Índice

O que é PHP?

PHP (um acrônimo recursivo para "PHP: Hypertext Preprocessor") é uma linguagem de script Open Source de uso geral, muito utilizada e especialmente guarnecida para o desenvolvimento de aplicações Web embútivel dentro do HTML.
É uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a JavaScript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro. Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.
O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.

Client Side scripts

São códigos de programa que são processados pela estação cliente. Geralmente em aplicações voltadas à Internet, o código que é executado no cliente cuida apenas de pequenas consistências de telas e validações de entrada de dados.
Em se tratando de páginas web, os client-side scripts terão de ser processados por um browser. O maior problema de se utilizar este tipo de artifício em uma aplicação é a incompatibilidade de interpretação da linguagem entre os browsers. O Microsoft Internet Explorer, por exemplo, é capaz de interpretar o Visual Basic Script, porém o Netscape não o faz sem o auxílio de um plug in (que foi desenvolvido por terceiros). Há ainda o problema de versões muito antigas de navegadores, que não conseguem interpretar nenhum script.
Em grande parte das situações, não é possível exigir que o usuário final disponha de determinado produto para acessar a aplicação. Portanto é importante pesar todos estes fatores ao planejar alguma aplicação com client-side scripts.
A linguagem script mais indicada para se construir client-side scripts é o JavaScript, devido a sua compatibilidade com os dois browsers (Netscape e Microsoft Internet Explorer, que devem ser de versões iguais ou superiores a 3.0 e 4.0 respectivamente).

Server Side scripts

São códigos de programa que são processados no servidor. Devido a este fato, não é necessário preocupar-se com a linguagem que o código foi criado: o servidor é quem se encarrega de interpretá-lo e de devolver uma resposta para o cliente. Em páginas PHP, são esses códigos os maiores responsáveis pelos resultados apresentados.

Ambiente de desenvolvimento de páginas PHP

Como os arquivos PHP são arquivos do tipo texto (ASCII), eles podem ser escritos em um editor de textos comum – Edit, Notepad, Emacs, por exemplo. Existem também outros ambientes que proporcionam uma forma mais agradável de desenvolvimento, mas exige os mesmos conhecimentos do programador.

O que pode ser feito com PHP?

Basicamente, qualquer coisa que pode ser feita por algum programa CGI pode ser feita também com PHP, como coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber cookies.
O PHP pode ser utilizado na maioria dos sistemas operacionais, incluindo Linux, várias variantes Unix (incluindo HP-UX, Solaris e OpenBSD), Microsoft Windows, Mac OS X, RISC OS, e provavelmente outros. O PHP também é suportado pela maioria dos servidores web atuais, incluindo Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet Servers, Oreilly Website Pro Server, Caudium, Xitami, OmniHTTPd, e muitos outros. O PHP pode ser configurado como módulo para a maioria dos servidores, e para os outros como um CGI comum.
Com o PHP, portanto, você tem a liberdade para escolher o sistema operacional e o servidor web. Do mesmo modo, você pode escolher entre utilizar programação estrutural ou programação orientada a objeto, ou ainda uma mistura deles. Mesmo não desenvolvendo nenhum recurso padrão de OOP (Object Oriented Programming, Programação Orientada a Objetos) na versão atual do PHP, muitas bibliotecas de código e grandes aplicações (incluindo a biblioteca PEAR) foram escritos somente utilizando OOP.
Com PHP você não está limitado a gerar somente HTML. As habilidades do PHP incluem geração de imagens, arquivos PDF e animações Flash (utilizando libswf ou Ming) criados dinamicamente, on the fly. Você pode facilmente criar qualquer padrão texto, como XHTML e outros arquivos XML. O PHP pode gerar esses padrões e os salvar no sistema de arquivos, em vez de imprimi-los, formando um cache dinâmico de suas informações no lado do servidor.
Talvez a mais forte e mais significativa característica do PHP é seu suporte a uma ampla variedade de banco de dados. Escrever uma página que consulte um banco de dados é incrivelmente simples. Os seguintes bancos de dados são atualmente suportados:

Tabela de bancos suportados pelo PHP5		
Adabas D	Ingres	Oracle (OCI7 and OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (read-only)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Unix dbm	Informix	ODBC

Também foi providenciada uma abstração de banco de dados DBX permitindo a você utilizar qualquer banco de dados transparentemente com sua extensão. Adicionalmente,

o PHP suporta ODBC (Open Database Connection, ou Padrão Aberto de Conexão com Bancos de Dados), permitindo que você utilize qualquer outro banco de dados que suporte esse padrão mundial.
O PHP também tem suporte para comunicação com outros serviços utilizando protocolos como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (em Windows) e incontáveis outros. Você pode abrir sockets de rede e interagir diretamente com qualquer protocolo. O PHP também suporta o intercâmbio de dados complexos WDDX, utilizado em virtualmente todas as linguagens de programação para web. Falando de comunicação, o PHP implementa a instanciação de objetos Java e os utiliza transparentemente como objetos PHP. Você ainda pode usar sua extensão CORBA para acessar objetos remotos.
O PHP é extremamente útil em recursos de processamento de texto, do POSIX Estendido ou expressões regulares Perl até como interpretador para documentos XML. Para acessar e processar documentos XML, são suportados os padrões SAX e DOM. Você ainda pode usar nossa extensão XSLT para transformar documentos XML.
Utilizando o PHP no campo do e-commerce, você poderá usar as funções específicas para Cybescash, CyberMUT, Verysign Payflow Pro e CCVS, práticos sistemas de pagamento online.

Como surgiu a linguagem PHP?

A linguagem PHP foi concebida durante o outono de 1994 por Rasmus Lerdorf. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua home-page apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas. A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como “Personal Home Page Tools” (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas macros e alguns utilitários que rodavam “por trás” das home-pages: um livro de visitas, um contador e algumas outras coisas.
Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de PHP/FI, o “FI” veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (Form Interpreter). Ele combinou os scripts do pacote Personal Home Page Tools com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.
Estima-se que em 1996 PHP/FI estava sendo usado por cerca de 15.000 sites pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP. Ele deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por Zeev Suraski e Andi Gutmans, e esse novo interpretador foi à base para a versão 3.

A sintaxe

Quando o PHP interpreta um arquivo, ele simplesmente repassa o texto do arquivo até encontrar uma das tags especiais que lhe diz para começar a interpretar o texto como código PHP. O interpretador então executa todo o código que encontra, até chegar em uma tag de fechamento PHP, que novamente o coloca simplesmente repassando texto novamente. Este é o mecanismo que permite a inclusão de código PHP dentro do

HTML: qualquer coisa fora das tags PHP é deixada como encontrado, enquanto tudo dentro é interpretado e executado.
Há quatro conjuntos de tags que podem ser usadas para marcar blocos de código PHP. Delas, somente duas (<?php. . .?> e <script language="php">. . .</script>) são sempre disponíveis. As outras podem ser ativadas ou desativadas a partir do arquivo de configuração php.ini. Enquanto as formas reduzidas das tags ou no seu estilo ASP serem convenientes, elas não são portáveis em todas as versões. Além disso, se você pretende incluir código PHP em XML ou XHTML, você precisará usar a forma <?php ... ?> para compatibilidade com o padrão XML.

Delimitando o código PHP

O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php
comandos
?>

<script language=”php”>
comandos
</script>

<?
comandos
?>

<%
comandos
%>
```

O terceiro tipo consiste em uma “abreviação” do primeiro. Para utilizá-lo, é necessário habilitar a opção **short_open_tag** na configuração do PHP, tornando **on**. 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**, tornando **on** a opção **asp_tags**.

Alternagem avançada

```
<?php
if ($expressao) {
    ?>
    <strong>Isso é verdadeiro.</strong>
    <?php
} else {
    ?>
    <strong>Isto é falso.</strong>
    <?php
}
?>
```

Isso funciona como esperado porque quando o PHP encontra a tag de fechamento ?>, ele simplesmente começa a imprimir tudo até encontrar outra tag de abertura. Obviamente, o exemplo acima se aplica à exibição de grandes blocos de texto, uma vez que sair do modo de interpretação do PHP é geralmente mais eficiente que imprimir todo o texto através da construção de linguagem como o **echo( )** ou função **print( )** e outras.

Separador de instruções

Instruções são separadas da mesma forma que o C ou o Perl - cada instrução termina com um ponto e vírgula.
A tag de fechamento (?>) também implica no fim de uma instrução, então os exemplos seguintes são equivalentes:

```
<?php
    echo "Isto é um exemplo com vírgula";
?>

<?php echo "Isto é um outro exemplo sem vírgula" ?>
```

Comentários

O PHP suporta comentários do 'C', 'C++' e Unix shell. Por exemplo:

```
<?php
    echo "Isto é um exemplo"; //Comentário de uma linha como no
C++
    /* Isto é um comentário de mais de uma linha
       e aqui temos outra linha como em C */
    echo "Isto é um outro exemplo";
    echo "O último exemplo"; #Comentário no estilo Unix shell
?>
```

O comentário de uma linha só tem efeito até o fim da linha ou fim do bloco de código PHP atual, o que ocorrer primeiro.

```
<h1>Isto é um <?php # echo " simples";?> exemplo.</h1>
<p>No título acima você lerá 'Isto é um exemplo'.
```

Você precisa ser cuidadoso com comentários estilo 'C' encadeados, pois eles podem causar problemas em grandes blocos.

```
<?php
 /*
    echo "Isto é um teste"; /* Este comentário causará um erro
*/
 */
?>
```

Nota: É importante trabalhar com padrões. Procure utilizar o formato C para comentários e deixe de lado o formato shell.

Variáveis

São códigos em forma de palavras que carregam informações que se alteram "variam" no decorrer de uma instrução. As variáveis são muito úteis, pois elas permitem uma flexibilidade na programação, onde não são possíveis de antever determinadas situações.

Regras para nomear as variáveis

Para nomear suas variáveis, é preciso seguir algumas regras:
Toda variável em PHP tem seu nome composto pelo caractere **$** e uma string, que deve iniciar por uma letra ou o caractere "_".
PHP é ***case sensitive***, ou seja, as variáveis **$integrator** e **$INTEGRATOR** são diferentes. Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui algumas variáveis pré-definidas cujos nomes são formados por letras maiúsculas.
Passagem por referência
O PHP 5 oferece um outro meio de atribuir valores a variáveis: a atribuição por referência. Isto significa que a nova variável simplesmente referencia (em outras palavras, "torna-se um apelido para" ou "aponta para") a variável original. Alterações na nova variável afetam a original e vice-versa. Isto significa também que nenhuma cópia é realizada, de modo que a atribuição ocorre mais rapidamente. Entretanto, qualquer aumento de velocidade só será realmente notado em loops complexos ou em atribuições de grandes matrizes (arrays) ou objetos.

```
<?php
$item1 = 'Banana'; // Atribui o valor 'Banana' a variável $item1
$item2 = &$item1; // Referecia $item1 através de $item2.
$item2 = "O produto é $item2"; // Altera $item2...
echo $item2.'<br>';
echo $item1; // $item1 é alterado também.
?>
```

variaveis.php

Uma observação importante a se fazer: somente variáveis nomeadas podem ser atribuídas por referência.

Variáveis Predefinidas

O PHP oferece um grande número de variáveis predefinidas para qualquer script que ele execute. Muitas destas variáveis, entretanto, não podem ser completamente documentadas uma vez dependem de diversos fatores, como o servidor no qual scripts são executados, a versão e configuração deste servidor e outros.

Variáveis do servidor: $_SERVER

$_SERVER é um array contendo informações como headers, caminhos e localizações do script. Os itens deste array são criados pelo servidor web. Não há garantias que todos os servidores web gerem todas elas: alguns servidores talvez omitam algumas ou gerem outras que não estão listadas aqui. Mesmo assim, um grande número dessas variáveis está de acordo com a especificação CGI 1.1, então você pode esperar encontrá-las nesse array.
Esta é uma variável **superglobal**, ou automaticamente global. Isto significa que ela é disponível em todos os escopos (níveis) de um script. Você não precisa fazer um global **$_SERVER**; para poder acessá-la dentro de funções ou métodos, como era necessário com **$HTTP_SERVER_VARS** (Disponível em versões anteriores ao PHP 4.1.0).
$HTTP_SERVER_VARS contém a mesmas informações, mas ela não é uma superglobal (note que $HTTP_SERVER_VARS e $_SERVER são variáveis diferentes como também o PHP as manipula diferentemente).
A seguir você tem como exemplo algumas das utilizações da variável $_SERVER:

'PHP_SELF'

O nome do arquivo do script atualmente em uso, relativo ao documento raiz. Por exemplo, $_SERVER['PHP_SELF'] em um script com o endereço **http://integrator.com.br/php_self.php** pode ser **/php_self.php**.

```
<?php
            echo $_SERVER['PHP_SELF'];
?>
```

php_self.php

'SERVER_NAME'

O nome host do servidor onde o script atual é executado. Se o script está rodando em um host virtual, este será o valor definido para aquele host virtual.

```
<?php
            echo $_SERVER['SERVER_NAME'];
?>
```

server_name.php

'SERVER_SOFTWARE'

A string de identificação do servidor, fornecida nos headers quando respondendo a requests.

```
<?php
            echo $_SERVER['SERVER_SOFTWARE'];
?>
```

server_software.php

'SERVER_PROTOCOL'

Nome e número de revisão do protocolo de informação pelo qual a página foi requerida, por exemplo, HTTP/1.0.

```
<?php
                echo $_SERVER['SERVER_PROTOCOL'];
?>
```

server_protocol.php

'REQUEST_METHOD'

Contém o método de request utilizando para acessar a página. Geralmente **GET**, **HEAD**, **POST** ou **PUT**.
Com o exemplo abaixo, você terá o retorno do método de request **GET**, isso porque você só está acessando a página, e os dados não vieram de um formulário com método POST.

```
<?php
                echo $_SERVER['REQUEST_METHOD'];
?>
```

request_method.php

'QUERY_STRING'

A query string (string de solicitação), se houver, pela qual a página foi acessada. Para a demonstração do exemplo, a seguir, você ira criar duas páginas:

```
<html>
 <head><title>Exemplo de variáveis de servidor</title></head>
<body>
<a href="query_string.php?empresa=integrator">Vai até a página
de query_string.php</a>
</body>
</html>
```

env_query_string.php

```
<?php
                echo $_SERVER['QUERY_STRING'];
?>
```

query_string.php

O resultado será **empresa=integrator**. Essas informações foram recuperadas do cabeçalho **http**, da qual estava sendo passada pelo link. Veremos mais detalhes adiante, no uso de recuperação pelo método GET.

'DOCUMENT_ROOT'

O diretório raiz sob onde o script atual é executado, como definido no arquivos de configuração do servidor.

```
<?php
                echo $_SERVER['DOCUMENT_ROOT'];
```

```
?>
```

document_root.php

'HTTP_ACCEPT_LANGUAGE'

O conteúdo do header Accept-Language: da requisição atual, se houver. Exemplo pt-br.

```
<?php
                echo $_SERVER['HTTP_ACCEPT_LANGUAGE'];
?>
```

http_accept_language.php

'HTTP_USER_AGENT'

O conteúdo do header User_Agent: da requisição atual, se houver. É uma string denotando o agente de usuário pelo qual a página é acessada. Um exemplo típico é: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586) ou Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0).

```
<?php
                echo $_SERVER['HTTP_USER_AGENT'];
?>
```

http_user_agent.php

'REMOTE_ADDR'

O endereço IP de onde o usuário está visualizado a página atual.

```
<?php
                echo $_SERVER['REMOTE_ADDR'];
?>
```

remote_addr.php

Além de outras coisas, o usuário pode ser bloqueado ou você pode criar um contador personalizado.

'SCRIPT_FILENAME'

O caminho absoluto o script atualmente em execução.

```
<?php
                echo $_SERVER['SCRIPT_FILENAME'];
?>
```

script_filename.php

'PATH_TRANSLATED'

O caminho real do script relativo ao sistema de arquivos (não o document root), depois realizou todos os mapeamentos de caminhos (virtual-to-real). É semelhante ao SCRIPT_FILENAME.

Não é funcional no Windows.

```
<?php
                echo $_SERVER['PATH_TRANSLATED'];
?>
```

path_translated.php

'SCRIPT_NAME'

Contém o caminho completo do script atual. Útil para páginas que precisam apontar para elas mesmas (dinamicamente). Semelhante ao PHP_SELF.

```
<?php
                echo $_SERVER['SCRIPT_NAME'];
?>
```

script_name.php

Usando strings de consulta

Uma string de consulta é parte do URL que aparece depois de um ponto de interrogação. Por exemplo, o URL a seguir contém uma string de consulta:
http://integrator.com.br/buscar/?p=Hypertext+Preprocessor
Nesse exemplo, a string de consulta contém uma variável denominada **p** cujo valor é "**Hypertext Preprocessor**".
As strings de consulta são usadas para transmitir informações do navegador para o servidor. Normalmente, você não digita a string de consulta diretamente na barra de endereços do navegador. Ao contrário, cria um link em uma página que contém a string de consulta.
Veja um exemplo abaixo da utilização de strings de consulta onde são oferecidas opções para o cliente:

```
<HTML>
 <HEAD><TITLE>Maçãs</TITLE></HEAD>
 <BODY>
  Pro favor, escolha o tipo de maçã:
  <P><A HREF="string_cons_recebe.php?maca=vermelha">Vermelhas
deliciosas</A>
  <P><A HREF="recebe_string_cons.php?maca=verde">Verdes
maravilhosas</A>
 </BODY>
</HTML>
```

env_string_cons.php

No exemplo acima, vemos dois links de hipertexto para uma página denominada **recebe_string_cons.php**. O primeiro link transmite a string de consulta **maca** que contém o valor **vermelha**. A segunda string de consulta, também denominado **maca**, têm o valor de **verde**.
Dentro da página **recebe_string_cons.php**, você pode determinar o link escolhido pelo

cliente acessando o conjunto através de $**HTTP_GET_VARS**. Abaixo a página **recebe_string_cons.php**:

```
<HTML>
 <HEAD><TITLE>Sua maçã escolhida</TITLE></HEAD>
 <BODY>
  <?php
   $maca = $HTTP_GET_VARS["maca"];
   echo "Você selecionou a maçã $maca";
  ?>
 </BODY>
</HTML>
```

recebe_string_cons.php

Variáveis HTTP GET: $_GET

Contém um array associativo de variáveis passadas para o script atual através do método HTTP GET.
Esta é uma variável '**superglobal**', ou automaticamente global. Isto significa que ela é disponível em todos os escopos (níveis) de um script.
O mesmo exemplo acima poderia ser feito da seguinte maneira:

```
<HTML>
 <HEAD><TITLE>Sua maçã escolhida</TITLE></HEAD>
 <BODY>
  <?php
   $maca = $_GET["maca"];
   echo "Você selecionou a maçã $maca";
  ?>
 </BODY>
</HTML>
```

string_cons_recebe.php

Transmitindo diversas variáveis de string de consulta

Você pode transmitir diversas variáveis de consulta em uma única string. Para fazer isso, basta separá-las com o caractere **&**(“e” comercial). Veja o exemplo abaixo:

```
1. <HTML>
2. <HEAD><TITLE>Escola as frutas desejadas</TITLE></HEAD>
3. <BODY>
4. Pro favor, escolha o tipo de fruta:
5. <P><A HREF="rec_divs_str_cons.php?fruta=laranja&tipo=lima">
6. Laranja Lima</A>
7. <P><A HREF="rec_divs_str_cons.php?fruta=maca&tipo=vermelho">
8. Maçã Vermelha</A>
9. <P><A HREF="rec_divs_str_cons.php?fruta=maca&tipo=verde">
10. Maçã Verde</A>
11. </BODY>
12. </HTML>
```

env_divs_str_cons.php

Os links de hipertexto contêm as strings de consulta que incluem duas variáveis. Cada string de consulta contém tanto uma variável denominada **fruta**, quanto uma variável denominada **tipo**. Quando um usuário clica em um dos três links, ambas as variáveis de string de consulta são transmitidas para a página **divs_str_cons.php**.
Veja o exemplo a seguir recuperando os valores passados pela página anterior:

```
<HTML>
 <HEAD><TITLE>Frutas escolhidas</TITLE></HEAD>
 <BODY>
  Você selecionou:
  <P>Fruta: <?php echo $_GET["fruta"]?></P>
  <P>Tipo: <?php echo $_GET["tipo"]?></P>
 </BODY>
</HTML>
```

rec_divs_str_cons.php

Variáveis da requisição: $_REQUEST

A variável superglobal **$_REQUEST** é usada para trabalhar tanto com requisições via método POST, como com métodos via GET e COOKIE.

```
<HTML>
 <HEAD><TITLE>Frutas escolhidas</TITLE></HEAD>
 <BODY>
  Você selecionou:
       <P>Fruta: <?php echo $_REQUEST["fruta"]?></P>
       <P>Tipo: <?php echo $_REQUEST["tipo"]?></P>
</BODY>
</HTML>
```

rec_divs_str_request.php

Transmitindo caracteres especiais em uma string de consulta

Não é permitido incluir espaços ou outros caracteres especiais no nome ou valor de uma variável de string de consulta. Por exemplo, suponhamos que você quisesse transmitir o conjunto de caracteres "**Laranja Lima**" em uma variável de string de consulta. Você pode fazer isso como mostrado abaixo:

```
<HTML>
 <HEAD><TITLE>Transmitindo caracteres especiais</TITLE></HEAD>
 <BODY>
    <A HREF="receb_caract_esp.php?fruta=Laranja Lima">Laranja
Lima</A>
  </BODY>
</HTML>
```

trans_caract_esp.php

```
<HTML>
 <HEAD><TITLE>Transmitindo caracteres especiais</TITLE></HEAD>
```

```
 <BODY>
    <?php
       echo $_GET["fruta"];
    ?>
  </BODY>
</HTML>
```

receb_caract_esp.php

Esta página contém um link de hipertexto com uma variável de string de consulta denominada **fruta**. Entretanto, ao clicar sobre esse link, o valor da string de consulta não será transmitido corretamente para a página **receb_caract_esp.php** em navegadores antigos. O valor da string de consulta será **truncado** (retirado) no primeiro espaço.
Antes de transmitir uma string de consulta que contém caracteres especiais ou espaços, você deve codificar como URL na string de consulta. Quando a string é codificada, possíveis caracteres problemáticos serão substituídos.
No exemplo a seguir veja a transmissão sendo codificada para ser enviada:

```
<html>
  <head><title> Transmitindo caracteres especiais</title></head>
  <body>
    <?php
       $string=urlencode("Laranja Lima");
    ?>
    <A HREF="receb_caract_esp.php?fruta=<?php echo $string?>">
      Laranja Lima
    </A>
  </body>
</html>
```

trans_caract_esp_cod.php

Um outro exemplo clássico ocorre quando as strings serão passadas em uma consulta. O usuário pode querer digitar caracteres especiais que causam problemas quando são tratados de maneira comum na URL. Como é o caso do **&** "e comercial". Veja abaixo o exemplo sem e com a codificação:

```
<html>
<head><title> Transmitindo caracteres especiais</title></head>
<body>
<?php $string="Flambers&Noble Ltda."; ?>
<A HREF="receb_caract_esp_novo.php?empresa=
       <?php echo $string?>">A empresa do momento</A>
</body>
</html>
```

trans_caract_esp_novo.php

```
<HTML>
 <HEAD><TITLE>Transmitindo caracteres especiais</TITLE></HEAD>
 <BODY>
    <?php
       echo $_GET["empresa"];
    ?>
  </BODY>
```

```
</HTML>
```

receb_caract_esp_novo.php

No exemplo acima, você nota nitidamente que parte dos caracteres não foram recebidos. Isso demonstra que é importante o uso do método **urlencode( )**.

