através das teclas Control ou Shift;
option – Cada item do tipo “option” acrescenta uma linha ao select;
value – Valor a ser enviado ao servidor se aquele elemento for selecionado. Default: o texto do item;
text – valor a ser exibido para aquele item. Não é definido por um parâmetro, mas pelo texto que fica entre as tags <option> e </option>

8 - Tabela de Acentuação/ Caracteres Especiais

As notações presentes em marcações, devem ser representadas com notações especiais para que possam ser exibidas em tela. Estas notações especiais sempre se iniciam por **&** (e comercial) e encerram-se com **;** (ponto e vírgula).

Notação	**descrição**	**aparência**
<	“Maior que”	**>**
>	“Menor que”	**<**
&	E comercial	**&**
"	aspas duplas	**“**
®	marca registrada	**®**
&copy	copyrights	**©**

É possível utilizar facilidades de acentuação (padrão do Windows por exemplo) em documentos HTML. No entanto, desta forma, apenas poderá visualizar a acentuação o computador que reconhecer este padrão específico.

Segue abaixo o padrão para acentuação- ISO Latin-1 alphabet -, semelhante ao utilizado para exibir caracteres das marcações, e que poderá ser visualizado pela grande maioria das máquinas.

Caracter	**Notação**
Acento agudo	&xacute; onde x é uma letra qualquer, maiúscula ou minúscula
Acento grave	&xgrave; onde x é uma letra qualquer, maiúscula ou minúscula
Acento circunflexo	◯ onde x é uma letra qualquer, maiúscula ou minúscula
Letra com til	&xtilde; onde x é uma letra qualquer, maiúscula ou minúscula
Letra com trema	&xuml; onde x é uma letra qualquer, maiúscula ou minúscula
Letras unidas	&Aelig; = **Æ** e æ = **æ**
Letra com argola	Å = **Å** e &aring = **å**
Cedilha	Ç = **Ç** e &ccedil= **ç**
O cortado	Ø = **Ø** e ø = **ø**
THORN maiúsculo	&THORN = **Þ**(Icelandic)
thorn minúsculo	&thorn= **þ** (Icelandic)
eth minúsculo	&eth **ð** (Icelandic)
eth maiúsculo	Ð= **Ð** (Icelandic)
Caracter alemão	&szlig = **ß**

PHP

1 - TAGS DO PHP

Assim como as TAGS básicas do HTML, o código em PHP é delimitado pelas suas TAGS. Podendo ser elas:

- <?php *CÓDIGO* ?>
- <? *CÓDIGO*?>
- <% *CÓDIGO %*>
- <script language="PHP"> *CÓDIGO* </script>

O tipo de *TAGS* mais utilizado é o segundo, que consiste em uma "abreviação" do primeiro. Para utilizá-lo, é necessário habilitar a opção *short-tags* na configuração do PHP. O último tipo serve para facilitar o uso por programadores acostumados à sintaxe de ASP. Para utilizá-lo também é necessário habilitá-lo no PHP, através do arquivo de configuração *php.ini.*
Para fins deste manual a TAG tomada como padrão será a "<?" para abrir o bloco. E "?>" para fechar o bloco. Como no exemplo da listagem **2.1.**

Listagem 2.1
```
<html>
     <head><title>Estudando PHP</title></head>
     <body>
     <h2>Exemplo de Código</h2>
<p><?
     echo "Só um exemplo";
?></p>
    </body>
</html>
```
Fim da Listagem 2.1

Como dá pra ver acima o código em PHP fica embutido no HTML sendo valido aplicar as formatações do mesmo como a de parágrafo usada no exemplo. Cada vez que se abre as TAGS do PHP dissemos que se abriu um bloco do PHP.