```
1. <html>
2. <head><title> Transmitindo caracteres especiais
   </title></head>
3. <body>
4. <?php
5. $string=urlencode("Flambers&Noble Ltda.");
6. ?>
7. <A HREF="receb_caract_esp_novo.php?empresa=
8.     <?php echo $string?>">A empresa do momento</A>
9. </body>
10.    </html>
```

trans_caract_esp_dec_novo.php

Array Associativo de recebimento de formulário

Para permitir que um cliente se registre em seu site da Web, preencha um formulário ou informe o número de seu cartão de crédito, é preciso usar um formulário HTML.
Para recuperar as informações que o cliente inclui no formulário HTML, você utiliza um conjunto de **Arrays Associativos**: ***$_POST***, ***$ _GET*** e ***$_FILES***, por exemplo.

O método mais comum de envio de formulários é o **POST**.

```
<html>
  <head>
    <title>Usando Array Associativo pelo método POST</title>
  </head>
  <body>
       <FORM METHOD="POST" ACTION="receb_met_post.php">
              Seu nome: <INPUT TYPE="TEXT" NAME="nome" /><BR />
              <INPUT TYPE="SUBMIT" VALUE="Enviar"/>
       </FORM>
  </body>
</html>
```

env_met_post.php

```
<HTML><HEAD>
<TITLE>Recebendo dados via formulário pelo método
POST</TITLE></HEAD>
 <BODY>
    <?php
       echo $HTTP_POST_VARS["nome"];
    ?>
  </BODY>
</HTML>
```

receb_met_post.php

Usando a superglobal **$_POST** você tem o mesmo efeito:

```
<HTML>
 <HEAD><TITLE>Recebendo dados via formulário pelo método
POST</TITLE></HEAD>
 <BODY>
    <?php
       echo $_POST ["nome"];
    ?>
  </BODY>
</HTML>
```

receb_met_post.php

Usando o método **GET** você tem a mesma funcionalidade dos envios criados por strings diretas nos links:

```
<html>
  <head><title>Usando Array Associativo pelo método
GET</title></head>
  <body>
       <FORM METHOD="GET" ACTION="receb_met_get.php">
              Seu nome: <INPUT TYPE="TEXT" NAME="nome" /><BR />
              <INPUT TYPE="SUBMIT" VALUE="Enviar"/>
       </FORM>
  </body>
</html>
```

env_met_get.php

```
<HTML>
 <HEAD><TITLE>Recebendo dados via formulário pelo método
GET</TITLE></HEAD>
 <BODY>
    <?php
       echo $_GET["nome"];
    ?>
  </BODY>
</HTML>
```

receb_met_get.php

Obs: As variáveis de arrays longas como **$HTTP_GET_VARS[ ]** ou **$HTTP_POST_VARS[ ]** são consideradas depreciadas e, portanto não são de uso recomendável, já que em versões posteriores podem vir desativadas por padrão e com o tempo desaparecer. A partir da versão do PHP 5.0.0 pode-se desativar com a diretiva **register_long_arrays=Off** no arquivo **PHP.INI**.

Problemas com migrações em versões inferiores ao PHP 4.2

Em versões anteriores ao PHP 4.2, existia uma forma de requisitar dados POST, GET e outros de forma global, sem necessidade de declaração de variáveis pré-definidas. Isso deixou de ser possível graças à diretiva **register_globals=Off** no **PHP.INI**, que, por motivos de segurança, passou a ficar desativada.

Cotas mágicas no PHP

A diretiva de configuração do PHP **magic_quotes_gpc** afeta os valores de GET, POST e Cookies. Se estiver ativada, o valor (Marca D'água "Versão 2006") se tornará automaticamente (Marca D\'água \"Versão 2006\"). **Escaping** é necessário para inserção em bancos de dados.
Abaixo um exemplo:

```
1. <html>
2. <head><title>Usando caracteres de escape</title></head>
3. <body>
4. <FORM METHOD="POST" ACTION="receb_met_post_esc.php">
5. Dados: <INPUT TYPE="TEXT" NAME="dados" VALUE= "Marca d' Água"
   /><BR />
6. <INPUT TYPE="SUBMIT" VALUE="Enviar"/>
7. </FORM>
8. </body>
9. </html>
```

env_met_post_esc.php

```
<HTML>
 <HEAD><TITLE>Usando caracteres de escape</TITLE></HEAD>
 <BODY>
    <?php
       echo $_POST ["dados"];
    ?>
  </BODY>
</HTML>
```

receb_met_post_esc.php

stripslashes( )

Ao receber informações com apóstrofo ou aspas, as cotas mágicas automaticamente colocam caracteres de escape. Para que estes caracteres de escape não apareçam, utilize o método **stripslashes( )**.
Abaixo o mesmo exemplo de recebimento só que com a utilização do método.

```
<HTML>
 <HEAD><TITLE>Usando caracteres de escape</TITLE></HEAD>
 <BODY>
    <?php
       echo stripslashes($_POST ["dados"]);
    ?>
  </BODY>
</HTML>
```

receb_met_post_esc.php

addslashes( )

Caso as informações estejam sendo enviadas para uma base de dados, é importante se previnir com o método **addslashes( )**. Assim, os caracteres de escape são forçados,

evitando erros possíveis causados pelas aspas e apóstrofos, principalmente quando as cotas mágicas não foram configuradas por você e sim, pelo administrador, onde é hospedado seu site PHP.

```
<HTML>
 <HEAD><TITLE>Usando caracteres de escape</TITLE></HEAD>
 <BODY>
    <?php
       $dados=addslashes($_POST ["dados"]);
       echo $dados;
    ?>
  </BODY>
</HTML>
```

receb_met_post_esc.php

Usando objetos de aplicação e sessão

A capacidade de identificar os clientes e personalizar conteúdo é importante porque pode ser usada para aumentar as vendas. Um exemplo simples: talvez você queira exibir anúncios distintos para clientes diferentes, de acordo com seus interesses. Se você registrou o fato de que um determinado cliente gosta de visitar as páginas de seu site da Web que mostram varas de pescar, pode mostrar automaticamente a esse cliente mais anúncios relacionados a material de pesca.

Usando cookies para monitorar os clientes

Os cookies receberam grande atenção da mídia nos últimos tempos pelo medo de que poderiam representar uma ameaça à privacidade das pessoas. Você pode usar um cookie para armazenar informações no computador de um cliente quando ele visitar seu site da Web.
Essas informações podem ser usadas para identificar o cliente quando ele retornar ao seu site.
Os cookies foram desenvolvidos pela Netscape para corrigir uma deficiência observada na interação entre servidores da Web e navegadores. Sem os cookies, a interação entre os servidores e navegadores sairiam do controle.

Curiosidade: De onde vem o termo "**cookie**"? Lou Montulli, que programou a especificação original do **cookie** para a Netscape, explica: "**Cookie** é um termo bastante conhecido em computação, utilizado quando se descreve uma parte opaca de dados mantida por um intermediário. O termo se aplica perfeitamente ao uso; simplesmente não é muito conhecido fora da área da computação."

Há dois tipos de cookies: cookies de sessão e cookies persistentes. Os cookies de sessão são armazenados na memória. Permanecem no computador do cliente somente enquanto ele está visitando o seu site da Web.
O cookie persistente, por outro lado, podem durar meses ou até anos. Os cookies persistentes são armazenados em um arquivo de texto no computador do cliente. Esse arquivo de texto é denominado arquivo Cookie nos computadores com sistema operacional Windows e arquivo Magic Cookie nos computadores Macintosh.

O Netscape Navigator e o Internet Explorer armazenam cookies persistentes de forma um pouco diferente. O Netscape armazena todos os cookies de todos os sites da Web em um arquivo denominado "Cookies.txt". Esse arquivo fica no diretório/Netscape ou /Netscape/User/Nomedousuario..
Já o Microsoft Internet Explorer cria um arquivo cookie independente para cada site da Web. Todos esses arquivos estão armazenados na pasta root:\Documents and Settings\Administrador\Cookies.
É importante entender que um site da Web só consegue ler os cookies que ele cria.
Também é importante compreender que nem todos os navegadores suportam cookies.
Jamais pressuponha que o navegador de um cliente suporta cookies. Um uso perfeitamente legítimo dos cookies é promover o login automático de um usuário ao seu site da Web. Entretanto, se você fizer isso, deve permitir que os usuários cujos computadores que não suportam cookies também façam o login.
O PHP suporta transparentemente cookies HTTP como os definidos pela especificação da Netscape. Cookies são um mecanismo de armazenamento de dados no browser cliente e permite o rastreamento ou identificação do retorno de usuários. Você pode criar cookies com a função **setcookie( )**. Cookies são parte do cabeçalho HTTP, então, a função setcookie( ) precisa ser chamada antes de qualquer saída ser enviada ao browser. Esta é a mesma restrição da função **header( )**. Quaisquer cookies enviados para você do cliente serão automaticamente transformados em variáveis PHP assim como os dados postados via GET ou POST.
A função tem o seguinte escopo:

```
int setcookie ( string nome [, string valor [, int expiração [,
string caminho [, string domínio [, int segurança]]]]])
```

Todos os argumentos, exceto o nome são opcionais.
A expiração é um argumento de tempo em inteiro regular do Unix, portanto somente poderá ser retornado pelas funções **time( )** ou **mktime( )**.

```
<?php
setcookie("MeuCookie", "Testando", time()+3600);
?>
<html>
<head>
<title>Exemplo de Cookie</title>
</head>
<body>
<h1>Seu cookie foi enviado com sucesso</h1>
</body>
</html>
```

env_cookie.php

Neste exemplo mostro claramente que o cookie deve ser enviado para o cliente antes de qualquer saída.
Este é um exemplo de cookie com erro:

```
<html>
<head>
<?php
setcookie("MeuCookie", "Testando", time()+3600);
```

```
?>
<title>Exemplo de Cookie</title>
</head>
<body>
<h1>Seu cookie foi enviado com sucesso</h1>
</body>
</html>
```

env_cookie_err.php

Este exemplo terá uma saída com o seguinte erro:

Warning: Cannot modify header information - headers already sent by (output started at **$PATH**\ env_cookie_err.php:3) **$PATH**\ env_cookie_err.php in on line 4

Se você precisa armazenar múltiplos valores em um único cookies, basta apenas acrescentar **[ ]** ao nome do cookie. Por exemplo:

setcookie("MeuCookie[ ]", "Testando", time( )+3600);

Note que um cookie substituirá um anterior com o mesmo nome em seu browser mesmo se o nome ou o caminho for diferente.
Para exibir o cookie criado:

```
<HTML>
 <HEAD><TITLE>Recebendo Cookie</TITLE></HEAD>
 <BODY>
  <?php
    echo $HTTP_COOKIE_VARS["MeuCookie"];
  ?>
 </BODY>
</HTML>
```

rec_cookie.php

Mas você também pode obter o valor do Cookie usando o método abreviado:

```
echo $_COOKIE["MeuCookie"];
```

Removendo um cookie criado

Para remover um cookie criado, você deve retroceder o tempo utilizado na criação do mesmo.

```
<?
setcookie ("MeuCookie", "", time() - 3600);
?>
```

remove_cookie.php

Usando variáveis de sessão para controlar clientes

Você pode usar variáveis **Session** como um método para controlar as informações sobre os clientes quando eles se movimentam de uma página para outra dentro do seu site da Web. As variáveis **Session** estão intimamente relacionadas aos cookies. Na verdade, as

variáveis **Session** baseiam-se nos cookies.
Você pode usar as variáveis Session para armazenar qualquer tipo de informação.
Para que a sessão funcione no **Sistema Operacional Windows**, você deve editar o caminho para que seja guardada corretamente no sistema. Para isso, vá até o **PHP.INI** e altere a seguinte informação:

PHP.INI no sistema operacional Windows

```
[Session]
;Neste local, por padrão, o arquivo está configurado para
;trabalhar no UNIX/LINUX.
;altere o path para o caminho do Windows.
session.save_path = C:\temp
```

```
<?php
       session_start();
       $_SESSION['nome']="Edson";
?>
<html>
  <head><title>Exemplo da utilização da Sessio<title></head>
  <body>
       <A HREF="session_recup.php">Próxima Página</A>
  </body>
</html>
```

session_inicio.php

```
<?
       session_start();
       if(!isset($_SESSION["nome"]))
              header("location:session_inicio.php");
?>

<HTML>
 <HEAD><TITLE>Recuperando uma Session</TITLE></HEAD>
 <BODY>
    <?php

         echo "Olá ".$_SESSION["nome"].", como vai?";

    ?>
       <BR /><A HREF="session_finalizar.php">Fechar a sessão</A>
  </BODY>
</HTML>
```

session_recup.php

No exemplo acima, na recuperação da Session, temos a construção da linguagem **isset( )** que é responsável pela verificação de existir ou não um valor na session. Se existir ele retorna **true**(verdadeiro), caso não aja nada, ele retornará **false**(falso).

```
<?php
       session_start();
       unset($_SESSION['nome']);
?>
```

```
<html>
       <head><title>Finalizando uma Sessão</title></head>
       <body>
         A variável <?php echo $_SESSION['nome']?>
         foi finalizada.
       </body>
</html>
```

session_finalizar.php

Para finalizar uma sessão, a construção da linguagem **unset( )** destrói a variável específica.

Nota: Tanto **isset( )** como **unset( )** não são apenas usadas em sessões.

Tipos

O PHP suporta os **oitos tipos primitivos**.

São quatros tipos básicos:
boolean
integer
float (número de ponto flutuante, ou também 'double')
string

Dois tipos compostos:
array
object

E finalmente dois tipos especiais:
resource
NULL

Você também pode encontrar algumas referências ao tipo **double**. Considere o tipo double como sendo o float, e os dois nomes existem por razões históricas.

```
<?php
     $i = 10;  // Inteiro
     $nome = "Edson";  // String
     $falso = FALSE;  // Booleano
     $valor = 100.50; /// Ponto flutuante
     $nulo = NULL;
     echo '$i é do Tipo ' . gettype($i) . '<br>';
     echo '$nome é do Tipo ' . gettype($nome) . '<br>';
     echo '$falso é do Tipo ' . gettype($falso) . '<br>';
     echo '$nulo é do Tipo ' . gettype($nulo) . '<br>';
     echo '$valor é do Tipo ' . gettype($valor);

?>
```

tipos_no_php5.php

Booleanos
Para especificar um literal booleano, use as palavras chave **TRUE** ou **FALSE**. Ambas são insensitivas ao caso.

Inteiros
Inteiros podem ser especificados em notação decimal (base 10), hexadecimal (base 16) ou octal (base 8), opcionalmente precedido de sinal (- ou +).
O tamanho de um inteiro é dependente de plataforma, sendo um numero aproximado a 2 bilhões o valor mais comum (número de 32 bits com sinal).

Overflow de inteiros
Se você especifica um número além dos limites do tipo inteiro, ele será interpretado como um ponto flutuante.

Números de pontos flutuantes
O tamanho de um número de ponto flutuante é dependente de plataforma, sendo o máximo com uma precisão de 14 decimais digitais.

Strings
Uma string é uma série de caracteres. No PHP, um caracter é o mesmo que um byte, ou seja, há exatamente 256 caracteres diferentes possíveis. Não há nenhum problema se as strings se tornarem muito grandes. Não há nenhum limite para o tamanho de strings imposta pelo PHP, então não há razão para se preocupar com strings longas.

Arrays
Um array no PHP é atualmente um mapa ordenado. Um mapa é um tipo que relaciona valores para chaves. Este tipo é otimizado de várias maneiras, então você pode usá-lo como um array real, ou uma lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente mais. Como você pode ter outro array PHP como um valor, você pode facilmente simular árvores. Arrays serão abordados mais adiante em matrizes.

Objetos
Mais adiante serão abordados objetos.

Resource
Recurso é uma variável especial, mantendo uma referência de recurso externo. Recursos são criados e utilizados por funções especiais. Mais adiante você aprenderá a usar **resource** com conexão a banco de dados MySQL (essa é uma das formas).

NULL
O valor especial NULL representa que a variável não tem valor.

Heredoc em Strings

Outra maneira para delimitar strings é utilizando a sintaxe heredoc ("<<<"). É informado um identificador depois de <<<, então a string, e então o mesmo identificador para fechar a delimitação.
O identificador de fechamento precisa começar na primeira coluna da linha. Além, o identificador utilizado precisa seguir as mesmas regras de nomeação que qualquer outro

rótulo no PHP: só pode conter caracteres alfanuméricos e sublinhados, e precisa começar com um caracter não numérico ou sublinhado.

```
<?php
$var1="Exemplo de string";
$var_string = <<< DOC
Exemplo da sintaxe heredoc, os caracteres escape
Funcionam, por exemplo, \\ (barra invertida) e \$,
além disso, funciona também:
\$var1  = $var1
DOC;
        echo "<pre>$var_string</pre>";
?>
```

heredoc.php

Interpretação de variáveis

Quando uma string é especificada dentro de aspas ou heredoc, variáveis são interpretadas dentro delas.
Há dois tipos de sintaxe: um simples e um complexo. A sintaxe simples é a mais comum e conveniente, provendo uma maneira de interpretar uma variável, um valor de array ou uma propriedade de object.
A sintaxe completa foi introduzida no PHP 4, e pode ser reconhecida por chaves ({}) envolvendo a expressão.

```
<?php
$cerveja = 'Heineken';
echo "O sabor das '$cerveja's é ótimo <br />";
// funciona,  " ' " é um caracter inválido para nome de variáveis
echo "Ele bebeu algumas $cervejas <br />";
// não funciona, 's' é um caracter válido para nome de variáveis
echo "Ele bebeu algumas ${cerveja}s <br />";  // funciona
?>
```

interp_vars.php

Uma outra utilidade desse tipo de interpretação está na impressão de variáveis como $_GET, $_POST e etc:
Veja o exemplo dessa utilização aplicada a um exemplo anterior:

```
<HTML>
 <HEAD><TITLE>Maçãs</TITLE></HEAD>
 <BODY>
  Pro favor, escolha o tipo de maçã:
  <P><A HREF="recebe_string_cons.php?maca=vermelha">
  Vermelhas deliciosas</A>
  <P><A HREF="recebe_string_cons.php?maca=verde">
  Verdes maravilhosas</A>
 </BODY>
</HTML>
```

env_string_cons.php

```
<HTML>
```

```
 <HEAD><TITLE>Sua maçã escolhida</TITLE></HEAD>
 <BODY>
  <?php
   echo "Você selecionou a maçã {$_GET['maca']}";
  ?>
 </BODY>
</HTML>
```

recebe_string_cons.php

Sem os colchetes haveria um erro:
Parse error: syntax error, unexpected T_ENCAPSED_AND_WHITESPACE, expecting T_STRING or T_VARIABLE or T_NUM_STRING in **$PATH\recebe_string_cons.php** on line **5**

Coerção de tipo

O PHP possui a conversão automática de tipo, chamada também de coerção de tipo automática. O PHP não requer (ou suporta) a definição de tipo explicita na declaração de variáveis: o tipo de uma variável é determinado pelo contexto em que a variável é utilizada. Isto significa que, se você assimila um valor string para a variável ***$var***, ***$var*** se torna uma string. Se você então assimila um valor inteiro para ***$var***, ela se torna um inteiro.

```
<?php
$var1 = "0";  // $var1 é string (ASCII 48)
echo $var1."<br />";
$var1 += 2;  // $var1 é agora um inteiro (2)
echo $var1."<br />";
$var1 = $var1 + 1.3;  // $var1 é agora um float (3.3)
echo $var1."<br />";
$var1 = 5 + "10 pequenos porcos";  // $var1 é inteiro (15)
echo $var1."<br />";
$var1 = 5 + "10 minúsculos porcos"; // $var1 é inteiro (15)
echo $var1."<br />";
?>
```

var_tipos.php

Type Casting – Coerção de tipo

A coerção de tipos no PHP funciona como no C: o nome de um tipo desejado é escrito entre parênteses antes da variável em que se deseja a coerção.

```
<?php echo (int)((0.1+0.7)*10.10)?><br /> <!--retorna o valor 8-->
<?php echo ((0.1+0.7)*10.10)?> <!--retorna o valor 8.08-->
```

type_casting.php

Nos dois casos, o resultado seria o mesmo, se não houvesse a coerção de tipo para **(int)**. O valor flutuante é truncado, restando apenas o valor inteiro.

As coerções permitidas são:

- **(int)**, **(integer)** - coerção para inteiro
- **(bool)**, **(boolean)** - coerção para booleano

- **(float)**, **(double)**, **(real)** - coerção para número de ponto flutuante
- **(string)** - coerção para string
- **(array)** - coerção para array
- **(object)** - coerção para objeto

Nota: Caso a sua intenção não seja fazer a coerção, mas arredondar o valor, use o método **round( )**.

Com o método **gettype( )** você consegue pegar o tipo. Usando o método **var_dump( )** você consegue receber o valor e também saber o seu tipo, mostrando assim informações sobre a variável.

```
<?php
        $numero = 2147483647;
        $numero2 = $numero * 10;
        echo gettype($numero2) . "<br>";
        $numero3 = 1000;
        var_dump($numero*$numero3);
?>
```

usando_var_dump.php

Constantes

Como já foi visto anteriormente, podemos alterar o valor de uma variável armazenada. Também podemos declarar constantes. Uma constante armazena um valor como uma variável, mas seu valor é configurado uma vez e, então, não pode mais ser alterado em outra parte do script. Por isso o nome **Constante**.

```
<?php
   define("EXEMPLO","Texto que acompanha a constante");
   define("EXEMPLO2",12); //valor que acompanha a constante
?>
<html>
<head>
<title>Constantes</title>
</head>
<body>
<?php echo EXEMPLO?><br>
<?php echo EXEMPLO2?>
</body>
</html>
```

constante.php

Você notará que os nomes da constante estão inteiramente em letras maiúsculas. Essa é uma convenção emprestada de C que facilita distinguir rapidamente entre variáveis e constantes.
Uma diferença importante entre constantes e variáveis é que quando você referencia a uma constante, ela não tem um sinal de cifrão na sua frente. Se quiser utilizar o valor de uma constante, utilize somente seu nome.
A regra para uma constante é simples:

- Constantes não podem ter um sinal de cifrão ($) antes delas;
- Constantes só podem ser definidas utilizando a função define( ), e não por simples assimilação;
- Constantes podem ser definidas e acessadas de qualquer lugar sem que as regras de escopo de variáveis sejam aplicadas;
- Constantes não podem ser redefinidas ou eliminadas depois que elas são criadas; e
- Constantes só podem conter valores escalares.

As constantes também podem ser configuradas, para que, sejam lidas sem ser case sensitive, passando um terceiro parâmetro, como mostra o exemplo a seguir em destaque:

```
<?php
  define("VALOR",10);
  define("FRUTA","Manga",True);
  echo "Fruta = " . fruta . "<br />"; // ou Fruta ou FRUTA , tanto faz
  echo "Valor = " . VALOR . "<br />"; // Ok
  echo "Valor = " . Valor . "<br />"; // Não irá funcionar, pois não há terceiro parâmetro
  define("VALOR",990); // Provocará um erro
?>
```

outros_ex_contantes.php

Operadores

Os operadores são símbolos que você pode utilizar para manipular valores e variáveis realizando uma operação neles.
Em geral, os operadores podem aceitar um, dois ou três argumentos, com a maioria aceitando dois. Por exemplo, o operador de atribuição aceita dois – a posição de memória no lado esquerdo do símbolo "=" e uma expressão no lado direito. Esses argumentos são chamados de operandos, isto é, aquilo que está sendo operado.

Operadores Aritméticos

Lembra-se da aritmética básica da escola? Estes operadores funcionam exatamente como aqueles.

Tabela dos Operadores Aritméticos		
exemplo	**nome**	**Resultado**
$a + $b	Adição	Soma de $a e $b.
$a - $b	Subtração	Diferença entre $a e $b.
$a * $b	Multiplicação	Produto de $a e $b.
$a / $b	Divisão	Quociente de $a por $b.
$a % $b	Módulo	Resto de $a dividido por $b.

Operadores de Atribuição

O operador básico de atribuição é "=". A sua primeira inclinação deve ser a de pensar nisto como "é igual". Não. Isto quer dizer, na verdade, que o operando da esquerda

recebe o valor da expressão da direita (ou seja, "é setado para").
O valor de uma expressão de atribuição é o valor atribuído. Ou seja, o valor de "$a = 3" é 3. Isto permite que você faça alguns truques:

```
$a = ($b = 4) + 5; // $a é igual a 9 agora
// e $b foi setado para 4.
```

Além do operador básico de atribuição, há "operadores combinados" para todos os operadores binários, aritméticos e de string que permitem a você usar um valor em uma expressão e então setar seu valor para o resultado daquela expressão. Por exemplo:

```
$a = 3;
$a += 5; // seta $a para 8, como se disséssemos: $a = $a + 5;
$b = "Olá ";
$b .= "você aí!"; // seta $b para "Olá você aí!"
```

Nota: O ponto aqui se destina à concatenação (união de strings).

Operadores de Comparação

Operadores de comparação, como os seus nomes implicam, permitem que você compare dois valores.

Tabela de Operadores de comparação		
exemplo	**nome**	**Resultado**
$a == $b	Igual	Verdadeiro se $a é igual a $b.
$a === $b	Idêntico	Verdadeiro se $a é igual a $b, e eles são do mesmo tipo.
$a != $b	Diferente	Verdadeiro se $a não é igual a $b.
$a < $b	Menor que	Verdadeiro se $a é estritamente menor que $b.
$a > $b	Maior que	Verdadeiro se $a é estritamente maior que $b.
$a <= $b	Menor ou igual	Verdadeiro se $a é menor ou igual a $b.
$a >= $b	Maior ou igual	Verdadeiro se $a é maior ou igual a $b.

Veja abaixo alguns exemplos de suas utilizações:

```
<?php
$a=5;
$b=5.0;
if($a==$b)
       echo "São iguais"; // Estes valores são iguais
?>
```

util_igualdade.php

```
<?php
```

```
$a=5;
$b=5.0;
if($a===$b)
	echo "São iguais";
else
	echo "São de tipos diferentes";
 //Neste são de tipos diferentes
?>
```

util_identico.php

O util_identico.php ilustra a utilização de idêntico "===" onde não só é comparado a igualdade como também o tipo de valor que cada variável carrega, isso é, se é *Inteiro* ou número de ponto *Flutuante*.

Neste caso ele irá imprimir na tela "**São de tipos diferentes**".

```
<?php
$a=5;
$b=7;
if($a!=$b)
	echo "São diferentes"; //São diferentes um do outro
?>
```

util_diferente.php

```
<?php
$a=5;
$b=3;
if($a>$b)
	echo 'A variável $a é maior do que $b';
//No caso uma variável tem valor maior que a outra
?>
```

util_maior.php

```
<?php
$a=5;
$b=7;
if($a<$b)
	echo 'A variável $a é menor do que $b';
//No caso uma variável tem valor menor que a outra
?>
```

util_menor.php

```
<?php
$a=5;
$b=3;
if($a>=$b)
	  echo 'A variável $a é maior ou igual a $b';
// Uma variável pode ter valor maior ou igual à outra
?>
```

util_maior_igual.php

```
<?php
```

```
$a=4;
$b=8;
if($a<=$b)
        echo 'A variável $a é menor ou igual a $b';
// Uma variável pode ter valor menor ou igual à outra
?>
```

util_menor_igual.php

Operadores de controle de erro

O PHP suporta um operador de controle de erro: o sinal 'em' (**@**). Quando ele precede uma expressão em PHP, qualquer mensagem de erro que possa ser gerada por aquela expressão será ignorada.

```
<html>
  <head><title>Operador de controle de erro</title></head>
  <body>
       <?php
              @session_start();
       ?>
  </body>
</html>
```

ocultando_erro.php

No exemplo acima, você tentou criar a abertura da sessão dentro de saídas de HTML. Isso não é possível, portanto gera um erro.
Com o arroba na frente, essa possibilidade fica nula, não gerando assim saída de erros.
Não é uma boa prática ocultar erros na criação de seus scripts. Isso porque você pode ocultar problemas que de certa forma ficam complexos de serem detectados devido essa ocultação.

Operadores de Incremento/Decremento

O PHP suporta operadores de pré e pós-incremento e decremento no estilo C.

Tabela de Operadores de Incremento/Decremento		
exemplo	**Nome**	**Efeito**
++$a	Pré-incremento	Incrementa $a de um, e então retorna $a.
$a++	Pós-incremento	Retorna $a, e então incrementa $a de um.
--$a	Pré-decremento	Decrementa $a de um, e então retorna $a.
$a--	Pós-decremento	Retorna $a, e então decrementa $a de um.

Aqui está um script de exemplo simples:

```
<HTML>
<HEAD><TITLE>Incremento e decremento</TITLE></HEAD>
<BODY>
<?php
echo "<h3>Pós-incremento</h3>";
$a = 5;
echo "Deve ser 5: " . $a++ . "<br>\n";
```

```
echo "Deve ser 6: " . $a . "<br>\n";

echo "<h3>Pré-incremento</h3>";
$a = 5;
echo "Deve ser 6: " . ++$a . "<br>\n";
echo "Deve ser 6: " . $a . "<br>\n";

echo "<h3>Pós-decremento</h3>";
$a = 5;
echo "Deve ser 5: " . $a-- . "<br>\n";
echo "Deve ser 4: " . $a . "<br>\n";

echo "<h3>Pré-decremento</h3>";
$a = 5;
echo "Deve ser 4: " . --$a . "<br>\n";
echo "Deve ser 4: " . $a . "<br>\n";
?>
</BODY>
</HTML>
```

incremento_decremento.php

Operadores Lógicos

Tabela de Operadores Lógicos		
exemplo	**nome**	**resultado**
$a and $b	E	Verdadeiro se tanto $a quanto $b são verdadeiros.
$a or $b	OU	Verdadeiro se $a ou $b são verdadeiros.
$a xor $b	XOR	Verdadeiro se $a ou $b são verdadeiros, mas não ambos.
! $a	NÃO	Verdadeiro se $a não é verdadeiro.
$a && $b	E	Verdadeiro se tanto $a quanto $b são verdadeiros.
$a || $b	OU	Verdadeiro se $a ou $b são verdadeiros.

A razão para as duas variantes dos operandos "**and**" e "**or**" é que eles operam com precedências diferentes.

Operadores de String

Há dois operadores de string. O primeiro é o operador de concatenação '**.**' (ponto), que retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação '**.=**' (ponto-igual).

Operadores de Arrays

Tabela de Operadores de Arrays		
Exemplo	**Nome**	**Resultado**
$a + $b	União	União de $a e $b.
$a == $b	Igualdade	TRUE se $a e $b tem os mesmos elementos.
$a === $b	Identidade	TRUE se $a e $b tem os mesmos elementos na mesma ordem.

Tabela de Operadores de Arrays		
Exemplo	**Nome**	**Resultado**
$a != $b	Desigualdade	TRUE se $a não é igual a $b.
$a <> $b	Desigualdade	TRUE se $a não é igual a $b.
$a !== $b	Não identidade	TRUE se $a não é idêntico a $b.

O operador + acrescenta o array da direita no array da esquerda, contudo, chaves duplicadas NÃO são sobrescritas. Veja Matrizes mais adiante.

Estruturas de Controle

Qualquer script PHP é construído por uma série de instruções. Uma instrução pode ser uma atribuição, uma chamada de função, um 'loop', uma instrução condicional, ou mesmo uma instrução que não faz nada(um comando vazio). Instruções geralmente terminam com um ponto e vírgula. Além disso, as instruções podem ser agrupadas em um grupo de comandos através do encapsulamento de um grupo de comandos com chaves. Um grupo de comandos é uma instrução também.

Estruturas Condicionais

if

A construção if é uma das mais importantes implementações de muitas linguagens, incluindo o PHP. Ela permite a execução condicional de fragmentos de código. O PHP implementa uma estrutura if que é similar àquela do C:

```
if (expressao)
    instruções
```

A condição expressao é avaliada por seu contexto Booleano. Se expressao for avaliado como **TRUE** (verdadeiro), o PHP executará instruções, e se for avaliado como **FALSE** (falso), ele será ignorado.
Os exemplos a seguir mostrariam que **a é maior que b** se **$a** for maior que **$b**:

```
if ($a > $b)
    echo "a é maior que b";
```

Freqüentemente você vai querer ter mais que uma instrução seja executada condicionalmente. E é claro, não há necessidade de englobar cada instrução com uma cláusula if. Em vez disso, você pode colocar várias instruções em um agrupamento de comandos. Por exemplo, este código mostraria a é maior que b se $a for maior que $b, e então atribuiria o valor de $a para $b:

```
if ($a > $b) {
    echo "a é maior que b";
    $b = $a;
}
```

Comandos if podem ser aninhados indefinidamente dentro de outros comandos if, o que faz com que você complete a flexibilidade para a execução condicional de várias partes do seu programa.

else

Freqüentemente você vai querer executar uma instrução se uma certa condição for encontrada, e uma instrução diferente se a condição não for encontrada. Isto é o que o else faz. else estende um comando if para executar uma instrução caso a expressão no comando if seja avaliada como **FALSE** (falso). Por exemplo, o código a seguir mostraria a é maior que b se $a for maior que $b, e a NÃO é maior que b caso contrário:

```
if ($a > $b) {
    echo "a é maior que b";
} else {
    echo "a NÃO é maior que b";
}
```

O comando else só é executado se a expressão **if** for avaliada como **FALSE**(falso), e se havendo qualquer expressão **elseif**, somente se todas elas forem avaliadas como **FALSE**(falso).

elseif

O comando elseif, como seu nome sugere, é uma combinação de if e else. Da mesma forma que o else, ele estende um comando if para executar uma instrução diferente no caso de a expressão if original ser avaliada como FALSE (falso). Porém, ao contrário de else, ele executará aquela expressão alternativa somente se a expressão condicional do elseif for avaliada como TRUE (verdadeiro). Por exemplo, o código a seguir mostraria a é maior que b, a é igual a b ou a é menor que b:

```
$a=5;
$b=7;
if ($a > $b) {
    echo "a é maior que b";
} elseif ($a == $b) {
    echo "a é igual a b";
} else {
    echo "a é menor que b";
}
```

Podem haver vários elseifs dentro da mesma instrução if. A primeira expressão elseif (se houver) que for avaliada como TRUE (verdadeiro) será executada. No PHP, você também pode escrever **'else if'** (em duas palavras) e o comportamento será *idêntico* a um **'elseif'** (em uma só palavra). O significado sintático é ligeiramente diferente (se você está familiarizado com C, eles tem o mesmo comportamento), mas no final de contas ambos teriam exatamente o mesmo comportamento.
O comando **elseif** só é executado se a expressão if precedente e quaisquer expressões elseif anteriores forem avaliadas como FALSE (falso), e a expressão elseif atual for avaliada como TRUE(verdadeiro).

Sintaxe alternativa para estruturas de controle

O PHP oferece uma sintaxe alternativa para algumas das suas estruturas de controle: if, while, for, foreach e switch. Em cada caso, a forma básica da sintaxe alternativa é mudar o sinal de abertura para **dois-pontos** (**:**) e o sinal de fechamento para **endif**;, **endwhile**;, **endfor**;, **endforeach**; ou **endswitch**;, respectivamente.

```
<?php if ($a == 5): ?>
A é igual a 5
<?php endif; ?>
```

No exemplo acima, o bloco HTML "**A é igual a 5**" está aninhado dentro de uma instrução if escrito na sintaxe alternativa. O bloco HTML será mostrado somente se **$a** é igual ao valor de **5**.
A sintaxe alternativa se aplica a **else** e **elseif** também. A seguir temos uma estrutura if com elseif e else no formato alternativo:

```
$a=7;
if ($a == 5):
    echo "a igual a 5";
elseif ($a == 6):
    echo "a igual a 6";
else:
    echo "a não é nem 5 nem 6";
endif;
```

Outra maneira de trabalhar com estruturas condicionais é utilizando o **switch**. Mais adiante você terá contato com a sintaxe.

Estruturas de Laços Condicionais (loop's)

while

Loops while são o tipo mais simples de criar um '**loop**' em PHP. Eles se comportam como seus compatíveis em C. O formato básico de um comando while é:

```
while (expressao)
instruções
```

O significado de um comando **while** é simples. Ele pede que o PHP execute os comandos aninhados repetidamente, enquanto a expressão do while é avaliada como **TRUE** (Verdadeiro). O valor da expressão é verificada cada vez que se passa no começo do *'loop'*, desta forma, mesmo que esse valor mude durante a execução do(s) comando(s) aninhado(s), a execução não parará até que o fim da iteração (cada vez que o PHP executa os comandos dentro do '**loop**' é uma **iteração**). Às vezes, se a expressão while é avaliada como **FALSE**(Falso) logo no início, o(s) comando(s) aninhado(s) não será(ão) rodado(s) nem uma vez sequer.
Como no comando if, você pode agrupar múltiplos comandos dentro do mesmo laço while englobando um grupo de instruções com chaves, ou usando a sintaxe alternativa:

```
while (expressao): instrucoes ... endwhile;
```

Os exemplos a seguir são idênticos, e ambos imprimem números de 1 to 10:

```
$i = 1;
while ($i <= 10) { //enquanto $i for menor ou igual a 10
    echo $i++;  /* o valor impresso será
                   $i depois do acréscimo
                   (pós-incremento) */
}
```

```
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
```

do...while

Loops do..while são bem similares aos loops while, exceto pelo fato de que a condição é verificada no fim de cada iteração em vez de no começo. A diferença principal dos loops while regulares é que a primeira iteração de um loop do..while é garantidamente executada (a condição só é verificada no fim da iteração) enquanto que ele pode não rodar necessariamente em um loop while normal (a condição é verificada no começo de cada iteração, se ela é avaliada como FALSE (falsa) logo no começo, a execução do loop terminaria imediatamente).
Há apenas uma sintaxe para loops do..while:

```
$i = 0;
do {
   echo $i;
} while ($i>0);
```

O **loop** acima rodaria exatamente uma vez, desde que depois da primeira iteração, quando a condição é verificada, ela é avaliada como **FALSE** (falsa) (**$i** não é maior que zero **0**) e a execução do loop termina.

for

Loops for são os laços mais complexos em PHP. Eles se comportam como os seus compatíveis em C. A sintaxe de um loop for é:

```
for (expressao1; expressao2; expressao3) instruções
```

A primeira expressão (expressao1) é avaliada (executada) uma vez incondicionalmente no começo do loop.
No começo de cada iteração, expressao2 é avaliada. Se ela é avaliada como **TRUE** (verdadeira), o loop continua e o(s) comando(s) aninhado(s) é (são) executado(s). Se é avaliada como **FALSE** (falsa), a execução do 'loop' termina.
No fim de cada iteração, expressao3 é avaliada (executada).
Cada uma das expressões pode ser vazia. expressao2 vazia significa que o loop pode rodar indefinidamente (PHP considera-a implicitamente como TRUE (verdadeira), como em C). Isto pode não ser tão inútil quanto você pode pensar, pois freqüentemente você pode querer terminar o 'loop' usando uma instrução condicional em vez de usar a

expressão-verdade do for.
Considere os seguintes exemplos. Todos eles mostram números de 1 a 10:

```
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
```

```
for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
```

```
$i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}
```

```
for ($i = 1; $i <= 10; echo $i, $i++);
```

Obviamente, o primeiro exemplo parece ser o mais bonito (ou talvez o quarto), mas você pode perceber que a possível utilização de expressões vazias em laços for se torna prático em algumas ocasiões.
O PHP também suporta a "**sintaxe de dois-pontos**" alternativa para laços for:

```
for (expressao1; expressao2; expressao3): instrucoes; ...;
endfor;
```

Existe ainda um outro tipo de loop que é dotado para trabalhar com coleções. Você verá isso mais adiante.

Um caso a parte – a estrutura do switch

switch

A instrução **switch** é similar a uma série de instruções **IF'**s seguidas. Em muitas ocasiões, você poderá ter que comparar a mesma variável (ou expressão) com muitos valores diferentes, executando códigos diferentes dependendo com qual valor ele se encaixar. É exatamente para isso que a instrução switch faz.
Os exemplos seguintes mostram duas maneiras diferentes de escrever a mesma coisa, uma utilizando uma série de ifs e a outra utilizando a instrução **switch**:

```
if ($i == 0) {
    echo "i igual a 0";
}
if ($i == 1) {
    echo "i igual a 1";
```

```
}
if ($i == 2) {
    echo "i igual a 2";
}
```

```
switch ($i) {
    case 0:
        echo "i igual a 0";
        break;
    case 1:
        echo "i igual a 1";
        break;
    case 2:
        echo "i igual a 2";
        break;
}
```

É importante entender como a instrução switch funciona para evitar enganos. A instrução switch executa linha a linha (atualmente, instrução a instrução). No início, nenhum código é executado. Somente quando uma instrução case é encontrada com um valor que combina com a expressão do switch faz com que o PHP execute as instruções a partir daí. O PHP continua executando as instruções até o fim do bloco switch ou na primeira vez que encontrar uma instrução break. Se você não escrever uma instrução break no fim das instruções case, o PHP continuará executando os *cases* seguintes. Exemplo:

```
switch ($i) {
    case 0:
        echo "i igual a 0";
    case 1:
        echo "i igual a 1";
    case 2:
        echo "i igual a 2";
}
```

Aqui, se **$i** é igual a zero, o PHP executará todas as instruções echo. Se **$i** é igual a **1**, o PHP executará as últimas duas instruções echo, e somente se **$i** for igual a **2**, você terá o comportamento 'esperado' apenas onde 'i igual a 2' será mostrado. Então é importante não se esquecer das instruções break (e às vezes não colocá-las para esse resultado em certas circunstâncias).
Em uma instrução switch, a condição somente será avaliada e resultado comparado para cada instrução case. Em uma instrução elseif, a condição é avaliada novamente. Se sua condição é mais complicada que uma simples comparação e/ou e dentro de um loop, um switch é mais rápido.
Um case pode não ter nenhuma instrução dentro, o que simplesmente passa o controle para o próximo case.

```
switch ($i) {
    case 0:
    case 1:
    case 2:
        echo "i é menor que 3 mas não negativo";
```

```
        break;
    case 3:
        echo "i é 3";
}
```

Um case especial é o default. Esse case é executado quando nenhum outro case combina. Ele precisa ser a última instrução case. Por exemplo:

```
switch ($i) {
    case 0:
        echo "i igual a 0";
        break;
    case 1:
        echo "i igual a 1";
        break;
    case 2:
        echo "i igual a 2";
        break;
    default:
        echo "i não é igual a 0, 1 ou 2";
}
```

A expressão avaliada pelo case precisa ser um tipo simples, ou seja, inteiros, números de ponto flutuante e strings. Arrays ou objetos não podem ser utilizados a não ser que eles impliquem num tipo simples.
A sintaxe alternativa para estruturas de controle é suportada para os switches.

Sintaxe alternativa para estruturas de controle

```
switch ($i):
    case 0:
        echo "i igual a 0";
        break;
    case 1:
        echo "i igual a 1";
        break;
    case 2:
        echo "i igual a 2";
        break;
    default:
        echo "i não é igual a 0, 1 ou 2";
endswitch;
```

break

break cancela a execução do comando for, foreach while, do..while ou switch atual.
break aceita um argumento numérico opcional que diz a ele quantas estruturas aninhadas englobadas devem ser quebradas.

continue

continue é usado dentro de estruturas de loops para saltar o resto da iteração do loop atual e continuar a execução no início da próxima iteração.

continue aceita um argumento numérico opcional que diz a ele de quantos níveis de loops aninhados ele deve saltar até o fim.

```
<?php
$i = 0;
while ($i++ < 5) {
    echo "Fora<br>\n";
    while (1) {
        echo "  Meio<br>\n";
        while (1) {
            echo "  Dentro<br>\n";
            continue 3;
        }
        echo "Isto nunca será exibido.<br>\n";
    }
    echo "Nem isso.<br>\n";
}
?>
```

ex_continue.php

return

Se chamada em uma função, a instrução **return( )** termina imediatamente a execução da função atual e retorna seu argumento como o valor da função.
return( ) também termina a execução de uma instrução **eval( )** ou de um script.
Se chamada no escopo global, a execução do script atual será terminada. Se o arquivo do script atual foi incluído com include( ), require( ) então a execução é devolvida para o arquivo chamador. Especificamente para arquivos de script incluídos com include( ), o valor fornecido para return( ) será devolvido como o valor da chamada include().

Nota: Note que **return( )** é um construtor de linguagem e não uma função, e parênteses em volta do seus argumentos não são necessários -- de fato, é mais comum não colocá-los que usá-los, sem, entretanto, haver diferença de um jeito ou de outro.

Criando bloco de códigos reutilizáveis

Quando estiver projetando seu site da Web, você apontará alguns elementos encontrados em muitas páginas em todo o site. Tais elementos podem ser barras de navegação ou uma linha de base com o endereço de e-mail do seu webmaster. Outros elementos podem ser fragmentos de códigos para exibir o dia, dados computados financeiros padrão e muitos outros códigos padronizados de HTML ou PHP, como por exemplo, constantes.

require( )

A instrução **require( )** e **include( )** são idênticos em todas as formas exceto pela manipulação de erros.
include( ) produz **Warning** enquanto **require( )** produzirá um **Fatal Error**. Em outras palavras, não hesite em utilizar **require( )** se na falta de um arquivo quiser parar o processamento da página.
include( ) não se comporta da mesma maneira, e o script poderá continuar nessa situação.

Veja os exemplos abaixo de sua utilização:

```
<?php

require 'teste.php';

require $arquivo;

require ('arquivo.txt');

?>
```

Nota: Até o PHP 4.0.2, havia o seguinte comportamento:
require( ) ocorre mesmo que a linha onde ele está nunca seja executada. É por isso que instruções condicionais não afetam **require( )**. Entretanto, se a linha onde ocorre o **require( )** não for executada, nada do código incluído do arquivo também será. Similarmente, estruturas de loop não afetam o funcionamento do **require( )**. Mas o código incluído pela função será *submetido* ao loop. A instrução **require( )** apenas ocorre uma vez.

include( )

A instrução **include( )** inclui e avalia o arquivo informado. Sua semelhança com o require dispensa maiores explicações.
Qualquer variável disponível da linha onde a chamada da inclusão ocorre estará disponível para o arquivo incluído, daquele ponto em diante.
Veja os exemplos de include( ):

```
<?php

$nome = 'Leonardo';
$apelido = 'Leo';

?>
```

includ.php

```
<?php

echo "O nome é $nome e seu apelido é $apelido";
 //As variáveis estão vazias

include 'includ.php';

echo " O nome é $nome e seu apelido é $apelido";
 // As variáveis neste caso contém as informações inclusas

?>
```

ex_include.php

Se o include ocorre dentro de uma função do arquivo principal, então todo o código incluído será executado como se ele tivesse sido definido dentro daquela função. Da

mesma forma, ele seguirá o escopo de variáveis da função.