SEPARADOR DE INSTRUÇÕES
Entre cada instrução em PHP é preciso utilizar o ponto-e-vírgula. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

NOMES DE VARIÁVEIS
Toda variável em PHP tem seu nome composto pelo caracter $ e uma string, que deve iniciar por uma letra ou o caracter "_". **PHP é case sensitive**, ou seja, as variáveis $vivas e $VIVAS são diferentes. Por isso é preciso ter muito cuidado ao definir os nomes das variáveis.

2 - TIPOS

TIPOS SUPORTADOS

- **Inteiros (integer ou long)**

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

```
$php = 1234;# inteiro positivo na base decimal
$php = -234; # inteiro negativo na base decimal
$php = 0234;# inteiro na base octal-simbolizado pelo 0
$php = 0x34;# inteiro na base hexadecimal(simbolizado  pelo 0x)
```

- **Números em Ponto Flutuante (double ou float)**

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

```
$php = 1.234;
$php = 23e4;# equivale a 230.000
```

- **Strings**

Strings podem ser atribuídas de duas maneiras:

a) utilizando aspas simples (') - Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \' - ver tabela abaixo)

b) utilizando aspas duplas (") - Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

```
<?
$teste = "Brasil";
$php = '---$teste--\n';
echo "$php";
?>
```

A saída desse script será: "---$teste--\n".

```
<?
$teste = "Brasil";
$php = "---$teste---\n";
echo "$php";
?>
```

A saída desse script será: "---Brasil--" (com uma quebra de linha no final).

A tabela seguinte lista os caracteres de escape:

Sintaxe	Significado
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\ )
\$	O símbolo $
\’	Aspa simples
\”	Aspa dupla

- **Arrays**

Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados. Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso. Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros. Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo

Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes

podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

```
<?
$cor[1] = “vermelho”;
$cor[2] = “verde”;
$cor[3] = “azul”;
$cor[“teste”] = 1;
?>
```

Equivalentemente, pode-se escrever:

```
<?
$cor = array(1 => “vermelho, 2 => “verde, 3 => “azul”, “teste => 1);
?>
```

- **Objetos**

Um objeto pode ser inicializado utilizando o comando *new* para instanciar uma classe para uma variável.

```
class teste {
        function nada() {
                echo “nada”;
        }
}
$php = new teste;
$php -> nada();
```

A utilização de objetos será mais detalhada mais à frente.

BOOLEANOS

PHP não possui um tipo booleano, mas é capaz de avaliar expressões e retornar *true* ou *false*, através do tipo integer: é usado o valor 0 (zero) para representar o estado *false*, e qualquer valor diferente de zero (geralmente 1) para representar o estado *true*.

3 - OPERADORES

ARITMÉTICOS

Só podem ser utilizados quando os operandos são números (integer ou float). Se forem de outro tipo, terão seus valores convertidos antes da realização da operação.

+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo

DE STRINGS

Só há um operador exclusivo para strings:

.	concatenação

DE ATRIBUIÇÃO

Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído. No caso dos operadores derivados de atribuição, a

operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro. A atribuição é sempre por valor, e não por referência.

=	atribuição simples
+=	atribuição com adição
-=	atribuição com subtração
*=	atribuição com multiplicação
/=	atribuição com divisão
%=	atribuição com módulo
.=	atribuição com concatenação

BIT A BIT

Comparam dois números bit a bit.

&	“e” lógico
|	“ou” lógico
^	ou exclusivo
~	não (inversão)
<<	shift left
>>	shift right

LÓGICOS

Utilizados para inteiros representando valores booleanos

and	“e” lógico
or	“ou” lógico
xor	ou exclusivo
!	não (inversão)
&&	“e” lógico
||	“ou” lógico

COMPARAÇÃO

As comparações são feitas entre os valores contidos nas variáveis, e não as referências. Sempre retornam um valor booleano.

==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

EXPRESSÃO CONDICIONAL

Existe um operador de seleção que é ternário. Funciona assim:

(expressao1)?(expressao2):(expressao3)

O interpretador PHP avalia a primeira expressão. Se ela for verdadeira, a expressão retorna o valor de expressão2. Senão, retorna o valor de expressão3.