```
<?php
function teste()
{
global $nome;
    include 'includ.php';
    echo " O nome é $nome e seu apelido é $apelido";
}
/* includ.php está no escopo da função teste( ),então $apelido
NÃO está disponível fora de seu escopo. $nome estará porque ela
foi declarada como global */

teste( );// Imprime o conteúdo da variável $nome e da variável
$apelido
echo "O nome é $nome e o apelido é $apelido"; //Imprime somente
$nome

?>
```

ex_include_2.php

Quando um arquivo é incluído, o interpretador sai do modo PHP e entra no modo HTML (no começo do arquivo incluído), e alterna novamente no seu fim. Por isso, qualquer código dentro do arquivo incluído que precisa ser executado como código PHP tem de ser delimitado por tags lidas de abertura e fechamento.
Se "**URL fopen wrappers**" estão ativas no PHP (normalmente na configuração padrão), você pode especificar um arquivo utilizando uma URL (via HTTP) em vez de um caminho local. Se o servidor apontado interpreta o arquivo informado como código PHP, variáveis podem ser passadas ao arquivo incluído na URL de requisição como num HTTP GET. Isto não é necessariamente a mesma coisa que incluir o arquivo e compartilhar o escopo de variável do arquivo principal: o script será executado no servidor remoto e apenas seu resultado será incluído no script local.

```
<?php

/* Este exemplo assume que www.exemplo.com está configurado para
interpretar arquivos .php mas não .txt. Além, 'Funciona' aqui
significa que as variáveis $teste e $teste2 estão disponíveis no
arquivo incluído*/

/*Não funciona: arquivos .txt não são manipulados em
www.exemplo.com como PHP    */
include 'http://www.exemplo.com/arquivo.txt?teste=1&teste2=2';

/* Não funciona: procura por um arquivo chamado
arquivo.php?teste=1&teste2=2' no  sistemas de arquivo local. */
include 'arquivo.php?teste=1&teste2=2';

// Funciona.
include 'http://www.exemplo.com/arquivo.php?teste=1&teste2=2';

$teste = 1;
$teste2 = 2;
```

```
include 'arquivo.txt';  // Funciona.
include 'arquivo.php';  // Funciona.

?>
```

Por serem **include( )** e **require( )** dois *construtores de linguagem especiais*, você precisa delimitá-los como um bloco de instruções quando utilizados dentro de instruções condicionais.

```
<?php

// Isto está errado e não funcionará como desejado
if ($condicao)
    include $arquivo;
else
    include $outro;


// E este está correto
if ($condicao) {
    include $arquivo;
} else {
    include $outro;
}

?>
```

Também é possível executar uma instrução **return( )** dentro de um arquivo incluído de maneira a finalizar o processamento daquele arquivo e retornar para o script que o chamou. Também é possível retornar valores de arquivos incluídos. Você pode pegar o valor de retorno de um include como faria com uma função normal.
return( ) se aplica apenas para a função e não para todo o arquivo.

```
<?php
$var = 'PHP';
return $var;
?>
ex_return_inc.php
<?php
$var = 'PHP';
?>
```

ex_return_inc_2.php

```
<?php
$teste = include ' ex_return_inc.php';

echo $teste; // imprime 'PHP'

$teste2 = include ' ex_return_inc_2.php';

echo $teste2; // imprime 1

?>
```

ex_return_inc_3.php

$teste2 assimila o valor **1** porque a inclusão foi realizada com sucesso. Verifique a diferença entre os exemplos. O primeiro utiliza **return( )**
dentro do arquivo incluído enquanto que o outro não. Há outras maneiras de "incluir" arquivos dentro de variáveis, com fopen( ), file( ) ou utilizando include( ) através das **Funções de Controle de Saída**.

require_once( )

A instrução require_once( ) avalia o arquivo especificado durante a execução do script. Seu comportamento é similar ao da instrução require( ), a não ser que o arquivo informado já tenha sido incluído, não refazendo a operação novamente.

include_once( )

A instrução include_once( ) inclui e avalia o arquivo especificado durante a execução de um script. Seu comportamento é similar à instrução include( ), a não ser que o arquivo informado já tenha sido incluído, não refazendo a operação novamente. Como o nome sugere, ele será incluído apenas uma vez.

Funções

Funções definidas pelo usuário

Uma função pode ser definida usando-se a sintaxe como a seguinte:

```
function teste($argumento1, $argumento2, ..., $argumentoN) {
    echo "Função de exemplo.<Br>\n";
    return $retonaValor;
}
```

Qualquer código PHP válido aparece dentro de uma função, mesmo outras funções e definições de classes.
O PHP não suporta sobrecarga de funções, e também não é possível cancelar ou alterar a definição de funções previamente declaradas.

Argumentos de funções

Informação pode ser passada para funções através da lista de argumentos, que é uma lista de variáveis e/ou constantes delimitados por vírgulas.
O PHP suporta a passagem de argumentos por valor (o padrão), passagem por referência valores padrão de argumento.

Valores padrão de argumentos

Uma função pode definir valores padrão no estilo C++ para argumentos escalares, como a seguir:

```
<?php
function argumento ($tipo = "exemplo") {
    return "Este é somente um $tipo.<br>\n";
}
```

```
echo argumento ( );// saída: Este é somente um exemplo.
echo argumento ("outro exemplo");
// saída: Este é somente um outro exemplo.
?>
```

func_val_padrao.php

O valor padrão precisa ser uma expressão constante, não (por exemplo) uma variável ou um membro de classe.
Note que usando argumentos padrão, qualquer padrão dever estar do lado direito de argumentos não-padrão; caso contrário, as coisas não funcionarão como esperado.

Considere o seguinte trecho de código:

```
<?php
function arg_errado ($tipo = "exemplo", $argumento2) {
    return "Este é um $tipo de $argumento2.\n";
}

echo arg_errado ("texto");   // saída: Irá gerar um erro
?>
```

func_arg_err.php

A saída do exemplo acima terá um erro como o mostrado abaixo:

Warning: Missing argument 2 for **arg_errado( )** in /var/www/aulaphp/func_arg_err.php on line **9**
Este é um texto de .

Agora, compare o que está acima com este:

```
<?php
function arg_certo ( $argumento2,$tipo = "exemplo") {
    return "Este é um $tipo de $argumento2.\n";
}

echo arg_certo ("texto");   // saída: Irá ter a saída correta
?>
```

func_arg_certo.php

A saída deste exemplo é: **Este é um exemplo de texto**.

Funções definidas condicionalmente

Aqui você percebe que podemos ter funções chamadas antes de serem construídas e condições para chamar uma função.

```
<?php

$chamar = true;

/* Nos nao podemos chamar exemplo( ) daqui
```

```
  porque ela ainda não existe,
  mas nos podemos chamar exemplo2( ) */

exemplo2( );

if ($chamar) {
 function exemplo ( )
 {
  echo "Eu não existo até que o programa passe por aqui.<br />";
 }
}

/* Agora nos podemos chamar exemplo( )
  porque $chamar foi avaliado como true */

if ($chamar) exemplo( );

function exemplo2( )
{
 echo "Eu existo imediatamente desde o início.<br />";
}

?>
```

func_def_cond.php

Funções dentro de funções

```
<?php
function exemplo( )
{
 function exemplo2()
 {
  echo "Eu não existo até exemplo( ) ser chamado.<br />";
 }
}

/* Nós não podemos chamar exemplo2( ) ainda
  porque essa função ainda não foi definida. */

exemplo( );

/* Agora nós podemos chamar exemplo2( ),
  porque o processamento de exemplo( )
  tornou a primeira acessível */

exemplo2( );

?>
```

func_dentro_func.php

Funções com argumentos variáveis

Imagine a situação:
Seu chefe lhe pediu para criar uma função, de somar valores, mas infelizmente ele lhe disse que não existe uma quantidade precisa para transmitir os valores a serem somados. Isso realmente é um problema, pois você precisa de uma matriz para que a função saiba quantos itens serão transmitidos. Alguma semelhança com as funções do Excel ou Calc?

```
<?php
function somar( ) {
  $numargs = func_num_args( );
  echo "Número de argumentos passados: $numargs<br />\n";
  if ($numargs >= 2) {
  echo "Selecionando apenas um argumento em especial: " . func_get_arg (1) . "<br />\n";
  }
  $arg_list = func_get_args( );
  for ($i = 0; $i < $numargs; $i++) {
                $total+=$arg_list[$i];
  }
   echo "O resultado é: " . $total;

}

somar (5, 2, 3, 5, 3);
?>
```

args_variaveis.php

O método **func_num_args( )** retorna o número de argumentos total. O método **func_get_arg( )** pega o valor de um argumento em especial.
Já o método estrela da função **somar( )** criada é o **func_get_args( )**, capaz de reunir os valores em uma matriz, podendo assim, você passar uma quantidade "ilimitada" de argumentos, sem se preocupar em declará-los ou não.

Retornando valores

Valores são retornados pelo uso de comandos opcionais de retorno. Qualquer tipo pode ser retornado, incluindo listas e objetos.

```
<?php
function teste ($numero) {
    return $numero+$numero*2;
}
echo teste (7);   // imprime '21'.
?>
```

func_return_val.php

Escopo de variáveis

O escopo de uma variável é o contexto onde ela foi definida. A maior parte das variáveis do PHP tem somente escopo local. Este escopo local inclui os arquivos

incluídos. Por exemplo:

```
$a = 1;
include "b.inc";
```

Aqui a variável **$a** estará disponível no script incluído **b.inc**. Entretanto, com as funções definidas pelo usuário, um escopo local é introduzido. Quaisquer variáveis utilizadas dento da função é por padrão limitada dentro do escopo local da função. Por exemplo:

```
<?php
$a = 1; /* escopo global */

function Teste()
{
    echo $a; /* referencia uma variável do escopo local (não
definida) */
}

Teste();
?>
```

ex_esc_err.php

Este script não produz nenhuma saída porque a instrução **echo( )** refere-se a uma versão local da variável **$a**, e ela não tem nenhum valor assimilado nesse escopo. Essa é uma pequena diferença da linguagem C quando variáveis globais são automaticamente disponíveis para funções sem sobrescrever uma eventual definição local. Isto causa problemas quando as pessoas mudam inadvertidamente uma variável global. No PHP, as variáveis globais precisam ser declaradas globais dentro de uma função se ela vai ser utilizada naquela função. Um exemplo:

```
<?php
$a = 1;
$b = 2;

function Soma()
{
    global $a, $b,$c;

    $c = $a + $b;
}

Soma();
echo $c;
?>
```

ex_esc_global.php

O script acima imprimirá "**3**". Declarando **$a** e **$b** globais na função, todas as referências a essas variáveis referem-se à versão global. Não há um limite para o número de variáveis globais que podem ser manipuladas por uma função.
Uma segunda maneira de acessar variáveis do escopo global é utilizando o array especial **$GLOBALS** definido pelo PHP. O exemplo anterior poderia ser rescrito como:

```
<?php
$a = 1;
$b = 2;

function Soma()
{
    $GLOBALS["c"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Soma();
echo $c;
?>
```

ex_arr_globals.php

$GLOBALS é um array associativo em que o nome das variáveis globais são suas chaves e o conteúdo dessas variáveis são os valores do elemento do array.
Outro recurso importante do escopo de variáveis é a variável estática. Uma variável estática existe somente no escopo local da função, mas ela não perde seu valor quando o nível de execução do programa deixa o escopo. Considere o seguinte exemplo:

Variáveis Estáticas

```
<?php
function Teste ()
{
    $a = 0;
    echo $a;
    $a++;
}
for($i=0;$i<10;$i++)
    Teste();
?>
```

Essa função é inútil partindo de que cada vez que ela é chamada, ela coloca em **$a** o valor **0** e imprime "**0**". A instrução **$a++** , que aumenta o valor da variável não tem sentido desde que a função sai e a variável $a desaparece. Para fazê-la mais útil como contadora sem deixar de perder o sua conta atual, a variável $a é declarada como estática:

```
<?php
function Teste()
{
    static $a = 0;
    echo $a;
    $a++;
}
for($i=0;$i<10;$i++)
    Teste();
?>
```

Agora, cada vez que a função **Teste( )** for chamada ele imprimirá o valor de **$a** e o incrementará.

Variáveis estáticas fornecem uma solução ideal para funções recursivas. Uma função recursiva é aquela que chama a si mesma. Cuidados especiais precisam ser tomados quando escrevendo funções recursivas porque é possível que ela continue na recursão indefinidamente. Você tem de ter certeza que há uma maneira segura de terminar a recursão. A seguinte função recursiva conta até 10, utilizando a variável estática $contar para saber quando parar:

```
<?php
function Teste()
{
    static $contar = 0;

    $contar++;
    echo $contar;
    if ($contar < 10) {
        Teste();
    }
    $contar--;
}
Teste();
?>
```

ex_var_statics.php

Uma outra maneira de utilizar variáveis estáticas é dentro de instruções de loops.

```
<?php
       while($i<10){
              static $i=0;
              $i++;
              echo $i."<br />";
       }
?>
```

ex_var_statics_2.php

Seria impossível ter a variável dentro do bloco e esperar a finalização do loop, sabendo que esta variável sempre voltará a zero, tornando-o infinito.

Enviando arquivos para o servidor

Algumas vezes você precisa enviar arquivos ao servidor e para fazer isso, você precisará de um formulário preparado para essa situação e o código PHP para receber os dados:

```
<html>
<head>
<title>Enviando Arquivos para o Servidor</title>
</head>
<body>
<form action="recebe_arq.php" method="post" enctype="multipart/form-data"
      name="form1" id="form1">
  <input name="arquivo" type="file" id="arquivo" />
  <input name="bt_env" type="button" value="Enviar" />
</form>
```

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

form_env_arqs.php

Uma situação importante a se observar na criação do formulário é que você deve colocar o atributo **enctype="multipart/form-data"**. Se você não fizer isso, o formulário não enviará arquivos para a página PHP que será responsável por recebê-lo.

```
<?php
  // o tamanho máximo necessário
  $tamanho_maximo = 1024*100; // 100 KB
  $tipos_aceitos = array("image/gif",
                                        "image/pjpeg",
                                        "image/x-png",
                                        "image/bmp",
                                        "image/jpg");
  // validar o arquivo enviado
  $arquivo = $_FILES['arquivo'];
  if($arquivo['error'] != 0) {
      echo '<p><b>Erro no Upload do arquivo<br>';
      switch($arquivo['erro']) {
      case  UPLOAD_ERR_INI_SIZE:
                          echo 'O Arquivo excede o tamanho máximo permitido';
                          break;
      case UPLOAD_ERR_FORM_SIZE:
          echo 'O Arquivo enviado é muito grande';
          break;
      case  UPLOAD_ERR_PARTIAL:
          echo 'O upload não foi completo';
          break;
      case UPLOAD_ERR_NO_FILE:
          echo 'Nenhum arquivo foi informado para upload';
          break;
      }
      echo        '</b></p>';
      exit;
  }
  if($arquivo['size']==0 || $arquivo['tmp_name']==NULL) {
      echo '<p><b>Envie um arquivo</b></p>';
      exit;
  }
  if($arquivo['size']>$tamanho_maximo) {
      echo '<p><b>O Arquivo enviado é maior que o limite:  ' .
round($tamanho_maximo/1024) .'KB</b></p>';
      exit;
  }
  if(array_search($arquivo['type'],$tipos_aceitos)===FALSE) {
      echo '<p><b>O Arquivo enviado não é do tipo (' .
          $arquivo['type'] . ') aceito para upload.
```

```
        Os Tipos Aceitos São:</b></p>';
    echo '<pre>';
    print_r($tipos_aceitos);
    echo '</pre>';
    exit;
  }
  // copiando o arquivo enviado

  $destino = realpath('arquivos') ."\\".$arquivo['name'];
  if(move_uploaded_file($arquivo['tmp_name'],$destino)) {
    // Tudo Ok, mostramos os dados
    echo  '<p><b>';
    echo 'O Arquivo foi recebido com sucesso!</b></p>';
    echo "<img src='$destino' border=0>";
  }
  else {
    echo '<p><b>Ocorreu um erro durante o upload</b></p>';
  }
?>
```

recebe_arq.php

O tamanho máximo do arquivo a ser feito UPLOAD é colocado em destaque no início do código.
O array **$tipos_aceitos** traz os tipos **MIME** aceitos pelo sistema de UPLOAD que você construiu.
Os conteúdos de **$_FILES** do script de exemplo é atribuído a **$arquivo**. Note que isso assume que o nome do upload do arquivo é **arquivo**, como o usado no exemplo anterior. Pode ser qualquer nome.
A seguir você tem o significado de cada item do array $_FILES:

$_FILES['arquivo']**['name']** - O nome original do arquivo no computador do usuário.

$_FILES['arquivo']**['type']** - O tipo MIME do arquivo, se o browser deu esta informação. Um exemplo pode ser "**image/gif**".

$_FILES['arquivo']**['size']** - O tamanho, em bytes, do arquivo.

$_FILES['arquivo']**['tmp_name']** - O nome temporário do arquivo, como foi guardado no servidor.

$_FILES['arquivo']**['error']** – Retorna os erros encontrados ao fazer o UPLOAD do arquivo.

Os códigos de erro de envio de arquivos

Os códigos de erro são explicados a seguir:

UPLOAD_ERR_OK
Valor: **0**: não houve erro, o upload foi bem sucedido.

UPLOAD_ERR_INI_SIZE
Valor **1**: O arquivo no upload é maior do que o limite definido em **upload_max_filesize** no **php.ini**.

UPLOAD_ERR_FORM_SIZE
Valor: **2**: O arquivo ultrapassa o limite de tamanho em **MAX_FILE_SIZE** que foi especificado no formulário HTML.

UPLOAD_ERR_PARTIAL
Valor: **3**: o upload do arquivo foi feito parcialmente.

UPLOAD_ERR_NO_FILE
Valor: **4**: Não foi feito o upload do arquivo.

Como fazer UPLOAD de múltiplos arquivos

```
<html>
<head>
<title>Enviando Múltiplos Arquivos para o Servidor</title>
</head>

<body>
<?php
$arquivo = $_FILES['arquivo'];

for($i=0;$i<count($arquivo);$i++)
        echo "{$arquivo['name'][$i]}<br />";
?>
<form action="" method="post"
enctype="multipart/form-data" name="form1" id="form1">
  <input name="arquivo[ ]" type="file" />
  <br />
  <input name="arquivo[ ]" type="file" />
  <br />
  <input name="arquivo[ ]" type="file" />
  <br />
  <input name="arquivo[ ]" type="file" />
  <br />
  <br />
  <input name="bt_env" type="submit" value="Enviar" />
</form>
</body>
</html>
```

form_env_mult_arqs.php

Orientação a Objetos com PHP 5

Classe

Classe é o termo técnico utilizado em linguagens orientadas a objetos que descreve um

conjunto de dados estruturados que são caracterizados por propriedades comuns. Também pode ser interpretado como uma estrutura modular completa que descreve as propriedades estáticas e dinâmicas dos elementos manipulados pelo programa. Podem-se definir classes de objetos como a descrição de um grupo de objetos por meio de um conjunto uniforme de atributos e serviços. Uma classe é um conjunto de objetos que compartilham as mesmas operações.

Enquanto um objeto individual é uma entidade concreta que executa algum papel no sistema como um todo, uma classe captura a estrutura e o comportamento comum a todos os objetos que são relacionados. Um objeto possui uma identidade e suas características serão definidas para a classe.

Uma classe é definida por:

- um nome da classe;
- o nome da sua superclasse;
- o nome de suas variáveis;
- os nomes e as definições de todas as operações associadas a esta classe;

O nome da classe

Toda definição de classe começa com a palavra-chave class, seguido por um nome da classe, que pode ser qualquer nome que não seja uma palavra reservada no PHP.

A seguir você tem a definição de uma classe simples:

```
<?php
class Classe {
  /* declaração de membro */
  public $var = 'um valor padrão de acesso público';

  /* declaração de método público mostrar*/
  public function mostrar( ) {
    echo $this->var;
  }
}
?>
```

class_simp.php

new

Para criar uma instância de um objeto, um novo objeto deve ser criado e atribuído a uma variável. Um objeto sempre será atribuído quando um novo objeto for criado a não ser que o objeto tenha um construtor definido que dispara uma exceção por um erro.

```
<?php
        require("class_simp.php");
?>
<html>
<head>
<title>Utilizando uma classe simples</title>
</head>
<body>
<?php
$instancia = new Classe( );
$instancia->mostrar( );
```

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

util_class_simp.php

A classe Caixa

A seguir, você tem mais um exemplo. A idéia é fixar a utilização de orientação a objetos com PHP5.
A classe **Caixa** teria como atributos características como as dimensões, cor, conteúdo e coisas semelhantes. As funções ou métodos que poderíamos incorporar a classe "Caixa" são as funcionalidades que queremos que a caixa realize, por exemplo: introduz( ), mostra_conteudo( ), comprova_se_cabe( ), esvaziar( )...

```
<?php
class Caixa{
  private $conteudo;

public function introduz($coisa){

  $this->conteudo = $coisa;

}

public function mostra_conteudo( ){

  echo $this->conteudo;

}
}

?>
```

class_caixa.php

Neste exemplo criou-se uma classe de nome Caixa, indicando como atributo o conteúdo. Para começar, um par de métodos foram criados, um para introduzir um elemento na caixa e outro para mostrar o conteúdo.
Se repararmos, os atributos definem-se declarando umas variáveis ao principio da classe. Os métodos definem-se declarando funções dentro de uma classe. A variável **$this**, utilizada dentro dos métodos será explicado mais adiante.

Atributos

Um atributo é um dado para o qual cada objeto tem seu próprio valor. Atributos são, basicamente, a estrutura de dados que vai representar a classe.

Exemplo de atributos, usando a classe fila:

private $fila;

```
private  $primeiro;

private $ultimo;
```

Métodos

Métodos são declarados dentro de uma classe para representar as operações que os objetos pertencentes a esta classe podem executar.
Um método é a implementação de uma rotina, ou seja, o código propriamente dito. Pode ser comparado a um procedimento ou função das linguagens imperativas.
Exemplo de métodos, utilizando a classe fila:

```
public function iniciar ( ) {
        $this->primeiro = 0;
        $this->ultimo = 0;
}
```

Como utilizar a classe Caixa

As classes são definições. Se quisermos utilizar uma classe temos de criar uma cópia dessa classe, o que normalmente se chama instanciar um objetos de uma classe.

```
$minhacaixa = new Caixa( );
```

Com isto instanciamos um objeto da classe Caixa chamado $minhacaixa:

```
$minhacaixa->introduz("algo");
$minhacaixa->mostra_conteudo( );
```

Com estas duas sentenças estamos inserindo a string "algo" na caixa e depois estamos mostrando esse conteúdo no texto da página. Repare que os métodos que o objeto chama utiliza o código "->".

```
nome_do_objeto->nome_do_metodo( )
```

Para aceder aos atributos de uma classe também se acede com o código "->". Desta forma:

```
nome_do_objeto->nome_do_atributo
```

A variável $this

Dentro de um método, a variável **$this** faz referência ao objeto sobre o qual invocamos o método. Na invocação $minhacaixa->introduz("algo") você está chamando o método introduzido sobre o objeto $minhacaixa. Quando executar esse método, passa-se o valor que recebe por parâmetro ao atributo contido. Neste caso **$this->conteudo** faz referência ao atributo contido no objeto **$minhacaixa**, que é sobre o que se invocava no método.

Visibilidade

A visibilidade de uma propriedade ou método pode ser definida prefixando a declaração com as palavras-chave: **public**, **protected** ou **private**. Itens declarados como **public** podem ser acessados por todo mundo. **Protected** limita o acesso a classes herdadas (e para a classe que define o item). **Private** limita a visibilidade para apenas a classe que define o item.
Para que você entenda o que isso significa, vamos fazer um exemplo:

```
<?php
        require("class_caixa.php");
?>
<html>
<head>
<title>Utilizando a classe Caixa</title>
</head>

<body>
<?php
        $minhacaixa = new Caixa( );
        $minhacaixa->introduz("algo");
        $minhacaixa->mostra_conteudo( );
?>
</body>
</html>
```

util_class_caixa.php

O exemplo mostrado acima demonstra a utilização da classe Caixa como se encontra. Modificaremos essa chamada, tentando acessar o atributo **$conteudo**:

```
<?php
        $minhacaixa = new Caixa( );
        $minhacaixa->introduz("algo");
        $minhacaixa->conteudo;

?>
```

util_class_caixa.php

Ao fazer isso, você terá um erro como o mostrado a seguir:

Fatal error: Cannot access private property Caixa::$conteudo in **$PATH\util_class_caixa.php** on line **15**

Esse erro indica que você está tentando acessar um membro **private**.

Agora vamos modificar a classe Caixa, tornando o membro **$conteudo public**:

```
<?php
class Caixa{
  public $conteudo;
...
```

class_caixa.php

Ao fazer isso, você notará que acessar esse membro não causará erro. Se você desejar ecoar o valor do membro **$conteudo**, você terá que alterar a página

util_class_caixa.php:

```
<?php
        $minhacaixa = new Caixa( );
        $minhacaixa->introduz("algo");
        echo $minhacaixa->conteudo;
?>
```

Objeto

O que caracteriza a programação orientada a objetos são os objetos. De um modo geral podemos encarar os objetos como sendo os objetos físicos do mundo real, tais como: carro, avião, casa, telefone, computador, etc., por isso que às vezes é dito que orientação a objetos representa os problemas mais próximos ao mundo real, dando assim mais facilidade a programação como um todo, mais isso não é sempre verdade, porque às vezes temos problemas que são extremamente funcionais. Nesses problemas funcionais é difícil representar a estrutura lógica em torno de objetos. Com isso, não são todos os problemas que giram em torno dos objetos facilmente visíveis.

De maneira simples, um objeto é uma entidade lógica que contém dados e código para manipular esses dados. Os dados são denominados como sendo atributos do objeto, ou seja, a estrutura que o objeto tem, e o código que o manipula denominamos método. Um método é uma função que manipula a estrutura de dados do objeto.

Construtores em PHP

Os construtores são funções, ou métodos, que se encarregam de realizar as tarefas de iniciação dos objetos ao serem instanciados. Isto é, quando se criam os objetos a partir das classes, chama-se um construtor que se encarrega de iniciar os atributos do objeto e realizar qualquer outra tarefa de inicialização que seja necessária.

Não é obrigatório dispor de um construtor, mas são muito úteis e a sua utilização é muito habitual. No exemplo da caixa, o normal seria iniciar as variáveis como cor ou as relacionadas com as dimensões e também indicar que o conteúdo da caixa está vazio. Se não há um construtor não se iniciam nenhum dos atributos dos objetos.

O construtor se define dentro da própria classe, como se fosse outro método. O único detalhe é que o construtor deve ser chamado como **__construct** .

Para a classe Caixa definida anteriormente, poderia declarar-se este construtor, como mostrado em destaque a seguir:

```
<?php
class Caixa{
  private $altura;
  private $espessura;
  private $largura;
  private $conteudo;
  private $cor;

        function __construct($altura=1,$espessura =1,$largura=1,$cor="preto"){

          $this->altura=$altura;
          $this->espessura=$espessura;
          $this->largura=$largura;
          $this->cor=$cor;
```

```
        $this->conteudo="";
    }

    public function introduz($coisa){

        $this->conteudo = $coisa;

    }

    public function mostra_conteudo( ){

        echo $this->conteudo;

    }
}

?>
```

class_caixa.php

É muito útil definir valores pré-definidos nos parâmetros que recebe o construtor, igualando o parâmetro a um valor dentro da declaração de parâmetros da função construtora, deste modo, ainda que se chame ao construtor sem proporcionar-lhe parâmetro, esse iniciará com os valores pré-definidos.
É importante assinalar que nos construtores não se tem porque receber todos os valores para iniciar o objeto. Há alguns valores que se podem iniciar com valores vazios (NULOS) ou qualquer outro valor fixo, como neste caso, o conteúdo da caixa, que inicialmente propusemos que estará vazia.

Destrutores

PHP 5 introduz um conceito de destrutor similar ao de outras linguagens orientadas a objeto, como o Java. O método destrutor será chamado assim que todas as referências a um objeto particular forem removidas ou quando o objeto for explicitamente destruído.

```
<?php
class MinhaClasseDestruivel {
  function __construct( ) {
     print "No construtor\n";
     $this->name = "MinhaClasseDestruivel<br />";
  }

  function __destruct( ) {
     print "Destruindo " . $this->name . "\n";
  }
}

$obj = new MinhaClasseDestruivel( );
?>
```

class_destruct.php

Como os construtores, destrutores pais não serão chamados implicitamente pelo engine.

Para executar o destrutor pai, deve-se fazer uma chamada explicitamente a **parent::__destruct( )** no corpo do destrutor.

Herança em PHP

A programação orientada a objetos tem um mecanismo chamado herança pela que qual se podem definir classes a partir de outras classes. As classes realizadas a partir de outra classe, ou melhor dizendo, que derivam de outra classe, chamam-se classes derivadas. As classes derivadas herdam todos os atributos e métodos da classe base, que podem ter tantos atributos e métodos novos como quiser.
Para ampliar o exemplo, da classe Caixa, vamos criar uma classe derivada chamada **Caixa_Tipo**. Esta classe herda de caixa, mas tem um "tipo", que é a descrição do tipo de coisas que colocamos na caixa. Com isto podemos ter varias caixas, cada uma com coisas diferentes.

```
<?php
require("class_caixa.php");

class Caixa_Tipo extends Caixa{

  private $tipo;

  function define_tipo($novo_tipo){

  $this->tipo = $novo_tipo;

  }
}
?>
```

class_caixa_tipo.php

Na construção dessa classe, herdamos de Caixa todos os atributos e métodos da classe base. Também se definiu um novo atributo, chamado **$tipo**, e um método, chamado **define_tipo( )**, que recebe o tema com o que se deseja inserir na caixa.

Poderíamos utilizar a classe Caixa_Tipo de maneira similar como fazíamos com a classe Caixa original:

```
<?php
        require("class_caixa_tipo.php");
?>
<html>
<head>
<title>Utilizando a classe caixa tipo</title>
</head>

<body>
   <?php

       $minhacaixa_tematica = new Caixa_Tipo( );
       $minhacaixa_tematica->define_tipo("Cabos e conectores");
       $minhacaixa_tematica->introduz("Cabo de rede");
```

```
        $minhacaixa_tematica->mostra_conteudo( );
        echo "<br />";
        $minhacaixa_tematica->introduz("Conector RJ45");
        $minhacaixa_tematica->mostra_conteudo( );

    ?>

</body>
</html>
```

util_class_caixa_tipo.php

Neste caso, o resultado que se obtém é parecido ao que se obtém para a classe base. Mas, quando se mostra o conteúdo de uma caixa, o mais interessante seria, que se indicasse também o tipo de objetos que contem o tipo de caixa. Para isso, temos que redefinir o método mostra_conteudo( ).

Redefinir métodos em classes derivadas

Redefinir métodos significa voltar a codificá-los, isto é, voltar a escrever o seu código para a classe derivada. Neste caso, temos que redefinir o método mostra_conteudo( ) para que mostre também o tipo da caixa.
Para redefinir um método, o único que devemos fazer é voltar a escrever dentro da classe derivada.

```
function mostra_conteudo( ){

  echo "Conteudo da caixa de <b>" . $this->tipo . "</b>: " . $this->conteudo;

}
```

Neste exemplo codificamos de novo o método inteiro para mostrar os dados completos. Em algumas ocasiões é muito útil basear-se na definição de um método da classe base para realizar as ações da classe derivada. Por exemplo, temos que definir um construtor para a classe Caixa_Tipo, para que também se inicie o tipo da caixa. Como já existe um método construtor na classe base, não vale a pena reescrever o código deste, o melhor é chamar ao construtor que tínhamos definido na classe Caixa original, com o qual se iniciarão todos os dados da classe base, e depois realizar a iniciação para os atributos da própria classe derivada.

Para chamar a um método da classe pai dentro do código de um método que estamos redefinindo, utilizamos uma sintaxe como esta:

```
<?php
require("class_caixa.php");

class Caixa_Tipo extends Caixa{

    private $tipo;

    function __construct($altura=1,$espessura=1,$largura=1,$cor="preto",$tipo="Sem classificação"){

            parent::__construct($altura,$espessura,$largura,$cor);
```

```
            $this->tipo=$tipo;

    }

    function define_tipo($novo_tipo){

            $this->tipo = $novo_tipo;

    }
}
?>
```

class_caixa_tipo.php

Aqui vemos a redefinição de construtor, da classe **Caixa**, para a classe **Caixa_Tipo**. O construtor faz primeiro uma chamada ao construtor da classe base, através de uma referência a **parent**. Depois inicia o valor do atributo $tipo, que é especifico da Caixa_Tipo.
Na mesma linha de trabalho, podemos redefinir o método mostra_conteudo( ) baseando-nos no que foi declarado para a classe base. O código ficaria assim:

```
<?php
require("class_caixa.php");

class Caixa_Tipo extends Caixa{

  private $tipo;

    function __construct($altura=1,$espessura=1,
                                    $largura=1,$cor="preto",
                                    $tipo="Sem classificação")
    {

            parent::__construct($altura,$espessura,$largura,$cor);
            $this->tipo=$tipo;

    }

    function define_tipo($novo_tipo){

            $this->tipo = $novo_tipo;

    }
    function mostra_conteudo( ){

            echo "Conteudo da caixa de <b>" . $this->tipo . "</b>: ";

            parent::mostra_conteudo( );
    }

}
?>
```

class_caixa_tipo.php

Operador de Resolução de Escopo (::)
O Operador de Resolução de Escopo, também chamado de **Paamayim Nekudotayim** (significa dois pontos duplos - em Hebreu), ou em termos mais simples, dois pontos

duplos, é um token que permite acesso a métodos ou membros estáticos, constantes, e sobrecarregados de uma classe.
Quando referenciando esses itens de fora da definição da classe, você usa o nome da classe.

Palavra-Chave 'static'
Declarando membros ou métodos de uma classe como estáticos, faz com que esses sejam "chamáveis" de fora do contexto do objeto. Um membro ou método declarado com static não pode ser acessado com uma variável que é uma instância do objeto e não pode ser redefinido em uma subclasse.
A declaração static deve estar depois da declaração de visibilidade. Por compatibilidade com o PHP 4, se nenhuma declaração de visibilidade for usada, então o membro ou método será tratado como se fosse declarado como public static.
Como métodos estáticos podem ser chamados sem uma instância do objeto ter sido criada, a pseudovariável **$this** não é disponível dentro do método declarado como estático.
De fato, chamadas a métodos estáticos são resolvidas em tempo de compilação. Quando usando um nome de classe explícito o método já é identificado completamente e nenhuma das regras de herança se aplicam. Se a chamada for feita por **self** então **self** é traduzido para a classe atual, isso é, a classe à qual o código pertence. Aqui também não se aplicam as regras de herança.
Propriedades estáticas não podem ser acessadas pelo objeto usando o operador seta ->.

```
<?php
class Primeira
{
  public static $meu_estatico = 'Valor estático';

  public function valorEstatico( ) {
     return self::$meu_estatico;
  }
}

class Segunda extends Primeira
{
  public function PrimeiraValorEstatico( ) {
     return parent::$meu_estatico;
  }
}

print Segunda::$meu_estatico . "<br />";
$segunda = new Segunda( );
print $segunda->PrimeiraValorEstatico( ) . "<br />";


print Primeira::$meu_estatico . "<br />";

$primeira = new Primeira( );
print $primeira->valorEstatico( ) . "<br />";
// "Propriedade" Indefinida $meu_estatico
print $primeira->$meu_estatico . "<br />";
```

```
?>
```

util_static.php

Constantes do Objeto

É possível definir valores constantes em cada classe permanecendo a mesma imutável. Constantes diferem de variáveis normais no não uso do símbolo $ para declará-las ou usá-las. Como membros estáticos (static) , valores constantes não podem ser acessados a partir de uma instância de um objeto.

```
<?php
class MinhaClasse{
  const constante = 'valor constante';

  function mostrarConstante( ) {
   echo  self::constante . "<br />";
  }
}

echo MinhaClasse::constante . "<br />";

$classe = new MinhaClasse( );
$classe->mostrarConstante( );
/* echo $classe::constante;  não é permitido */

?>
```

util_const.php

Abstração de um Objeto

Quando pensamos em um tipo de classe, supomos que os programas criam objetos desse tipo. Em alguns casos, porém, é útil declarar classes para as quais o programador nunca pensará em instanciar objetos. Essas classes são chamadas de classes abstratas. Essas classes não podem ser utilizadas para instanciar objetos, porque, classes abstratas são incompletas. As subclasses devem declarar as 'partes ausentes'.

Não é permitido criar uma instância de uma classe que foi definida como abstrata. Qualquer classe que contêm pelo menos um método abstrato deve também ser abstrata. Métodos definidos como abstratos simplesmente declaram a assinatura do método, eles não podem definir a implementação.

A classe que implementa o método abstrato deve definir com a mesma visibilidade ou mais fraca. Se o método abstrato é definido como protected, a implementação da função deve ser definida ou como protected ou como public.

```
<?php
//superclasse abstrata Empregado
abstract class Empregado{
        private $nomeEmp;
        private $sobrenomeEmp;
        private $cpfEmp;
        protected $dados;
```

```
        //construtor com três argumentos
        protected function __construct($nome, $sobrenome, $cpf){
                $this->nomeEmp = $nome;
                $this->sobrenomeEmp = $sobrenome;
                $this->cpfEmp = $cpf;

        }// fim do construtor

        //retorna o nome
        public function getNome( )
        {
                return $this->nomeEmp;
        } //fim do método getNome

        //retorna o sobrenome
        public function getSobrenome( )
        {
                return $this->sobrenomeEmp;
        } //fim do método getSobrenome

        //retorna o cpf
        public function getCpf( )
        {
                return $this->cpfEmp;
        } //fim do método getSobrenome

        //método abstrato responsável por salvar os dados
        abstract protected function salvar( );
}
?>
```

class_abstract_emp.php

A seguir você tem a classe que extenderá a classe abstrata:

```
<?php
require("class_abstract_emp.php");

//classe SalarioEmpregado que estende Empregado
class SalarioEmpregado extends Empregado{
        private $salarioEmp;

        //construtor com quatro argumentos
        public function __construct($nome, $sobrenome, $cpf, $salario){
                //passa para o construtor Empregado
                parent::__construct($nome,$sobrenome,$cpf);

                $this->setSalario($salario);//valida e armazena o salário

        }//fim do construtor SalarioEmpregado
```

```
        //configura o salário
        public function setSalario($salario){
                //se o salário for negativo, configura-o em zero
                $this->salarioEmp = $salario <0 ? 0 : $salario;

        }//fim do método setSalario

        //retorna o salário
        public function getSalario( ){
                return $this->salarioEmp;

        }//fim do método getSalario

        //salva o salário
        public function salvar( ){
                //abre ou cria e abre o arquivo salarios.txt
                //coloca o ponteiro no fim do arquivo (letra a)
          if(!$fp=fopen('salarios.txt','a')){
            throw new Exception('Erro ao abrir o caminho do arquivo');
           }
                $dados = $this->getNome( )." ";
                $dados .= $this->getSobrenome( )."|";
                $dados .= $this->getCpf( )."|";
                $dados .= $this->getSalario( )."\n";
      fwrite($fp,$dados); //escreve os dados no arquivo
      fclose($fp);//fecha o arquivo
   }//fim do método salvar

}
?>
```

class_salario_empregado.php

O arquivo seguinte utiliza a classe para salvar os dados do empregado:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Utilizando classes abstratas</title>
</head>
<body>

<?php
require("class_salario_empregado.php");
//cria objetos da subclasse
try{
  $salarioEmpregado = new SalarioEmpregado("Edson ","Gonçalves","222.123.145-01", 800.00);
  $salarioEmpregado->salvar( );

  $salarioEmpregado = new SalarioEmpregado("Edna ","Gonçalves","222.123.145-01", -5);
```

```
  $salarioEmpregado->salvar( );
}
catch(Exception $e){
  echo $e->getMessage( );
}

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

main_sal_emp.php

Para ilustrar seu aprendizado, construa uma classe de lixeira, onde haverá tipos como **recicláveis: vidros, plástico e papel** e não **recicláveis: lixo orgânico** e **não orgânico**.

Arrays (Matrizes)

Umas das construções mais importantes da programação são chamadas de arrays. Até agora, as variáveis que você viu e utilizou são chamadas de variáveis escalares, que armazenam um único valor. Um array é uma variável que armazena um conjunto ou seqüência de valores. Um array pode conter muitos elementos. Cada elemento pode armazenar um único valor, como texto ou números ou outro array. Um array contendo outro array é conhecido como array multidimensional.

O que é um array?

Um array é um lugar identificado para armazenar um conjunto de valores, permitindo assim armazenar variáveis escalares comuns.
Veja um exemplo simples:
Digamos que você tenha três produtos para vender e quer listá-los. Esses produtos são considerados três valores dentro de uma matriz. Assim teríamos a tabela abaixo:

Monitor	**Teclado**	**Mouse**

produto

Depois de ter as informações como uma matriz, podemos utilizar um loop para realizar a varredura no array e obter os produtos guardados. Com a utilização de funções específicas do PHP, você pode ter, por exemplo, os produtos listados em ordem alfabética.

Arrays numericamente indexados

Suportados pela grande parte das linguagens de programação, os índices indicam de forma numérica qual é o primeiro produto e qual é o último. Uma coisa a ser colocada em pauta é que o primeiro produto é considerado numericamente como zero.

```
<html>
  <head><title>Exemplos de PHP</title></head>
  <body>
<?php

       $produto=array('Monitor','Teclado','Mouse');
```

```
        $imprimir= "Primeiro Produto - $produto[0]<br />";
        $imprimir.="Segundo Produto - $produto[1]<br />";
        $imprimir.="Terceiro Produto - $produto[2]";
        echo $imprimir;
?>


  </body>
</html>
```

ex_cria_matriz_linear.php

Acessando o conteúdo de um array

Para acessar o conteúdo de uma variável, utilize o nome dela. Se a variável for uma matriz, acesse o conteúdo dela utilizando o nome e a chave ou índice. A chave ou índice indica quais valores armazenados você deseja acessar. O índice é colocado entre colchetes depois do nome da variável seguido de um número.
O primeiro elemento de um array é o zero, como já dito anteriormente. Essa convenção é a mesma utilizada por C, C++, Java entre outras linguagens.
Talvez isso seja um pouco confuso no começo, se você não está familiarizado com isso, mas basta entender que se você quiser acessar o valor Monitor dentro da matriz, basta digitar **$produto[0]**.
Para que haja um array como o mostrado anteriormente, você deve usar a construção da linguagem **array( )**.

Uma outra maneira de criar arrays

Uma outra maneira de se criar arrays é fazer a construção direta, como mostrado no exemplo abaixo:

```
<?php

        $produto[0]="Monitor";
        $produto[1]="Teclado";
        $produto[2]="Mouse";

        echo $produto[0];
?>
```

Utilizando loops para acessar o array

Como o array está indexado por uma seqüência de números, podemos utilizar um loop para exibir todo o conteúdo mais eficientemente:

```
<html>
  <head><title>Exemplos de PHP</title></head>
  <body>
<?php

        $produto[0]="Monitor";
        $produto[1]="Teclado";
        $produto[2]="Mouse";

        for($i=0;$i<3;$i++){
```

```
                echo "O produto $i é: $produto[$i]<br />";
        }
?>
  </body>
</html>
```

ex_matriz_ac_loop.php

A diferença é que este loop exigirá menos digitação. Isso torna seu código mais viável e rápido.

Arrays Associativos

Arrays associativos são chaves ou índices que usamos para acessar cada valor armazenado.
O código a seguir mostra um exemplo de um array associativo:

```
<html>
 <body>
<?php
        $preco=array('Monitor'=>550,'Teclado'=>48,'Mouse'=>15);

        echo $preco ['Monitor'];
?>
 </body>
</html>
```

ex_arr_ass.php

Note que ao testar o código, você terá na sua tela o valor de **550**, que é o valor que está atribuído à chave **Monitor**.

Uma outra maneira de se acessar as informações do array é utilizando esta forma ligeiramente diferente do numérico:

```
<?php
        $produto["Monitor"]=550;
        $produto["Teclado"]=48;
        $produto["Mouse"]=15;
        echo $produto["Monitor"];
?>
```

Utilizando loops com each( ) e list( )

Como os índices nesse array associativo não são números, não podemos utilizar um contador simples em um loop for para trabalhar com a matriz. Veja abaixo um código funcional:

```
<?php
$precos=array('Monitor'=>550,'Teclado'=>48,'Mouse'=>15);

while($dados = each($precos)){
        echo $dados["key"]." - ".$dados["value"]."<br />";
}
?>
```

Com a instrução **each( )**, temos as localizações **key** que contém a chave do elemento atual e as localizações **value** que contém o valor do elemento atual.
Uma outra maneira é utilizar valores numéricos, onde **key** equivale a **0** e **value** equivale a **1**.