DE INCREMENTO E DECREMENTO

Podem ser utilizados de duas formas: antes ou depois da variável. Quando utilizado antes, retorna o valor da variável antes de incrementá-la ou decrementá-la. Quando utilizado depois, retorna o valor da variável já incrementado ou decrementado.

++	incremento
--	decremento

```
$a = $b = 10; // $a e $b recebem o valor 10
$c = $a++; // $c recebe 10 e $a passa a ter 11
$d = ++$b; // $d recebe 11, valor de $b já incrementado
```

ORDEM DE PRECEDÊNCIA DOS OPERADORES

Precedência	Associatividade	Operadores
1.	Esquerda	,
2.	Esquerda	or
3.	Esquerda	xor
4.	Esquerda	and
5.	Direita	print
6.	Esquerda	= += -= *= /= .= %= &= != ~= <<= >>=
7.	Esquerda	? :
8.	Esquerda	||
9.	Esquerda	&&
10.	Esquerda	|
11.	Esquerda	^
12.	Esquerda	&
13.	não associa	== !=
14.	não associa	< <= > >=
15.	Esquerda	<< >>
16.	Esquerda	+ - .
17.	Esquerda	* / %
18.	Direita	! ~ ++ -- (int) (double) (string) (array) (object) @
19.	Direita	[
20.	não associa	new

4 - ESTRUTURAS DE CONTROLE

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando, portanto, descrever a sintaxe de cada uma delas, resumindo o funcionamento.

BLOCOS

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em PHP são delimitados pelos caracteres “{” e “}”. A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador.

```
if ($x == $y)
        comando1;
comando2;
```

```
if ($x == $y){
        comando1;
        comando2;
}
```

COMANDOS DE SELEÇÃO

Também chamados de condicionais, os comandos de seleção permitem executar comandos ou blocos de comandos com base em testes feitos durante a execução.

- **if**

O mais trivial dos comandos condicionais é o if. Ele testa a condição e executa o comando indicado se o resultado for true (valor diferente de zero). Ele possui duas sintaxes:

```
if (expressão)                  if (expressão){
        comando;                        comando1;
                                        comando2;
                                        comando3;
                                        comando4;
                                }
```

Para incluir mais de um comando no if da primeira sintaxe, é preciso utilizar um bloco, demarcado por chaves.

O **else** é um complemento opcional para o if. Se utilizado, o comando será executado se a expressão retornar o valor false (zero). Suas duas sintaxes são:

```
if (expressão)                  if (expressão){
        comando1;                       comando1;
else                                    comando2;
        comando2;               }
                                else{
                                        comando3;
                                        comando4;
                                }
```

A seguir, temos um exemplo do comando if utilizado com else:

```
if ($a > $b)
        $maior = $a;
else
        $maior = $b;
```

O exemplo acima coloca em $maior o maior valor entre $a e $b.

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
        comando1;
else
        if (expressao2)
                comando2;
        else
                if (expressao3)
                        comando3;
                else
                        comando4;
```

Foi criado o comando, também opcional elseif. Ele tem a mesma função de um else e um if usados seqüencialmente, como no exemplo acima. Num mesmo if podem ser utilizados diversos elseif's, ficando essa utilização a critério do

programador, que deve zelar pela legibilidade de seu script.

O comando elseif também pode ser utilizado com dois tipos de sintaxe. Em resumo, a sintaxe geral do comando if fica das seguintes maneiras:

```
if (expressao1)
        comando;
[ elseif (expressao2)
        comando; ]
[ elseif (expressao3)
        comando; ]
[ else
        comando; ]
```

- **switch**

O comando switch atua de maneira semelhante a uma série de comandos if na mesma expressão. Freqüentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável. Quando isso for necessário, deve-se usar o comando switch.

```
switch ($i) {
        case 0:
                print “i é igual a zero”;
        break;
        case 1:
                print “i é igual a um”;
        break;
        case 2:
                print “i é igual a dois”;
        break;
}
```

É importante compreender o funcionamento do switch para não cometer enganos. O comando switch testa linha a linha os cases encontrados, e a partir do momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outro teste, até o fim do bloco. por isso usa-se o comando break, quebrando o fluxo e fazendo com que o código seja executado da maneira desejada. Veremos mais sobre o break mais adiante. Veja o exemplo:

```
switch ($i) {
        case 0:
                print “i é igual a zero”;
        case 1:
                print “i é igual a um”;
        case 2:
                print “i é igual a dois”;
}
```

No exemplo acima, se $i for igual a zero, os três comandos “print” serão executados. Se $i for igual a 1, os dois últimos “print” serão executados. O comando só funcionará da maneira desejada se $i for igual a 2.

COMANDOS DE REPETIÇÃO

- **while**

O while é o comando de repetição (laço) mais simples. Ele testa uma condição e executa um comando, ou um bloco de comandos, até que a condição testada seja falsa. Assim como o if, o while também possui duas sintaxes alternativas:

```
while (<expressão>)
        <comando>;
        OU
while (<expressão>){
        <comando1>;
        <comando2>;
}
```

A expressão só é testada a cada vez que o bloco de instruções termina, além do teste inicial. Se o valor da expressão passar a ser false no meio do bloco de instruções, a execução segue até o final do bloco. Se no teste inicial a condição for avaliada como false, o bloco de comandos não será executado.

O exemplo a seguir mostra o uso do while para imprimir os números de 1 a 10:

```
$i = 1;
while ($i <=10)
print $i++;
```

- **do... while**

O laço do...while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço do...while possui apenas uma sintaxe, que é a seguinte:

```
do {
        <comando>
        <comando>
} while (<expressão>);
```

O exemplo utilizado para ilustrar o uso do while pode ser feito da seguinte maneira utilizando o do... while:

```
$i = 0;
do {
        print ++$i;
} while ($i < 10);
```

- **for**

O tipo de laço mais complexo é o for. Para os que programam em C, C++ ou Java, a assimilação do funcionamento do for é natural. Mas para aqueles que estão acostumados a linguagens como Pascal, há uma grande mudança para o uso do for. As três sintaxes permitidas são:

```
for (<inicialização>;<condição>;<incremento ou decremento>)
        <comando>;

for (<inicialização>;<condição>;<incremento ou decremento>){
        <comando>;
        <comando>;
}
```

As três expressões que ficam entre parênteses têm as seguintes finalidades:

Inicialização: comando ou seqüência de comandos a serem realizados antes do inicio do laço. Serve para inicializar variáveis.

Condição: Expressão booleana que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira (valor diferente de zero) os comandos serão executados.
Incremento: Comando executado ao final de cada execução do laço.

QUEBRA DE FLUXO

- **Break**

O comando break pode ser utilizado em laços de do, for e while, além do uso já visto no comando switch. Ao encontrar um break dentro de um desses laços, o interpretador PHP para imediatamente a execução do laço, seguindo normalmente o fluxo do script.

```
while ($x > 0) {
...
        if ($x == 20) {
                echo “erro! x = 20”;
        break;
...
}
```

No trecho de código acima, o laço while tem uma condição para seu término normal ($x <= 0), mas foi utilizado o break para o caso de um término não previsto no início do laço. Assim o interpretador seguirá para o comando seguinte ao laço.

- **Continue**

O comando continue também deve ser utilizado no interior de laços, e funciona de maneira semelhante ao break, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Vejamos o exemplo:

```
for ($i = 0; $i < 100; $i++) {
        if ($i % 2) continue;
                echo “ $i “;
}
```

O exemplo acima é uma maneira ineficiente de imprimir os números pares entre 0 e 99. O que o laço faz é testar se o resto da divisão entre o número e 2 é 0. Se for diferente de zero (valor lógico true) o interpretador encontrará um continue, que faz com que os comandos seguintes do interior do laço sejam ignorados, seguindo para a próxima iteração.

5 - FUNÇÕES

DEFININDO FUNÇÕES

A sintaxe básica para definir uma função é:

```
function nome_da_função([arg1, arg2, arg3]) {
        Comandos;
        ... ;
        [return <valor de retorno>];
}
```

Qualquer código PHP válido pode estar contido no interior de uma função. Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser

declarado, sendo necessário que o programador esteja atento para que a função retorne o tipo desejado. É recomendável que esteja tudo bem documentado para facilitar a leitura e compreensão do código. Para efeito de documentação, utiliza-se o seguinte formato de declaração de função:

```
tipo function nome_da_funcao(tipo arg1, tipo arg2, ...);
```

Este formato só deve ser utilizado na documentação do script, pois o PHP não aceita a declaração de tipos. Isso significa que em muitos casos o programador deve estar atento ao tipos dos valores passados como parâmetros, pois se não for passado o tipo esperado não é emitido nenhum alerta pelo interpretador PHP, já que este não testa os tipos.

VALOR DE RETORNO

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum.

Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou arrays.

ARGUMENTOS

É possível passar argumentos para uma função. Eles devem ser declarados logo após o nome da função, entre parênteses, e tornam-se variáveis pertencentes ao escopo local da função. A declaração do tipo de cada argumento também é utilizada apenas para efeito de documentação.

```
function imprime($texto){
        echo $texto;
}
imprime(“teste de funções”);
```

PASSAGEM DE PARÂMETROS POR REFERÊNCIA

Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original.

```
function mais5($numero) {
$numero += 5;
}
$a = 3;
mais5($a); //$a continua valendo 3
```

No exemplo acima, como a passagem de parâmetros é por valor, a função mais5 é inútil, já que após a execução sair da função o valor anterior da variável é recuperado. Se a passagem de valor fosse feita por referência, a variável $a teria 8 como valor. O que ocorre normalmente é que ao ser chamada uma função, o interpretador salva todo o escopo atual, ou seja, os conteúdos das variáveis. Se uma dessas variáveis for passada como parâmetro, seu conteúdo fica preservado, pois a função irá trabalhar na verdade com uma cópia da variável.7 Porém, se a passagem de parâmetros for feita por referência, toda alteração que a função realizar no valor passado como parâmetro afetará a variável que o contém.

Há duas maneiras de fazer com que uma função tenha parâmetros passados por referência: indicando isso na declaração da função, o que faz com que a passagem de parâmetros sempre seja assim; e também na própria chamada da função. Nos dois casos utiliza-se o modificador “&”. Vejamos um exemplo que ilustra os dois casos:

```
function mais5(&$num1, $num2) {
        $num1 += 5;
        $num2 += 5;
}
$a = $b = 1;
mais5($a, $b); /* Só $num1 terá seu valor alterado, pois a passagem por referência está definida na declaração da função. */
mais5($a, &$b); /* Aqui as duas variáveis terão seus valores alterados. */
```

CONTEXTO

O contexto é o conjunto de variáveis e seus respectivos valores num determinado ponto do programa. Na chamada de uma função, ao iniciar a execução do bloco que contém a implementação da mesma é criado um novo contexto, contendo as variáveis declaradas dentro do bloco, ou seja, todas as variáveis utilizadas dentro daquele bloco serão eliminadas ao término da execução da função.

ESCOPO

O escopo de uma variável em PHP define a porção do programa onde ela pode ser utilizada. Na maioria dos casos todas as variáveis têm escopo global. Entretanto, em funções definidas pelo usuário um escopo local é criado. Uma variável de escopo global não pode ser utilizada no interior de uma função sem que haja uma declaração.