```
<?php
$precos=array('Monitor'=>550,'Teclado'=>48,'Mouse'=>15);

while($dados = each($precos)){
	echo $dados[0]." - ".$dados[1]."<br />";
}
?>
```

No caso acima fora utilizado localizações identificadas não nomeadas e sim numeradas.
O script a seguir demonstra uma maneira diferente de se ler os dados de um array de maneira simples e mais compreensível. A função **list( )** pode ser utilizada para dividir um array em vários valores. Neste caso, podemos separar dois dos valores que a função **each( )** oferece:

```
<?php
$precos=array('Monitor'=>550,'Teclado'=>48,'Mouse'=>15);

while(list($produto,$preco) = each($precos)){
	echo "$produto - $preco<br />";
}
?>
```

Um caso bem comum é o de você desejar criar dois loops em um mesmo script. Para isso utilize a função **reset( )** para reiniciar o array. Isso acontece porque o cursor após a execução de um loop se encontra no fim do array.
O script completo seria:

```
<html>
<body>
<?php
$precos=array('Monitor'=>550,'Teclado'=>48,'Mouse'=>15);

while($dados = each($precos)){
	echo $dados["key"]." - ".$dados["value"]."<br />";
}

reset($precos);
while(list($produto,$preco) = each($precos)){
	echo "$produto - $preco<br />";
}
?>
</body>
</html>
```

ex_loop_each_list.php

Com uma variação do loop for, temos com a junção de each ao loop, fora criado um

loop especial para coleções, que funciona muito bem nestes casos. Seu nome é...

foreach

O PHP5 inclui uma construção foreach, muito parecido com o PERL e outras linguagens. Isto simplesmente oferece uma maneira fácil de interar sobre matrizes. Há duas sintaxes; a segunda é uma abreviatura, mas útil, da primeira:

```
foreach(expressao_matriz as $valor) instrucoes
```

```
foreach(expressao_matriz as $chave => $valor) instruções
```

A primeira forma varre um array dado por expressao_matriz. Em cada 'loop', o valor do elemento corrente é atribuído a $valor e o ponteiro interno da matriz é avançado em uma posição (assim, na próxima iteração, você estará olhando para o próximo elemento).
A segunda forma faz a mesma coisa, exceto pelo fato de que a chave do elemento atual será atribuída à variável **$chave** em cada iteração.

Obs.: Quando o foreach inicia sua primeira execução, o ponteiro interno da matriz é *zerado automaticamente*, posicionando-o no início. Isto significa que você não precisa chamar **reset( )** antes de um loop foreach .
Obs 2.: Note também que foreach opera sobre uma cópia do array especificado, não o próprio array, portanto o ponteiro do array não é modificado como na instrução **each( )**, que altera o elemento do array selecionado, mas isso não se reflete o array original.
Obs 3.: foreach tem a habilidade de evitar mensagens de erro com o **@**.

Você pode ter notado que os seguintes itens são funcionalmente idênticos:

```
reset ($matriz); //a função reset() reinicia a matriz
while (list(, $valor) = each ($matriz)) {
    echo "Valor: $valor<br />\n";
}//exemplo de utilização do loop while com a função each()
```

```
foreach ($matriz as $valor) {
    echo "Valor: $valor<br />\n";
} //exemplo do loop foreach()
```

Os seguintes também são funcionalmente idênticos:

```
foreach ($matriz as $chave => $valor) {
    echo "Chave: $chave; Valor: $valor<br>\n";
}
```

Mais alguns exemplos para demonstrar os usos:

```
<?php
$a = array ("Roberto", "João", "Luana", "Lilian");

foreach ($a as $v) {
   echo 'O nome atual de $a é: '.$v.'<br>';
}
```

```
?>
```

ex_foreach.php

```
<?php
$a = array ("Roberto", "João", "Luana", "Lilian");

$i = 0;

foreach($a as $v) {
    echo "\$a[$i] => $v.<br>\n";
    $i++;
}
?>
```

ex_foreach_2.php

```
<?php
$a = array (
    "primeiro" => 'Roberto',
    "segundo" => 'João',
    "terceiro" => 'Luana',
    "quarto" => 'Lilian'
);

foreach($a as $c => $v) {
    echo "$c => $v.<br>";
}
?>
ex_foreach_3.php

<?php
$a[0][0] = "Roberto";
$a[0][1] = "João";
$a[1][0] = "Luana";
$a[1][1] = "Lilian";

foreach($a as $v1) {
    foreach ($v1 as $v2) {
        echo "$v2<br>";
    }
} //exemplo da utilização do loop com array multidimensional
?>
```

ex_foreach_arr_mult.php

```
1.     <?php
2.     foreach(array('Roberto', 'João', 'Luana', 'Lilian',
'Roberta') as $v) {
2.1.1. echo "$v<br>";
3.     }// exemplo da utilização do loop com array dinâmico
4.     ?>
```

ex_foreach_arr_dinam.php

Exemplo prático do uso do foreach

Considere a seguinte situação:
Você tem um formulário de informações e gostaria de varrê-lo para saber quais foram escolhidos pelo usuário. Veja o caso:

```
<html>
<head>
<title>Utilizando o loop Foreach</title>
</head>
<body>
<?php
if(count($_POST["musica"])>0){
 echo "<h2>Você escolheu as músicas</h2>";
        foreach($_POST["musica"] as $musica)
        {
                echo "$musica<br>";

        }
}
?>
<form name="form1" method="post" action="<?=$PHP_SELF?>">
  <h2>Escolha as músicas</h2>
  <p>
    <input name="musica[]" type="checkbox" id="musica[]" value="rock">
    Rock<br>
    <input name="musica[]" type="checkbox" id="musica[]" value="pop">
    POP<br>
    <input name="musica[]" type="checkbox" id="musica[]" value="dance">
    Dance<br>
    <input name="musica[]" type="checkbox" id="musica[]" value="mpb">
    MPB<br>
    <input name="musica[]" type="checkbox" id="musica[]" value="sertanejo">
    Sertanejo</p>
  <p>
    <input type="submit" name="Submit" value="Escolher">
  </p>
  </form>
</body>
</html>
```

ex_prat_foreach.php

Arrays Bidimensionais

As matrizes não têm de ser uma lista simples de chaves e valores – cada localização na matriz pode armazenar outra matriz. Dessa maneira, podemos criar uma matriz bidimensional. Você pode pensar em uma matriz de duas dimensões como sendo uma grade, com largura e altura ou linhas e colunas.
Veja no quadro abaixo:

Código	Descrição	Preço
MO	***Monitor***	***550,00***
TE	***Teclado***	***48,00***
MS	***Mouse***	***15,00***

No exemplo abaixo, você tem um array bidimensional com a seqüência que o acessa.

```
<?php
$produtos=array(
```

```
                        array("MO","Monitor",550),
                        array("TE","Teclado",48),
                        array("MS","Mouse",15)
                    );

echo   "| ".$produtos[0][0]." | ".$produtos[0][1]." | ".$produtos[0][2]." | <br />
                    | ".$produtos[1][0]." | ".$produtos[1][1]." | ".$produtos[1][2]." | <br />
                    | ".$produtos[2][0]." | ".$produtos[2][1]." | ".$produtos[2][2]." |";

?>
```

arr_bid.php

Conclui-se que o primeiro valor é a linha e o segundo é a coluna, logo:

$produtos[$linha][$coluna];

Para fazer uma varredura no array bidimensional, essas informações são imprescindíveis.

```
<?php
$produtos=array(
                        array("MO","Monitor",550),
                        array("TE","Teclado",48),
                        array("MS","Mouse",15)
                    );
        for($linha=0;$linha<3;$linha++){
                for($coluna=0;$coluna<3;$coluna++){
                        echo "| ".$produtos[$linha][$coluna]." ";
                }//fim do loop da coluna
                echo "|<br />";
        }//fim do loop da linha
?>
```

loop_arr_bid.php

Classificando arrays

Freqüentemente é útil classificar os dados relacionados armazenados em um array. Pegar um array dimensional e classificá-lo no pedido é bem fácil.

Utilizando sort( )

O seguinte código resulta no array sendo classificado em ordem alfabética crescente:

```
<?php
$produtos=array("Monitor","Teclado","Mouse","Armário");
sort($produtos);
for($i=0;$i<4;$i++){
        echo $produtos[$i]."<br>";
}
?>
```

arr_func_sort.php

A função **sort( )** faz distinção de letras maiúsculas e minúsculas. Todas as letras maiúsculas vêm antes de todas as letras minúsculas. Então 'A' é menor que 'Z', mas 'Z' é menor que 'a'. O inverso seria **rsort( )**.

Reordenando arrays aleatoriamente

Para alguns sites, talvez você queira manipular a ordem do array de outras maneiras. A função **shuffle( )** reordena aleatoriamente os elementos de um array( ).

```
<?php
$produtos=array("Monitor","Teclado","Mouse","Armário");
shuffle($produtos);
for($i=0;$i<4;$i++){
        echo $produtos[$i]."<br>";
}
?>
```

arr_func_shuffle.php

Redirecionando um usuário

Para redirecionar um usuário indesejado, utilize a função **header( )**.

```
header("Location: http://www.integrator.com.br");
```

Nota: A função **header( )** deve ser usada antes de lançado qualquer informação no browser, caso contrário, esta retornará um erro.

Expressões Regulares (compatíveis com o padrão POSIX)

Expressões regulares são usadas para manipulações complexas de strings no PHP. Algumas das funções que suportam as expressões regulares:

- **ereg( )**
- **ereg_replace( )**
- **eregi( )**
- **eregi_replace( )**

Um exemplo é mostrado a seguir:

```
<?php
$string="Texto";
// Retorna true se "Te" existir em qualquer lugar de $string.
if(ereg ("Te", $string))
        echo "existe em qualquer lugar<br />";

// Retorna true se "te" existir no início de $string.
```

```
if(eregi ("^te", $string))
        echo "existe no início<br />";

// Retorna true se "to" existir no final de $string.
if(ereg ("to$", $string))
        echo "existe no final<br />";

// Retorna true se o navegador do cliente  for Netscape 2, 3 ou MSIE 6.

if(eregi ("(ozilla.[23]|MSIE.6)", $_SERVER['HTTP_USER_AGENT']))
        echo "A versão é a {$_SERVER['HTTP_USER_AGENT']} <br />";

echo "<br />";
// Acrescenta o texto no início de $string.
echo $string = ereg_replace ("^", "Esse ", $string);

echo "<br />";
// Acrescenta o texto no final de $string.
echo $string = ereg_replace ("$", " aqui", $string);
echo "<br />";
// Remove todos caracteres t  em $string.
echo $string = eregi_replace ("t", "", $string);
?>
```

expressoes_regulares.php

As expressões regulares

Tabela de expressões regulares	
Símbolo	**O que significa?**
^	Início da linha
$	Fim da linha
n?	Zero ou somente uma única ocorrência de caracteres '**n**'
n*	Zero ou mais ocorrência de caracteres '**n**'
n+	Pelo menos um ou mais ocorrências de caracteres '**n**'
n{2}	Exatamente duas ocorrências de '**n**'
n{2,}	Pelo menos 2 ou mais ocorrências de '**n**'
n{2,4}	De 2 até 4 ocorrências de '**n**'
.	Qualquer caractere único
()	Parênteses para agrupar expressão
(.*)	Zero ou mais ocorrências de qualquer caractere único
(n|a)	Ou '**n**' ou '**a**'
[1-6]	Qualquer dígito único entre **1** e **6**
[c-h]	Qualquer letra única em minúscula entre **c** e **h**
[D-M]	Qualquer letra única em maiúscula entre **D** e **M**
[^a-z]	Qualquer caractere único **EXCETO** qualquer letra que esteja entre a seqüência à até z.

Um exemplo da aplicação de expressões regulares

^.{2}[a-z]{1,2}_?[0-9]*([1-6]|[a-f])[^1-9]{2}a+$

^.{2}	O começo da linha com dois quaisquer caracteres
[a-z]{1,2}	Seguido por 1 ou 2 letras em minúsculo,
?	Seguido por um sublinhado (underline) opcional,
[0-9]*	Seguido por zero ou mais dígitos,

([1-6]|[a-f])	Seguido por um ou outro dígito entre 1 e 6 OU uma letra em minúsculo entre a e f,
[^1-9]{2}	Seguido por dois quaisquer caracteres EXCETO dígitos entre 1 e 9 (0 é possível),
a+$	seguida por pelo menos um ou mais ocorrências de '**a**' até o fim da linha.

Validando formatações e verificando preenchimentos

Algumas vezes você será obrigado a fazer algumas verificações a fim de evitar problemas na hora de inserir dados em um banco de dados, tornando assim o seu site estável. Os exemplos a seguir demonstram como você pode facilitar sua vida adicionando apenas algumas pequenas funções que validam e verificam.

A função ereg( )

A função **ereg(** string expressao, string variavel [, array registros] **)** é uma função **booleana**, ou seja, retorna **TRUE**, caso tenha sucesso em sua verificação:
Verifica se a *variável* é similar com a expressão regular definida em *expressao* em um modo sensível a distinção de caracteres (case sensitive).
Se existirem parênteses de substrings na *expressao* e for passado o terceiro parâmetro *registros* para a função, a execução guardará os elementos resultantes na matriz *registros*. $registros[1] irá conter a substring indicada pelo primeiro parênteses da esquerda; $registros[2] contém a segunda substring, e assim por diante. $registros[0] conterá uma cópia completa da variável casada.

```
<?php
	function valida_cep($_cep) {
		// Valida um CEP no formato 99999-999 ou 99999999
		if(ereg("^([0-9]){5}-?([0-9]){3}$",$_cep,$_c)) {
			return Array($_c[1],$_c[2]);
		}
		else {
			return FALSE;
		}
	}

	$_cp = array("05735-010","03578000","A0010-000","98120");
	foreach($_cp as $_cep) {
		if(($_r=valida_cep($_cep))===FALSE) {
			echo "$_cep: CEP Incorreto";
		}
		else {
			echo  "$_cep: CEP OK";
		}
		echo "<br/>";
	}
?>
```

ereg_valida_cep.php

A função eregi( )

A função **eregi (** string expressao, string variavel [, array registros] **)** retorna valores **booleanos** assim como a função **ereg( )** .
Essa função é idêntica à função **ereg ( )** com exceção de não fazer distinções alfabéticas entre caracteres (case insensitive) na hora de casar resultados.

```
<?php
function valida_email($_email) {
// Valida um e-mail no formato maiúsculo ou minúsculo
if (eregi("^[a-zA-Z0-9_\.]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+$",
$_email,$_e))
{
        return Array($_e[1],$_e[2],$_e[3] );
}
else {
        return FALSE;
}
}

$_emails = Array("edson@integrator.com.br","EDSON.g@gmail.com",
                        "edgonn@aaa","edsonaaa.com");
foreach($_emails as $_email) {
        if(($_r=valida_email($_email))===FALSE) {
                echo "$_email: e-mail Incorreto";
        }
        else {
                echo  "$_email: e-mail OK";
        }
        echo "<br/>";
}
?>
```

eregi_valida_email.php

A função ereg_replace( )

A função **ereg_replace (** string expressao, string substituicao, string variavel **)** busca em variável resultados para a *expressao*, substituindo se casar pelo texto em *substituição*.
A *variável* modificada será retornada (poderá ocorrer da string original ser retornada caso não aconteça nenhuma substituição).
Se a *expressao* contiver parênteses de substrings, a substituicao será realizada através do formato \\digit, que indicará qual parte do texto substituído deverá ser retornada; \\0 retornará a string na integra. Até nove substrings podem ser usadas. Os parênteses podem ser aninhados, sendo que a contagem será feita através da quantidade de parênteses abertos.
Se não tiverem valores que casem com o parâmetro variavel, o resultado será a variável inalterada.

```
<?php
        $_str = "http://www.integrator.com.br";
        echo ereg_replace("http://www", "ftp://ftp", $_str);
        echo "<br/>";
```

```
        echo ereg_replace("(http):", "\\1s:", $_str);//contagem com um parênteses
        echo "<br/>";
        echo ereg_replace("[[:alpha:]]+://[^<>[:space:]]+[[:alnum:]/]",
                "<a href=\"\\0\">\\0</a>", $_str); //retorna na integra \\0
        echo "<br/>";
        $string = "Esse é um teste";
        echo ereg_replace (" é um", " está sendo", $string);
        echo "<br/>";
        echo ereg_replace ("( )é", "\\1foi", $string);
        echo "<br/>";
        echo ereg_replace ("(( )é)", "\\2será", $string);//contagem com dois parênteses

?>
```

ereg_replace.php

Metacaracteres em expressões regulares	
[:digit:]	Dígitos de 0 até 9
[:alnum:]	Caracteres alfanuméricos de 0 até 9 ou A até Z ou a até z.
[:alpha:]	Caracteres alfabéticos de A até Z ou a até z.
[:blank:]	Caracteres de espaço e tabulação.
[:xdigit:]	Dígitos hexadecimais
[:punct:]	Símbolos de pontuação . , " ' ? ! ; :
[:print:]	Todos os caracteres que podem ser impressos.
[:space:]	Caracteres de espaço.
[:graph:]	Todos os caracteres impressos, menos espaço.
[:upper:]	Caracteres alfabéticos maiúsculos de A até Z.
[:lower:]	Caracteres minúsculos de a até z.
[:cntrl:]	Caracteres de control.

Tabela de símbolos de caracteres	
\a	caractere control
\b	Backspace
\f	form feed
\n	line feed
\r	carriage return
\t	horizontal tab
\v	vertical tab

A função eregi_replace( )

Não será colocado um exemplo nessa apostila por ser idêntica a ereg_replace( ), diferenciando-se apenas por ser insensível a maiúsculas e minúsculas.

A função str_replace( )

A função **str_replace (** mixed pesquisa, mixed substitui, mixed assunto [, int &count] **)** substitui todas as ocorrências da string de procura com a string de substituição.
Esta função retorna uma string ou um array com todas as ocorrências de pesquisa em assunto substituídas com a o valor dado para substituir.

```
<?php
        $str_inv = array("/", "\\", "'");//lista de strings inválidas
        $_str="Esse/é um exemplo de\\informação inválida'//como você pode ver<br>";
        echo $str_retornada = str_replace($str_inv, " ", $_str, $c);
        echo $c;
?>
```

str_replace.php

A função number_format( )

A função **number_format (** float number [, int decimals] **)** ou **number_format (** float number, int decimals, string dec_point, string thousands_sep **)** formata um número com os milhares agrupados.
A função **number_format( )** retorna uma versão formatada de *number*. Esta função aceita um, dois ou quatro parâmetros (não três):
Se apenas um parâmetro é dado, number será formatado sem decimais, mas com uma virgula (",") entre cada grupo de milhar.
Se dois parâmetros são dados, number será formatado com o número de casas decimais especificadas em decimals com um ponto (".") na frente, e uma vírgula (",") entre cada grupo de milhar.
Se todos os quatro parâmetros forem dados, number será formatado com o número de casas decimais em decimals, dec_point ao invés do ponto (".") antes das casas decimais e thousands_sep ao invés de uma vírgula (",") entre os grupos de milhares.

```
<?php
        $numero = 125456.72;
        $f1 = number_format($numero);
        $f2 = number_format($numero,3);
        $f3 = number_format($numero,2,",",".");
        echo "number_format, com 1 parâmetro: $f1 ";
        echo "<br>com 2 parâmetros: $f2";
        echo "<br>com todos os parâmetros: $f3";
?>
```

number_format.php

A função nl2br( )

A função **nl2br (** string **)** insere quebras de linha HTML antes de todas newlines em uma string.
Como essa função retorna string com '
' inserido antes de todas as newlines, é excelente em uso com a tag <TEXTAREA> do HTML.

```
<?php
        $texto = "Este é um texto\ncom line-feed como quebra\n de página";
        echo "sem nl2br( ):<br />$texto<br /><br />Com nl2br( ):<br />" . nl2br($texto);
?>
```

nl2br.php

A função wordwrap( )

A função **wrodwrap(** string string [, int largura [, string quebra [, boolean corte]]] **)** insere na string informada o caracter de quebra de linha \\n (ou um outro caracter qualquer, por exemplo
), a cada n caracteres (o padrão é 75). O parâmetro corte serve para forçarmos a quebra mesmo para palavras extremamente grandes.
Essa função é excelente quando temos um site formatado em determinada largura (dentro ou não de uma tabela), mas por ser dinâmico, as informações postas podem ser grandes por erros na digitação, bagunçando assim o layout criado.

```
<?php
        $texto = "A função wrodwrap insere na string informada o caracter de
```

```
                    quebra de linha \\n (ou um outro caracter qualquer, por exemplo
                    <BR>), a cada n caracteres (o padrão é 75). O parâmetro corte
                    serve para forçarmos a quebra mesmo para palavras
extremamente
                    exxxxxxxxteeeeeeeeeeeeensssssssssssssssaaaaaaaaaaassssssss.";
        $texto = htmlentities($texto);
        echo wordwrap($texto);
        echo "<br><br><b>Sem o parâmetro corte:</b><br>";
        echo wordwrap($texto,30,"<br/>");
        echo "<br><br><b>Sem o parâmetro corte:</b><br>";
        echo wordwrap($texto,30,"<br/>",TRUE);
?>
```

wordwrap.php

A função strip_tags( )

Extremamente útil em um sistema dinâmico, principalmente de uso público como fóruns e livro de visitas.

A função **strip_tags (** string str [, string allowable_tags] **)** retira tags HTML e PHP de uma string. Você pode utilizar o segundo parâmetro, que é opcional, para indicar tags que não devem ser retiradas.

```
<?php
        $texto = "<b>Exemplo de Strig_tags</b>
                    <script>alert('Este é um Java
                    script embutido no texto')</script>";
        echo $texto . "<br>";
        echo strip_tags($texto," <b>");
?>
```

strip_tags.php

A função htmlentities( )

Também muito útil, a função **htmlentities(** string string [, int quote_style [, string charset]] **)** tem a habilidade de converter as tags para entidades HTML, não causando formatações indesejadas em sites dinâmicos, principalmente os de inserção de conteúdo público.

```
<?php
        $valor = "<B><FONT COLOR='RED'>
                    Exemplo   de htmlentities( )
                    </B></FONT><BR>";
        echo $valor;
        echo htmlentities($valor);
?>
```

htmlentities.php

O inverso dessa função é **html_entity_decode( )**.