```
$php = “Testando”;
function Teste() {
        echo $php;
}
Teste();
```

O trecho acima não produzirá saída alguma, pois a variável $php é de escopo global, e não pode ser referida num escopo local, mesmo que não haja outra com nome igual que cubra a sua visibilidade. Para que o script funcione da forma desejada, a variável global a ser utilizada deve ser declarada.

```
$php = “Testando”;
function Teste() {
        global $php;
        echo $php;
}
Teste();
```

Uma declaração “global” pode conter várias variáveis, separadas por vírgulas. Uma outra maneira de acessar variáveis de escopo global dentro de uma função é utilizando um array pré-definido pelo PHP cujo nome é $GLOBALS. O índice para a variável referida é o próprio nome da variável, sem o caracter $. O exemplo acima e o abaixo produzem o mesmo resultado:

```
$php = "Testando";
function Teste() {
        echo $GLOBALS["php"]; // imprime $php
        echo $php; // não imprime nada
}
Teste();
```

6 – VARIÁVEIS

O MODIFICADOR STATIC

Uma variável estática é visível num escopo local, mas ela é inicializada apenas uma vez e seu valor não é perdido quando a execução do script deixa esse escopo. Veja o seguinte exemplo:

```
function Teste() {
        $a = 0;
        echo $a;
        $a++;
}
```

O último comando da função é inútil, pois assim que for encerrada a execução da função a variável $a perde seu valor. Já no exemplo seguinte, a cada chamada da função a variável $a terá seu valor impresso e será incrementada:

```
function Teste() {
        static $a = 0;
        echo $a;
        $a++;
}
```

O modificador static é muito utilizado em funções recursivas, já que o valor de algumas variáveis precisa ser mantido. Ele funciona da seguinte forma: O valor das variáveis declaradas como estáticas é mantido ao terminar a execução da função. Na próxima execução da função, ao encontrar novamente a declaração com static, o valor da variável é recuperado.

Em outras palavras, uma variável declarada como static tem o mesmo “tempo de vida” que uma variável global, porém sua visibilidade é restrita ao escopo local em que foi declarada e só é recuperada após a declaração.

```
function Teste() {
        echo "$a";
        static $a = 0;
        $a++;
}
```

O exemplo acima não produzirá saída alguma. Na primeira execução da função, a impressão ocorre antes da atribuição de um valor à função e, portanto o conteúdo de $a é nulo (string vazia). Nas execuções seguintes da função Teste() a impressão ocorre antes da recuperação do valor de $a e, portanto nesse momento seu valor ainda é nulo. Para que a função retorne algum valor o modificador static deve ser utilizado.

VARIÁVEIS VARIÁVEIS

O PHP tem um recurso conhecido como variáveis variáveis, que consiste em variáveis cujos nomes também são variáveis. Sua utilização é feita através do duplo cifrão ($$).

```
$a = “teste”;
$$a = “Daniel Garcia”;
```

O exemplo acima e equivalente ao seguinte:

```
$a = “teste”;
$teste = “Daniel Garcia”;
```

VARIÁVEIS ENVIADAS PELO NAVEGADOR

Para interagir com a navegação feita pelo usuário, é necessário que o PHP possa enviar e receber informações para o software de navegação. A maneira de enviar informações, como já foi visto anteriormente, geralmente é através de um comando de impressão, como o *echo*. Para receber informações vindas do navegador através de um *link* ou um formulário html o PHP utiliza as informações enviadas através da URL. Por exemplo: se seu script php está localizado em "http://localhost/teste.php3" e você o chama com a url "http://localhost/teste.php3?php=teste", automaticamente o PHP criará uma variável com o nome $php contendo a string "teste". Note que o conteúdo da variável está no formato urlencode. Os formulários html já enviam informações automaticamente nesse formato, e o PHP decodifica sem necessitar de tratamento pelo programador.

VERIFICANDO O TIPO DE UMA VARIÁVEL

Por causa da tipagem dinâmica utilizada pelo PHP, nem sempre é possível saber qual o tipo de uma variável em determinado instantes não contar com a ajuda de algumas funções que ajudam a verificar isso. A verificação pode ser feita de duas maneiras:

- **Função que retorna o tipo da variável**

Esta função é a gettype. Sua assinatura é a seguinte:

string **gettype**(mixed var);

A palavra "mixed" indica que a variável var pode ser de diversos tipos.
A função gettype pode retornar as seguintes strings: "integer", "double", "string", "array", "object" e "unknown type".

- **Funções que testam o tipo da variável**

São as funções is_int, is_integer, is_real, is_long, is_float, is_string, is_array e is_object. Todas têm o mesmo formato, seguindo modelo da assinatura a seguir:

int **is_integer**(mixed var);

Todas essas funções retornam true se a variável for daquele tipo, e false em caso contrário.

DESTRUINDO UMA VARIÁVEL

É possível desalocar uma variável se ela não for usada posteriormente através da função unset, que tem a seguinte assinatura:

int **unset**(mixed var);

A função destrói a variável, ou seja, libera a memória ocupada por ela, fazendo com que ela deixe de existir. Se mais na frente for feita uma chamada á variável, será criada uma nova variável de mesmo nome e de conteúdo vazio, a não ser que a chamada seja pela função isset. Se a operação for bem sucedida, retorna true.

8 - NOÇÕES DE SQL

ESTRUTURA DAS TABELAS

- **Comando Create**

Este comando permite a criação de tabelas no banco de dados ou mesmo de bases de dados.

```
CREATE TABLE < nome_tabela > (
        nome_atributo1 < tipo > [ NOT NULL ],
         nome_atributo2 < tipo > [ NOT NULL ],
         .....
         nome_atributoN < tipo > [ NOT NULL ]
) ;
```

onde:
nome_table - indica o nome da tabela a ser criada.
nome_atributo - indica o nome do campo a ser criado na tabela.
tipo - indica a definição do tipo de atributo (integer(n), char(n), ...).

- **Comando Drop**

Este comando elimina a definição da tabela, seus dados e referências.

```
DROP TABLE  < nome_tabela > ;
```

MANIPULANDO DADOS DAS TABELAS

- **Comando SELECT**

Permite recuperar informações existentes nas tabelas.

```
SELECT [DISTINCT] expressao [AS nom-atributo]
[FROM from-list]
[WHERE condicao]
[ORDER BY attr_name1 [ASC | DESC ]]
```

onde:
DISTINCT : Para eliminar linhas duplicadas na saída.
Expressao: Define os dados que queremos na saída, normalmente uma ou mais colunas de uma tabela da lista FROM.
AS nom-atributo : um *alias* para o nome da coluna, exemplo:
FROM : lista das tabelas na entrada
WHERE : critérios da seleção
ORDER BY : Critério de ordenação das tabelas de saída. ASC ordem ascendente, DESC ordem descendente

Exemplo:

```
SELECT cidade, estado from brasil where populacao > 100000;
```

- **Comando INSERT**

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

```
INSERT INTO destino [(campo1[, campo2[, ...]])]
VALUES (valor1[, valor2[, ...])
```

A instrução INSERT INTO tem as partes abaixo:
Destino- O nome da tabela ou consulta em que os registros devem ser anexados.

campo1, campo2 - Os nomes dos campos aos quais os dados devem ser anexados
valor1, valor2 - Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante. Você deve separar os valores com uma vírgula e colocar os campos de textos entre aspas (" ").

- **Comando UPDATE**

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

```
UPDATE tabela
SET campo1 = valornovo, ...
WHERE critério;
```

Onde:
Tabela - O nome da tabela cujos os dados você quer modificar.
Valornovo - Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.
critério - Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizados.

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo.
UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

- **Comando DELETE**

Remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE.

```
DELETE [tabela.*]
FROM tabela
WHERE critério
```

onde:
tabela.* - O nome opcional da tabela da qual os registros são excluídos.
tabela - O nome da tabela da qual os registros são excluídos.
critério - Uma expressão que determina qual registro deve ser excluído.

O QUE HÁ EM COMUM ENTRE: DELETE X UPDATE

Você deve lembrar-se de utilizar uma condição toda vez que atualizar ou excluir. Se você não fizer isso, todas as linhas na tabela sofrerão a mesma alteração ou exclusão. Mesmo os programadores mais experientes esquecem a condição, para seu grande constrangimento profissional. Cuidado para não esquecer a condição.

9 - ACESSANDO O MYSQL VIA PHP

ESTABELECENDO CONEXÕES

Para acessar bases de dados num servidor mySQL, é necessário antes estabelecer uma conexão. Para isso, deve ser utilizado o comando mysql_connect, como pode ser verificado a seguir:

```
int mysql_connect(string [host[:porta]] , string [login] , string [senha] );
```

O valor de retorno é um inteiro que identifica a conexão, ou falso se a conexão falhar.

Uma conexão estabelecida com o comando mysql_connect é encerrada ao final da execução do script. Para encerrá-la antes disso deve ser utilizado o comando mysql_close, que tem a seguinte assinatura:

```
int mysql_close(int [identificador da conexão] );
```

Se o identificador não for fornecido, a última conexão estabelecida será encerrada.

SELECIONANDO A BASE DE DADOS

Depois de estabelecida a conexão, é preciso selecionar a base de dados a ser utilizada, através do comando mysql_select_db, que segue o seguinte modelo:

```
int mysql_select_db(string base, int [conexao] );
```

Novamente, se o identificador da conexão não for fornecido, a última conexão estabelecida será utilizada.

REALIZANDO CONSULTAS

Para executar consultas SQL no mySQL, utiliza-se o comando mysql_query, que tem a seguinte assinatura:

```
int mysql_query(string query, int [conexao] );
```

Onde query é a expressão SQL a ser executada, sem o ponto-e-vírgula no final, e conexao é o identificador da conexão a ser utilizada. A consulta será executada na base de dados selecionada pelo comando mysql_select_db.

É bom lembrar que uma consulta não significa apenas um comando SELECT. A consulta pode conter qualquer comando SQL aceito pelo banco.

O valor de retorno é falso se a expressão SQL for incorreta, e diferente de zero se for correta. No caso de uma expressão SELECT, as linhas retornadas são armazenadas numa memória de resultados, e o valor de retorno é o identificador do resultado. Alguns comandos podem ser realizados com esse resultado. Como será visto a seguir.

APAGANDO O RESULTADO

```
int mysql_free_result(int result);
```

O comando mysql_free-result deve ser utilizado para apagar da memória o resultado indicado.

NÚMERO DE LINHAS

```
int mysql_num_rows(int result);
```

O comando mysql_num_rows retorna o número de linhas contidas num resultado.

UTILIZANDO OS RESULTADOS

Existem diversas maneiras de ler os resultados de uma query SELECT. As mais comuns serão vistas a seguir:

int mysql_result(int result, int linha, mixed [campo]);

Retorna o conteúdo de uma célula da tabela de resultados.
result é o identificador do resultado;
linha é o número da linha, iniciado por 0;
campo é uma string com o nome do campo, ou um número correspondente ao número da coluna. Se foi utilizado um alias na consulta, este deve ser utilizado no comando mysql_result.

Este comando deve ser utilizado apenas para resultados pequenos. Quando o volume de dados for maior, é recomendado utilizar um dos métodos a seguir:

array mysql_fetch_array(int result);

Lê uma linha do resultado e devolve um array, cujos índices são os nomes dos campos. A execução seguinte do mesmo comando lerá a próxima linha, até chegar ao final do resultado.

array mysql_fetch_row(int result);

Semelhante ao comando anterior, com a diferença que os índices do array são numéricos, iniciando pelo 0 (zero).