```
<?php
        $string = "&lt;B&gt;&lt;FONT COLOR=
                    &#039;RED&#039;&gt;Exemplo de
                    htmlentities()&lt;/B&gt;&lt;/FONT&gt;&lt;BR&gt;";
```

```
        echo html_entity_decode($string);
?>
```

html_entity_decode.php

Funções úteis, mas não essenciais

As funções mostradas a seguir são úteis em alguns casos, mas não serão explicadas, pois são compreensíveis em seus exemplos:

```
<?php
$str_m="esse é um texto que será manipulado<br>";
$str_M="VEJA SÓ ESSE TEXTO<br>";
echo strtolower($str_M);//converte para minúsculo
echo strtoupper($str_m);//converte para maiúsculo
echo ucwords($str_m);//capitula a primeira letra de cada palavra
echo ucfirst($str_m);//capitula apenas a primeira letra da frase

?>
```

diversas_func_str.php

Enviando um e-mail

Função mail( )

Envia um e-mail de um formulário ou diretamente por uma página em PHP.
Escopo da função:

```
bool mail ( string para, string assunto, string mensagem [, string
cabeçalhos_adicionais [, string parametros_adicionais]])
```

```
<?php
$para="integrator@integrator.com.br";
$assunto="Envio de e-mail";
$mensagem="Testando o e-mail pelo PHP";

mail( $para,$assunto,$mensagem);
?>
```

O exemplo acima envia e-mail diretamente para o e-mail do integrator.
Abaixo você tem um exemplo de envio de html:

```
<?
$para="integrator@integrator.com.br";
$assunto="Envio de e-mail em HTML";
$mensagem="<h2>Esta é uma mensagem em formato HTML</h2>
                        <strong>Estas informações são enviadas como estão formatadas</strong>";

$headers   = "MIME-Version: 1.0\r\n";

$headers .="Content-Type: text/html; charset=iso-8859-1\n";
```

```
$headers .= "From: \"Edson\" <edson@integrator.com.br>\r\n";

$headers .= "Cc:contato@integrator.com.br\r\nCc:webmaster@integrator.com.br\r\n";//cópias carbono

$headers .= "Bcc:webmaster@cestashow.com\r\n";//Cópia carbono oculta

mail($para, $assunto, $mensagem, $cabecalho); //envio do e-mail

?>
```

Nota: Este exemplo só funciona onde o **PHP.INI** estiver configurado para apontar para o servidor de e-mail do seu computador.

Trabalhando com arquivos

Até agora, você trabalhou com envio e recebimento de informações que não eram armazenadas. Mas antes de você conhecer o bancos de dados, você terá uma breve inicialização por manipulação de arquivos de texto simples – flat file.
As funções que iremos utilizar são **fopen( )**, **fwrite( )** e **fclose( )**.

Descrição:

Função	O que faz
fopen( )	abre um arquivo
fwrite( )	escreve em um arquivo
fclose( )	fecha um arquivo

```
<?php
// Abre ou cria o arquivo exemplo1.txt
// "a" representa que o arquivo é aberto para ser escrito
$fp = fopen("exemplo1.txt", "a");

// Escreve "primeiro exemplo" no exemplo1.txt
$escreve = fwrite($fp, "primeiro exemplo");

// Fecha o arquivo
fclose($fp);
?>
```

ex_cria_escreve.php

Como resultado é criado um arquivo chamado **exemplo1.txt**, escrito "**primeiro exemplo**". Se você repetir a ação, "**primeiro exemplo**" será escrito novamente no fim desse arquivo.
É importante notar que você tem diversas maneiras de abrir um arquivo, que são representadas por letras. No exemplo acima utilizamos o modo "a", veja abaixo a lista de todos os modos e suas descrições:

Tabela de Códigos e Significados da função fopen( )		
Cód.	**Significado**	**Informações Adicionais**

'r'	Abre somente para leitura	Coloca o ponteiro no começo do arquivo.
'r+'	Abre para leitura e gravação	Coloca o ponteiro no começo do arquivo.
'w'	Abre somente para gravação	coloca o ponteiro no começo do arquivo e apaga o conteúdo que já foi escrito. Se o arquivo não existir, tentar criá-lo.
'w+'	Abre para leitura e escrita	coloca o ponteiro no início do arquivo e apaga o conteúdo que já foi escrito. Se o arquivo não existir, tentar criá-lo.
'a'	Abre o arquivo somente para escrita	coloca o ponteiro no fim do arquivo. Se o arquivo não existir, tentar criá-lo.
'a+'	Abre o arquivo para leitura e gravação	coloca o ponteiro no fim do arquivo. Se o arquivo não existir, tentar criá-lo.
'x'	Cria e abre somente para gravação	coloca o ponteiro no início do arquivo. Se o arquivo já existe, a chamada a **fopen( )** irá falhar, retornando **FALSE** e gerando um erro nível **E_WARNING**. Se o arquivo não existe, tenta criá-lo.
'x+'	Cria e abre para leitura e escrita	coloca o ponteiro no início do arquivo. Se o arquivo já existe, a chamada a **fopen( )** irá falhar, retornando **FALSE** e gerando um erro nível **E_WARNING**. Se o arquivo não existe, tenta criá-lo.

Com estas três funções você pode criar, escrever e fechar um arquivo facilmente. Você também pode usar a função **fputs( )**.
fputs( ) é uma função sinônima (alias) para **fwrite( )**, e é idêntica em todas as maneiras.

Armazenando e recuperando informações

Como ocorre com o banco de dados, após armazenar as informações você deve reaproveitá-las. Para isso você deverá usar algumas funções extras.

Função	O que faz
explode( )	Divide uma string
opendir( )	abre um diretório
readdir( )	lê os arquivos de um diretório
closedir( )	fecha um diretório
file_exists( )	verifica se um arquivo existe

```
<html>
<head><title>Cadastro de usuários</title></head>
<body>
<form method="POST" action="cad_dados_txt.php">
Nome: <input type="text" size="10" name="nome"><br>
E-mail: <input type="text" size="10" name="email"><br>
Endereço: <input type="text" size="10" name="endereco"><br>
Telefone: <input type="text" size="10" name="telefone"><br>
```

```
<br><input type="submit" value="cadastrar">
</form>
</body>
</html>
```

form_cad_dados.php

```
<?php
// Verifica se todos os campos foram preenchidos
if (!$nome || !$email || !$endereco || !$telefone) {
    echo "preencha todos os campos";
} else {
    // Verifica se um usuário com o mesmo nome ja foi cadastrado
    if(!file_exists($nome . ".txt")) {
        // Criamos o arquivo do usuário com w+
        $cria = fopen($nome . ".txt", "w+");

        // Aqui iremos declarar as informações do usuário
        // São separadas por | para depois podemos recupera-las
com explode
        $dados .= "$nome|";
        $dados .= "$email|";
        $dados .= "$endereco|";
        $dados .= "$telefone";

        // Agora escrevemos estes dados no arquivo
        $escreve = fwrite($cria,$dados);

        // Fechando o arquivo
        fclose($cria);

        // Exibe a mensagem de usuário cadastrado
        echo "usuário cadastrado com sucesso!";
    } else {
        // Se ja houver um usuário cadastrado com o mesmo nome
        echo "um usuário chamado $nome ja foi cadastrado";
    }
}
?>
```

cad_dados_txt.php

```
<?php
// Primeiro definiremos onde estão os arquivos
// ./ significa que os arquivos estão no diretório atual
$dir = "./";

// Abre o diretorio $dir
$abredir = opendir($dir);

// Faz o loop para a exibição de usuários
while ($arqs = readdir($abredir)) {
    // Aqui fazemos o php tirar "." e ".." que são "bugs" do
readdir()
    // Também faz com que só sejam abertos arquivos de texto
    if ($arqs != "." && $arqs != ".." && is_file($arqs) &&
```

```
ereg(".txt", $arqs)) {

        // Agora iremos abrir arquivo por arquivo, e exibir os
dados do usuário
        // Usamos o "r" pois somente queremos ler o arquivo
        $abre = fopen($arqs,"r");

        // Usamos fread agora para ler o arquivo
        $le = fread($abre,filesize($arqs));

        // Agora vem o grande truque, separamos os dados pelo
"|" com explode
        $dado = explode("|",$le);

        // Define os registros
        $nome = $dado['0'];
        $email = $dado['1'];
        $endereco = $dado['2'];
        $telefone = $dado['3'];

        // Vamos mostrar os dados que obtivemos
        echo "Usuário: <b>$nome</b><br>";
        echo "nome: $nome<br>";
        echo "e-mail: $email<br>";
        echo "endereço: $endereco<br>";
        echo "telefone: $telefone<br><br>";
    }
}

// Fecha o diretorio
closedir($abredir);
?>
```

ver_dados.php

Uma alternativa a escrita de arquivos

No PHP5, você tem uma alternativa para escrever arquivos, saindo do tradicional **fopen( )**, **fwrite( )** e **fclose( )**, chamado de **file_put_contents ( )**.

```
<?php
        $_str = "Escrevendo arquivos com file_put_contents( )";
        file_put_contents("arquivo.txt",$_str);
        $_str = "<P>Data: <?php echo date(\"d-m-Y H:i:s\")?></P>";
        //acrescentando ao final do arquivo com o parâmetro FILE_APPEND
        file_put_contents("arquivo.txt",$_str, FILE_APPEND);
        echo htmlentities(file_get_contents("arquivo.txt"));
        include_once("arquivo.txt");
?>
```

file_put_contents.php

Uma alternativa a leitura de arquivos

Em substituição da função **fread( )** você pode usar a função **file_get_contents( )**.

```
<?php
```

```
        echo file_get_contents("arquivo.txt");
?>
```

file_get_contents.php

Trabalhando com Datas

Com a função date( ), você pode retornar valores de datas nos seus scripts PHP.
A função date( ) aceita dois parâmetros, sendo que um deles é opcional.

```
<?php
echo date("d/m/Y");
?>
```

data_formatada.php

O exemplo acima retornará a data atual do sistema no formato dd/mm/aaaa.
Uma coisa a ser levada em consideração é que como a linguagem está em inglês, algumas opções não estão no formato desejado. Abaixo veja a tabela:

Código	***Descrição***
a -	"am" ou "pm"
A -	"AM" ou "PM"
B -	Swatch Internet time
d -	dia do mês, 2 digitos com zeros à esquerda; i.e. "01" até "31"
D -	dia da semana, texto, 3 letras; e.g. "Fri"
F -	mês, texto, longo; e.g. "January"
g -	hora, Forma com 12-horas sem zeros à esquerda; i.e. "1" até "12"
G -	hora, Forma com 24-horas sem zeros à esquerda; i.e. "0" até "23"
h -	hora, Forma com 12-horas; i.e. "01" até "12"
H -	hora, Forma com 24-horas; i.e. "00" até "23"
i -	minutos; i.e. "00" até "59"
I -	"1" Se no horário de verão, senão "0".
j -	Dia do mês sem zeros à esquerda; i.e. "1" até "31"
l-	dia da semana, texto longo; e.g. "Friday"
L -	booleano se é um ano bissexto; i.e. "0" ou "1"
m -	mês; i.e. "01" até "12"
M -	mês, texto, 3 letras; e.g. "Jan"
n -	mês sem zeros à esquerda; i.e. "1" até "12"
O -	Diferença entre o horário de Greenwich em horas; e.g. "+0200"
r -	RFC 822 formatted date; e.g. "Thu, 21 Dec 2000 16:01:07 +0200"
s -	segundos; i.e. "00" até "59"
S -	Sufixo ordinal para o dia do mês, 2 caracteres; i.e. "st", "nd", "rd" or "th"
t -	número de dias do dado mês; i.e. "28" até "31"
T -	Timezone setting desta máquina; e.g. "EST" or "MDT"
U -	segundos desde a época Unix (January 1 1970 00:00:00 GMT)
w -	dia da semana, numérico, i.e. "0" (domingo) até "6" (Sábado)
W -	ISO-8601 números de semanas do ano, semana começa na segunda-feira
Y -	ano, 4 dígitos; e.g. "1999"
y -	ano, 2 dígitos; e.g. "99"
z -	dia do ano; i.e. "0" até "365"

Z -	timezone offset em segundos (i.e. "-43200" to "43200"). O offset para as timezones oeste de UTC é sempre negativa, e para as leste de UTC é sempre positiva.

Existem alguns casos em que você poderá inserir dados em formato de data no MySQL. Se isso acontecer, você terá o seguinte formato: *aaaa/mm/dd*.

MySQL

Até agora, você está em contato somente com o PHP, mas em si, o PHP não teria grande serventia se não houvesse uma integração com banco de dados.
Utilizar um banco de dados é a melhor decisão que você poderá tomar, mesmo porque, o acesso a dados é bem mais rápido que o acesso a arquivos simples (*flat files*).
Em termos mais concretos, utilizar um banco de dados relacional permite rápida e facilmente responder consultas a respeito de onde são seus clientes, qual de seus produtos está vendendo melhor ou que tipo de clientes gastam mais.
O banco de dados que utilizaremos nesta seção é o MySQL.
Essa escolha não é ao acaso, pois sabemos que existem grandes sistemas de banco de dados espalhados no mercado.
Mas como o PHP pertence à comunidade livre, fica claro que seus desenvolvedores tenham também a preferência por um sistema de banco de dados livre.
Mas você não precisa cruzar os braços e ficar parado só porque não conhece o MySQL e trabalha com Oracle ou MSSQL. O PHP também tem suporte para esses bancos de dados.

O que é MySQL?

MySQL é um sistema de gerenciamento de banco de dados relacional, multiencadeado, de código-fonte aberto e nível corporativo.
O MySQL foi desenvolvido por uma empresa de consultoria na Suécia chamada inicialmente de TcX, depois, com a popularidade do MySQL, passou a se chamar MySQL AB.
Seu desenvolvimento ocorreu quando estavam precisando de um sistema de banco de dados que fosse extremamente rápido e flexível. Foi, assim então, que eles criaram o MySQL, que é vagamente baseado em outro sistema de gerenciamento de banco de dados chamado de mSQL.
O MySQL é rápido, flexível e confiável. É utilizado em muitos lugares por todo o mundo.

Obs.: A propósito, à parte "AB" do nome da companhia é o acrônimo para a palavra sueca "aktiebolag", ou "sociedade anônima". Ela é traduzida para "MySQL, Inc." De fato, MySQL Inc. e MySQL GmbH são exemplos de subsidiárias da MySQL AB. Elas estão localizadas nos EUA e Alemanha, respectivamente.

O que é um banco de dados relacional?

Um banco no mundo de cimento e tijolo é o lugar onde guardamos dinheiro. Um banco de dados também guarda, só que neste caso são dados.

Chamamos de dados tudo que possamos inserir no computador, números, letras, caracteres, imagens e etc.
Um banco de dados relacional é uma composição de tabelas e colunas que se relacionam entre si. Esses relacionamentos são baseados em um valor-chave que é contido em cada tabela, em uma coluna.

Instalando o banco de dados

O MySQL tem diferentes formas de instalação quando se trata de sistemas operacionais. No caso do Windows, você pode baixar a última distribuição através do site:

http://www.mysql.com/downloads.

Instalando no Windows

Procure pelo formato executável. O arquivo vem compactado no formato .**zip**.

Descompacte e instale. A instalação, como não poderia deixar de ser, é feita por um assistente. Siga os passos até a finalização.

Caso sua máquina tenha o sistema operacional Windows pertencente a família NT(NT, 2000 ou XP), o MySQL é instalado como serviço. Então basta iniciar ou parar o serviço, encontrado no **Painel de Controle>Ferramentas Administrativas>Serviços**.
Você também pode utilizar o comando pelo prompt, desde que você saiba o nome do serviço do seu MySQL:
Para iniciar o serviço
net start mysql
Para parar o serviço
net stop mysql

Instalando o MySQL no Linux

O MySQL Server pode ser instalado no Linux de várias formas. A forma recomendada é a que está em formato RPM.
Você deve baixar dois arquivos para instalar o MySQL na sua máquina. Esses arquivos são:

MySQL-server-[versão].i386.rpm – para instalar o servidor mysqld no Linux
MySQL-client-[versão].i386.rpm – para instalar o cliente mysql para executar os comandos no Linux.

A instalação poderá ser feita através do comando rpm, no Shell do seu Linux. Um exemplo seria:

Shell> rpm –ivh MySQL-server-5.0.1.i386.rpm MySQL-client-5.0.1.i386.rpm

A versão RPM já vem com pré-configurações e assim que ocorrer a instalação, para iniciar ou parar o servidor, a seguinte sintaxe poderá ser feita:

Shell>/etc/init.d/./mysql start – para iniciar o servidor MySQL
Shell>/etc/init.d/./mysql stop – para parar o servidor MySQL

Acessando o banco de dados MySQL

No Windows

Se você estiver usando o sistema operacional Windows e utilizou a instalação padrão do programa, abra o prompt de comando e digite a seqüência:

```
cd\mysql\bin
```

Lembrando que você deve estar no drive em que o MySQL está instalado. Por padrão você o instala no drive **C**.
Digitando o comando a seguir você entra no MySQL.

```
mysql –u root -p
```

Tecle ENTER e receberá o pedido de senha:
password
Digite a senha que você configurou na instalação e tecle ENTER novamente.

Nota: Versões mais modernas do MySQL para o sistema operacional Windows não necessitam de tantos passos para iniciar, bastando ir até o atalho encontrado no menu Iniciar do sistema e no atalho do MySQL iniciar o prompt de comando encontrado neste local.

No Linux

Se você utilizou a instalação binária, em rpm (recomendado), basta abrir o terminal e digitar a seqüência:

```
shell>mysql –u root
```

Se já estiver logado como **root**, no seu sistema operacional, não há necessidade de colocar o **–u root** depois do comando **mysql**.

Os comandos CREATE e DROP

Quando pensar nos comandos **CREATE** e **DROP**, você deve imaginar equipamentos de terraplanagem, caminhões basculantes e guindastes, porque são ferramentas que você utiliza para criar o seu banco de dados. Esses comandos, embora raramente utilizados, são os mais importantes.

O comando CREATE

Há muitas maneiras diferentes de criar banco de dados no MySQL. Ao criar um banco de dados, você normalmente terá o layout inteiro pronto. Normalmente adicionaria as tabelas imediatamente depois de criar o banco de dado, mas, teremos uma etapa por vez. A primeira etapa para criar um banco de dados no MySQL é inserir o comando ***CREATE DATABASE nome_banco_de_dados*** da SQL (Structured Query Language) no monitor MySQL, onde nome_banco_de_dados é o nome do banco de dados que você está criado.

No prompt de comando, no monitor do MySQL, insira o seguinte comando:

```
mysql> CREATE DATABASE livraria;
```

Note que não foi utilizado acentuação e em casos de palavras compostas não insira espaços, se for o caso insira sublinhado **“ _ ”** .

O comando USE

Depois de confirmado a criação do banco de dados, você deverá utilizar o comando **USE** para utilizar o banco de dados **livraria**.

```
USE livraria;
```

Um ponto importante é que o MySQL não torna ativo o banco de dados que você criou, isso deve ser implícito.

O comando DROP

O comando **DROP** é semelhante ao comando **CREATE**. Enquanto o último cria um banco de dados, o primeiro exclui. O comando **DROP** do **SQL** é imperdoável. Não há caixas de confirmação para ver se você tem certeza. Este comando exclui o banco de dados e tudo o que estiver nele.

É só ir até o prompt de comando e no monitor do MySQL e digitar:

```
mysql> DROP DATABASE livraria;
```

Isso excluirá o banco de dados veículos e tudo o que estiver nele.

Criando tabelas

Criar tabela no MySQL é uma tarefa relativamente fácil. Para se criar uma tabela basta usar a seqüência:

shell>mysql –u root

Após estar no monitor do MySQL digite a seguinte seqüência:

```
mysql> CREATE DATABASE livraria;

mysql> USE livraria;

mysql> CREATE TABLE autores(
       -> autor_id INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
       -> nome VARCHAR(100),
       -> dt_nasc DATE);

mysql> CREATE TABLE editora(
       -> editora_id INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
       -> editora_nome VARCHAR(100)
       -> );

mysql> CREATE TABLE publicacao(
      -> isbn CHAR(13),
      -> autor_id INT UNSIGNED,
      -> editora_id INT UNSIGNED
      -> );

mysql> CREATE TABLE  livros(
       -> isbn CHAR(13) NOT NULL PRIMARY KEY,
       -> titulo VARCHAR(50),
       -> edicao_num TINYINT(2),
       -> ano_publicacao YEAR,
       -> descricao TEXT);
```

O comando SHOW

Assim que criada sua primeira tabela. Para ver o resultado basta digitar a seqüência:

```
SHOW TABLES FROM livraria;
```

Para ver as colunas que existem na sua tabela digite:

```
SHOW COLUMNS FROM livros;
```

Ou **DESCRIBE**.

O comando DESCRIBE

Se preferir, o comando **DESCRIBE** faz a mesma coisa que **SHOW**, mostrando as colunas existentes em sua tabela.

```
DESCRIBE livros;
```

Ou simplesmente:

```
DESC livros;
```

IF NOT EXISTS

Uma maneira de se ter certeza de se não está criando uma tabela novamente é fazer o comando **IF NOT EXISTS**:

```
mysql> CREATE TABLE  IF NOT EXISTS livros(
    -> isbn CHAR(13) NOT NULL PRIMARY KEY,
    -> titulo VARCHAR(50),
    -> edicao_num TINYINT(2),
    -> ano_publicacao YEAR,
    -> descricao TEXT);
```

Criando uma cópia de uma tabela

A partir da versão 4.1 você pode copiar uma tabela com sua estrutura da seguinte maneira:

```
CREATE TABLE copia_livros LIKE livros;
```

Alterando tabelas existentes

Agora que você criou a sua tabela o que aconteceria se você precisasse alterar algo que fez?
Confira os seguintes exemplos para alterar o nome da tabela, tipo de dados e o nome da coluna:

Alterando o nome da coluna

```
ALTER TABLE copia_livros CHANGE titulo titulo_do_livro VARCHAR(50);
```

DESC copia_livros; *# descreva as colunas de clientes*

Nota: # (sustenido) é o inicio de um comentário e não interfere em uma execução de comando.

Alterando o tipo de coluna

```
mysql>  ALTER TABLE copia_livros
    ->  MODIFY  titulo_do_livro VARCHAR(30) NOT NULL;
```

Renomeando uma tabela

```
ALTER TABLE copia_livros RENAME livros2;
```

Excluindo / adicionando colunas e tabelas

Como você pode ver quando uma coluna é criada ou uma tabela estas não são escritas na pedra e podem ser alteradas facilmente. Isso também implica em adicionar colunas em uma tabela existente ou excluí-la.

Eliminando tabelas e colunas

O comando **DROP** também é utilizado para eliminar as colunas de uma tabela.
Para excluir uma tabela existente execute a seguinte seqüência:

```
DROP TABLE livros2;
```

Para excluir somente uma coluna execute a seguinte seqüência:

```
ALTER TABLE livros2 DROP editora_id;
```

Isso excluirá a coluna e todas as informações que você armazenou.

Adicionando colunas

O comando **ADD** é o responsável pela inserção de uma nova coluna.

```
ALTER TABLE livros2 ADD editora_id INT; #onde ADD é adicionar
```

Adicionando colunas após uma outra determinada

O comando **AFTER** adiciona a nova coluna na tabela após o nome mencionado.

```
ALTER TABLE livros2 ADD editora_id INT AFTER edicao_num;
```

Utilizando índices

Um índice é um arquivo estruturado que facilita o acesso a dados.
Isso significa que um índice na coluna correta aumentará a velocidade de uma consulta consideravelmente. Um índice trabalha da mesma forma que pastas com separador alfabético em um gabinete de arquivo ele permite pular para a parte do alfabeto que você está procurando.

Decidindo quais colunas incluir no índice

Você deve colocar um índice na(s) coluna(s) que utilizará com mais freqüência como filtro em suas consultas.
Os índices também funcionam melhor em colunas que contêm dados únicos. Essa é uma das razões pela as quais chaves são normalmente suas melhores escolhas para índices. Essa também pode ser uma das razões que as pessoas confundem chaves e índices. Uma chave ajuda a definir a estrutura de um banco de dados, ao passo que índice apenas aprimora o desempenho.
Um índice pode ser composto de uma ou mais colunas. Você também pode ter mais de um índice em uma tabela.

Criando um índice

Por padrão, o MySQL cria um índice se você declara uma coluna como uma chave primária. Não há necessidade de criar um índice nessa coluna; caso contrário você teria dois índices em uma mesma coluna.
A sintaxe para criar um índice em uma coluna:

```
ALTER TABLE livros ADD INDEX  idx_titulo(titulo);
```

Excluindo índices

Excluir um índice é tão simples quanto criar. A sintaxe é a mesma que excluir uma coluna ou uma tabela:

```
DROP INDEX  nomedoindice ON nomedatabela;
```

Ou...

```
ALTER TABLE nomedatabela DROP INDEX nomedoindice;
```

Para alterar uma tabela eliminando uma chave primária, utilize a seguinte sintaxe:

```
ALTER TABLE nomedatabela DROP PRIMARY KEY;
```

Nota: Se você estiver usando uma coluna com **AUTO_INCREMENT**, você não excluirá a chave primária enquanto não retirar esse modificador.

Tipos de tabelas

O MySQL possui uma característica um pouco diferente dos outros sistemas gerenciadores de banco de dados, uma vez que no MySQL é possível escolher o tipo da tabela no momento da criação. O formato de armazenamento dos dados, bem como alguns recursos do banco de dados são dependentes do tipo de tabela escolhido.

A definição do tipo de tabela pode ser feita na criação da tabela, como você pode ver a seguir:

```
CREATE TABLE teste (
      id INT NOT NULL,
      nome VARCHAR(30) NOT NULL,
      PRIMARY KEY (id)
) TYPE=MyISAM;
```

No comando criado, o tipo da tabela, indicado em TYPE=MyISAM, significa que você está criando uma tabela com o tipo MyISAM, que é o padrão das tabelas, caso não seja informado o TYPE. A partir da versão 4.0.18 você pode utilizar ENGINE como sinônimo de TYPE.
A seguir você tem alguns dos tipos de tabelas no qual você pode criar:

O tipo MyISAM

Este é o tipo de tabela padrão do MySQL. Caso não seja informado o tipo de tabela, o MySQL criará a tabela do tipo MyISAM. O tipo de tabela padrão pode ser alterado incluindo-se no arquivo de configuração, chamado de **my.cnf (**no Linux**)** ou **my.ini (**no Windows**)**, a opção a seguir:

```
default-storage-engine=INNODB
```

As tabelas MyISAM são armazenadas em 3 arquivos, com o mesmo nome da tabela, mas com extensões diferentes:

- **.FRM** que armazena a definição da tabela.
- **.MYD** que contém os dados.
- **.MYI** contendo os índices.

Estas tabelas são de grande desempenho para leitura, uma vez que os seus índices são armazenados em árvores binárias balanceadas, o que provê um ganho para o acesso às informações. O MyISAM não trabalha com transações (commit ou rollback) e também não possui integridade referencial, isto é, ao incluir uma chave estrangeira com alguns constraints, esta servirá apenas como documentação, mas as restrições não serão respeitadas pelo banco.

O tipo Memory

Tabelas do tipo **MEMORY** (conhecida anteriormente como **HEAP**) são armazenadas em memória e, graças a isso, são extremamente rápidas. Em contrapartida, o seu conteúdo é volátil, uma vez que não são gravadas em disco. Caso haja uma queda do SGBD os dados destas tabelas serão perdidos. Além disto, é necessário um processo para dar a carga inicial nos dados quando o servidor de banco iniciar e sua execução. A principal aplicação das tabelas MEMORY seria para tabelas que são consultadas com muita freqüência, mas que não sofrem muitas alterações (**lookup tables**).

O tipo MERGE

As tabelas do tipo **MERGE** são coleções de tabelas **MyISAM** idênticas. Este recurso permite a divisão de uma tabela grande em várias partes menores, e ainda assim permite acesso ao conteúdo de todas elas como se fossem uma única tabela. Veja um exemplo de como utilizá-la:

```
CREATE TABLE exemplo1 (
      id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      exemplo VARCHAR(20)
);

CREATE TABLE exemplo2 (
      id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      exemplo VARCHAR(20)
);
```

```
INSERT INTO exemplo1  VALUES (null, 'Exemplo1'),(null,‘Teste1');
INSERT INTO exemplo2  VALUES (null, 'Exemplo2'),(null, 'Teste2');

CREATE TABLE mesclar (
      ids INT NOT NULL AUTO_INCREMENT,
      exemplos VARCHAR(20), INDEX(ids)
)TYPE=MERGE UNION=(exemplo1,exemplo2);

SELECT * FROM mesclar;
```

ids	Exemplos
1	Exemplo1
2	Teste1
1	Exemplo2
2	Teste2

Resultado da Mesclagem

O tipo BDB

O tipo de tabela **BDB** vem de **BerkeleyDB**, e é desenvolvido pela Sleepycat (**http://www.sleepycat.com**). O tipo BDB provê ao MySQL um manipulador de tabelas com controle de transação, dando assim a você a possibilidade de usar os comandos COMMIT e ROLLBACK, além de fornecer a recuperação automática de dados em caso de queda do sistema. O BDB apresenta um mecanismo de lock em nível de página, onde apenas os dados de uma mesma página ficarão bloqueados durante um período de lock.

O tipo InnoDB

O tipo **InnoDB** é do tipo de tabela transacional, desenvolvido pela **InnoDBase Oy**. A partir da versão 4.0 do MySQL ele passa a ser parte integrante das distribuições do MySQL. O InnoDB apresenta, além da capacidade transacional, outros recursos que são realmente úteis na utilização de tabelas:

- Integridade referencial, com implementação dos constraints **SET NULL**, **SET DEFAULT**, **RESTRICT** e **CASCADE**;
- Ferramenta de backup on-line (ferramenta comercial, não GPL);
- Lock de registro, como Oracle, DB2, etc;
- Níveis de isolamento;
- Armazenamentos de dados em tablespace.

Por se tratar de um tipo de tabela com recursos mais avançados, requer mais espaço em memória e disco, além de se apresentar, em determinadas situações, um pouco mais lento que tabelas do tipo MyISAM. Apesar disto, o tipo InnoDB tem se mostrado extremamente rápido se comparado com outros SGBDs transacionais.

Alterando o tipo de uma tabela

Com o comando **ALTER TABLE** não é possível alterar o tipo da tabela, por isso, você pode alterar da seguinte maneira:

```
ALTER TABLE livros ENGINE=INNODB;
```

Tipo de dados

Como a maioria dos sistemas de gerenciamento de banco de dados relacional (Relational Database Management Systems – RDBMS), o MySQL tem tipos de dados específicos de coluna.
O MySQL tem vários tipos de dados que suportam funções diferentes. Um tipo de dados é a definição das informações que uma coluna armazenará. Pode haver muitos tipos de dados em uma tabela, mas cada coluna armazenará seu próprio tipo de informações específicas.
Há quatro tipos de grupos de formatos de dados. O primeiro é o numérico. O segundo tipo é o formato de caractere ou string. Esse formato consiste em letras e números ou qualquer coisa que você coloque entre aspas. O terceiro grupo é formado por datas e horas. O tipo final é uma forma de miscelânea. Ele consiste em tudo que não se encaixa em qualquer uma das outras categorias.

Tipos numéricos

Os tipos numéricos destinam-se somente a números. Os diferentes tipos de números ocupam uma quantidade diferente de espaço na memória.
Um bom exemplo é você tentando comprar um chocolate em uma loja e ao passar no caixa a você descobre que deve pagar pela caixa inteira. Você diz que não precisa de tudo, mas é atacado e só e vendido de caixa. Se você vai utilizar 3 números, por que ocupar um espaço na memória como se estivesse utilizando 100?
Lembre-se: você só deve pagar pelo que vai usar.

Armazenamento numérico

Nome do tipo	Espaço na memória
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
BIGINT	8 bytes
FLOAT(Inteiro,Decimal)	4 bytes
DOUBLE(Inteiro,Decimal)	8 bytes
DECIMAL(Inteiro,Decimal)	O valor de bytes Inteiro + 2

Se a coluna é numérica e declarada **UNSIGNED**, o intervalo dobra para o tipo dado. Por exemplo, se você declara que uma coluna que é ***UNSIGNED TINYINT***, o intervalo dessa coluna é de **0** a **255**. Declarando dessa forma você faz com que essa coluna tenha somente valores positivos.

Tipos numéricos

Nome do tipo	Intervalo de valor	Sem sinal
TINYINT	-128 a 127	0 – 255
SMALLINT	-32768 a 32767	0 – 65535
MEDIUMINT	-8388608 a 83888607	0 - 16777215
INT	-2147483648 a 2147483647	0 - 4294967295
BIGINT	-9223372036854775808 a 9223372036854775807	0 - 18446744073709550615
FLOAT(Inteiro,Decimal)	Varia dependendo dos valores	
DOUBLE(Inteiro,Decimal)	Varia dependendo dos valores	
DECIMAL(Inteiro,Decimal)	Varia dependendo dos valores	

FLOATs, **DOUBLE**s e **DECIMAL**s são tipos numéricos que podem armazenar frações. Os outros não.

Utilize **DECIMAL** para números realmente grandes. **DECIMALs** são armazenados de maneira diferente e não têm limites.

Modificadores AUTO_INCREMENT, UNSIGNED e ZEROFILL

Esses modificadores só podem ser utilizados com tipos de dados numéricos. Eles utilizam operações que somente podem ser feitas com números.

AUTO_INCREMENT

O modificador de coluna **AUTO_INCREMENT** automaticamente aumenta o valor de uma coluna adicionando 1 ao valor máximo atual. Ele fornece um contador que é ótimo para criar valores únicos.
Você também pode incluir um número. Se quiser que uma coluna **AUTO_INCREMENT** inicie com **9.000**, por exemplo, é só declarar explicitamente um ponto inicial utilizando a seguinte sintaxe:

```
mysql> CREATE TABLE  teste (
    -> id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
    -> ) AUTO_INCREMENT=9000;
```

UNSIGNED

UNSIGNED depois de um tipo inteiro significa que ele só pode ter um zero ou valor positivo.

ZEROFILL

O modificador de coluna **ZEROFILL** é utilizado para exibir zeros à esquerda de um número com base na largura de exibição.
Como todos os tipos de dados numéricos têm uma largura de exibição opcional, se você declara um **INT(8) ZEROFILL** e o valor armazenado é **23**, ele será exibido como **00000023**.
Para isso utilize a seguinte sintaxe:

```
mysql>CREATE TABLE teste (id INT(4) ZEROFILL);
```

Tipos de caractere ou de dados de string

O outro grupo importante de tipo de dados são os tipos de strings ou de caractere.
Uma string é um conjunto de caracteres. Um tipo de string pode armazenar dados como São Paulo ou Avenida São João, n.º 255. Qualquer valor pode ser armazenado em um tipo de dados de string.

Tipos string

Nome de tipo	Tamanho máximo	Espaço de armazenamento
CHAR(X)	255 bytes	X bytes
VARCHAR(X)	255 bytes	X + 1 byte
TINYTEXT	255 bytes	X + 1 byte
TINYBLOB	255 bytes	X + 2 bytes
TEXT	65.535 bytes	X + 2 bytes
BLOB	65.535 bytes	X + 2 bytes
MEDIUMTEXT	1,6 MB	X + 3 bytes
MEDIUMBLOB	1,6 MB	X + 3 bytes
LONGTEXT	4,2 GB	X + 4 bytes
LONGBLOB	4,2 GB	X + 4 bytes

CHAR e VARCHAR

Fora todos esses tipos, os tipos **VARCHAR** e **CHAR** são os mais utilizados. A diferença entre eles é que o **VARCHAR** tem um comprimento variável e o **CHAR** não. Os tipos **CHAR** são utilizados para comprimentos fixos. Você utilizará esse tipo quando os valores não variam muito. Se você declara um **CHAR(10)**, todos os valores armazenados nessa coluna terão **10 bytes** de comprimento, mesmo se ele tiver **3 bytes** de comprimento. O MySQL preenche esse valor para ajustar o tamanho que foi declarado. O tipo **VARCHAR** faz o contrário. Se você declara um **VARCHAR(10)** e armazena um valor que tem somente **3** caracteres de comprimento, a quantidade total de espaço de armazenamento é de **4 bytes** (o comprimento mais um).
A vantagem de utilizar os tipos **CHAR** é que as tabelas que contêm esses valores fixos são processadas mais rapidamente que aquelas que são compostas pelo tipo **VARCHAR**. A desvantagem de utilizar o tipo **CHAR** é o espaço desperdiçado.
De um modo geral não se pode utilizar os dois na mesma tabela, pois quando feito o MySQL converte automaticamente uma coluna com o tipo **CHAR** em **VARCHAR**.
A única exceção é quando você declara uma coluna como **VARCHAR(3)**, o MySQL converte automaticamente em **CHAR(3)**. Isso acontece porque valores de **4** caracteres ou menores são muito pequenos para o tipo **VARCHAR**.

TEXT e BLOB

TEXT e **BLOB*(Binary Large Object)*** são tipos variáveis de comprimento que podem armazenar grandes quantidades de dados. Você utilizará esses tipos quando quiser armazenar *imagens, sons* ou *grandes* quantidades de *textos*, como páginas da *Web* ou

documentos.
Um bom exemplo é se você estiver querendo armazenar valores de uma <*TEXTAREA*> de uma sessão de comentários em uma página da Web, o tipo **TEXT** seria uma boa escolha.

Tipos variados

Há basicamente três tipos variados; os tipos *ENUM, SET e DATE/TIME*.

Tipo ENUM

O tipo **ENUM** é uma lista ENUMerada. Significa que essa coluna pode armazenar apenas um dos valores que estão declarados na lista dada.
A sintaxe para criar uma coluna ENUMerada é como segue:

```
mysql> CREATE TABLE coluna_enum (
    -> estados ENUM('SP','RJ','MG','RS')
    -> );
```

Você pode ter até *65.535 itens* em sua lista enumerada. É uma boa escolha para caixas de combinação.

Tipo SET

O tipo **SET** é muito parecido com o tipo ENUM. O tipo SET, como o tipo ENUM, armazena uma lista de valores. A diferença é que no tipo SET, você pode escolher mais de uma opção para armazenar. Um tipo SET pode conter até *64 itens*. O tipo SET é uma boa escolha para opções em uma página da Web em que o usuário pode escolher mais de um valor.

A sintaxe para criar um tipo SET:

```
mysql>CREATE TABLE teste (
    -> passatempo SET('Televisão','Futebol','Vídeo Game','Cinema')
    -> );
```

A coluna criada seria capaz de armazenar uma ou mais escolhas, como por exemplo:
"Futebol" #como uma escolha
"Futebol, Vídeo Game" # como mais de uma escolha.
Quando se inserem os valores em uma coluna do tipo SET, se coloca entre aspas ou apóstrofos e separados por vírgula:

```
mysql>INSERT INTO teste (passatempo) VALUES ('Futebol,Vídeo Game');
```

Modificadores adicionais de coluna

O MySQL tem várias palavras-chave que modificam a maneira como uma coluna funciona.
Como vimos acima, temos **AUTO_INCREMENT** e **ZEROFILL** e como eles afetam a coluna em que são utilizados. Alguns modificadores se aplicam apenas em colunas de um certo tipo de dado.

Tabela de Modificadores

Nome de modificador	Tipos aplicáveis
AUTO_INCREMENT	Todos os tipos INT
BINARY	CHAR, VARCHAR
DEFAULT	Todos, exceto BLOB, TEXT
NOT NULL	Todos os tipos
NULL	Todos os tipos
PRIMARY KEY	Todos os tipos
UNIQUE	Todos os tipos
UNSIGNED	Tipos numéricos
ZEROFILL	Tipos numéricos

O modificador **BINARY** faz com que os valores armazenados sejam tratados como strings binárias, fazendo-os distinguir letras maiúsculas e minúsculas. Ao classificar ou comparar essas strings, a distinção entre maiúsculas e minúsculas será considerada.
Por padrão os tipos **CHAR** e **VARCHAR** não são armazenados como binários.
O modificador **DEFAULT** permite especificar o valor de uma coluna se não existir um valor.
Os modificadores **NULL** e **NOT NULL** especifica se na coluna deve haver um valor ou não.
Por exemplo; se você especificar a coluna como **NOT NULL** você é forçado a colocar um valor, pois esse campo é requerido.
PRIMARY KEY é um índice que não deve conter valores nulos (NULL). Cada tabela deve conter uma chave primária, isso facilita uma consulta de dados. Abordarei essa questão mais adiante.
O modificador **UNIQUE** impõe a regra que todos os dados dentro da coluna declarada devem ser únicos.

Por exemplo; se você declarar (não faça isso) que a coluna nome deve ser *UNIQUE*(única), não pode haver valores duplicados, caso contrário gerará um erro.
A sintaxe para sua criação é:

```
mysql>CREATE TABLE IF NOT EXISTS teste (nome VARCHAR(30)UNIQUE NOT NULL);
```

Tipos de data e hora

O MySQL suporta vários tipos de data e hora. Esses são mostrados na tabela a seguir:

Tabela de data e hora

Tipo	Intervalo	Descrição
DATE	1000-01-01 a 9999-12-31	Datas. Será exibida como YYYY-MM-DD
TIME	-838:59:59 a 838:59:59	Hora. Será exibida como HH:MM:SS
DATETIME	1000-01-01 00:00:00 a 9999-12-31 23:59:59	Data e Hora. Será exibida como YYYY-MM-DD HH:MM:SS
TIMESTAMP[(F)]	1970-01-01 00:00:00	Um registro de data/hora, útil para relatório de transação. O formato de exibição depende do formato de F.
YEAR[(2 | 4)]	70-69 (1970-2069) 1901-2155	Um ano. Você pode especificar 2 ou 4 formatos de dígitos. Cada um desses tem um intervalo diferente, como mostrado.

Sintaxe básica da SQL

A primeira coisa que devemos fazer quando criamos um banco de dados e depois uma tabela e utilizá-la inserindo dados.

Comando INSERT

O comando **INSERT INTO** adiciona dados em uma tabela.
A sua sintaxe é:

```
mysql> INSERT INTO livros VALUES (
    -> '85-7585-120-5',
    ->'Core Java Fundamental',
    ->6,
    ->‘2004’,
    ->'Desenvolva Java com vários exemplos');
```

O nome da tabela em que você irá inserir deverá ser declarada logo no início INSIRA DENTRO nomedatabela (colunas) VALORES (‘valores inseridos dentro de cada coluna’);
É importante salientar que strings ficam entre aspas ou apóstrofos e valores numéricos (declarados como tipo de dados numéricos) não precisam de “aspas” ou ‘apóstrofos’.
Para inserir mais de um valor separe-os por vírgula:

```
mysql> INSERT INTO nomedatabela(colunas) VALUES (‘valores inseridos 1’),
    ->(‘valores   inseridos 2’),
    ->(‘e assim por diante’);
```

Comando SELECT

A instrução SELECT é provavelmente a mais utilizada de todas as instruções de SQL. A instrução SELECT somente retornará os dados que são armazenados no banco de dados dentro de uma tabela. O MySQL realiza essa instrução mais rápido que qualquer outro banco de dados do mercado.
A sintaxe é:

SELECT nomedacoluna FROM nomedatabela WHERE condições;

No caso do nosso banco de dados livraria:

```
mysql>SELECT * FROM livros; # o asterisco indica  todas as colunas
```

Um outro caso, a cláusula WHERE

```
mysql>SELECT * FROM livros WHERE ISBN='85-7585-120-5';
```

Nesse caso foi colocada uma condição que dentre todos os registros só deverá aparecer os dados *ONDE* a coluna **ISBN** for igual à **'85-7585-120-5'**.
A cláusula **WHERE** especifica o critério utilizado para selecionar linhas particulares. O único sinal igual é utilizado para testar igualdade – observe que isso é diferente do Java e é fácil se confundir.
Além da igualdade, o MySQL suporta um conjunto completo de operadores e expressões regulares. Na tabela a seguir estão listadas as mais utilizadas por você:

Tabela de Operadores no MySQL			
Operador	**Nome**	**Exemplos**	**Descrição**
=	igual à	autor_id = 1	Testa se os dois valores são iguais
>	maior que	Quantidade > 50	Testa se um valor é maior que o outro
<	menor que	Quantidade < 50	Testa se um valor é menor que o outro
>=	maior ou igual a	Quantidade >= 50	Testa se um valor é maior ou igual ao outro
<=	menor ou igual a	Quantidade <= 50	Testa se um valor é menor ou igual ao outro
!= ou <>	diferente de	Quantidade !=0	Testa se um valor é diferente do outro
IN		cidade in (‘São Paulo’, ‘Minas Gerais’)	Testa se o valor está em um conjunto particular
NOT IN		cidade not in (‘São Paulo’, ‘Minas Gerais’)	Testa se o valor não está em um conjunto particular
IS NOT	Endereço não é nulo		
IS NULL	Endereço é nulo	promocao is null	Testa se o campo não contém um valor
BETWEEN	Quantidade	valor BETWEEN 200 AND 350	Testa se o campo tem

	entre um valor e outro		valores entre um e outro

Algumas funções que trabalham com a instrução SELECT

MAX( )

SELECT MAX(coluna) FROM tabela;

Essa função seleciona o valor máximo de uma coluna.

MIN( )

SELECT MIN(coluna) FROM tabela;

O contrário de MAX, retorna o valor mínimo de uma coluna.

LIMIT

*SELECT * FROM tabela LIMIT 2;*

Limita a visualização de 2 linhas de dados.

*SELECT * FROM tabela LIMIT 2,5;*

Limita a visualização da linha 2 a linha 5 de dados.

COUNT( )

SELECT COUNT(coluna) FROM tabela;

Conta quantas linhas de dados existem na coluna nome.

SELECT COUNT(*) FROM tabela;

Conta quantas linhas de dados existem em todas as linhas.

Nota: Em caso de fazer a contagem em campo de valor **NULL** a contagem será diferente da no valor total.

SUM( )

SELECT SUM(coluna) FROM tabela;

Soma todos os dados da coluna.

ORDER BY

*SELECT * FROM tabela ORDER BY coluna;*

Coloca os dados selecionados em ordem crescente pela coluna.

ORDER BY ... DESC

*SELECT * FROM tabela ORDER BY coluna DESC;*

Coloca os dados selecionados em ordem decrescente pela coluna.

AVG( )

SELECT AVG(coluna) FROM tabela;

Faz a média aritmética da coluna designada.

LIKE

```
mysql>SELECT * FROM livros WHERE titulo LIKE 'Java%';
```

Neste caso pode-se fazer uma busca por apenas a inicial do valor desejado.

O sinal de %(porcentagem) é o caractere curinga que significa qualquer caractere.

```
mysql>SELECT * FROM livros WHERE titulo LIKE '%Java%';
```

Colocando a % no início e no fim, com um valor no meio, é possível buscar todos os valores que contenham as letras **Java**, seja no começo, meio ou fim.

Um caso a parte: a união do INSERT INTO ... SELECT

INSERT INTO tabela1(coluna) SELECT tabela.coluna2 FROM tabela2;

Insere na tabela tabela1.coluna valores da coluna2 da tabela tabela2.

Comando UPDATE

O comando UPDATE permite editar os valores de dados existentes. A sintaxe para

modificar os valores é:

UPDATE tabela SET coluna= 'valor' WHERE coluna='valor';

Atualiza os dados da coluna determinada em SET na condição passada em WHERE.

Comando DELETE

A instrução DELETE é muito semelhante á instrução SELECT. A única diferença em vez de selecionar registros para visualizar, essa instrução exclui esses registros.
A instrução DELETE tem a seguinte sintaxe:

DELETE FROM tabela WHERE coluna='valor';

Trabalhando com Junções

As junções são uma parte integrante de um banco de dados relacional. As junções permitem ao usuário de banco de dados tirar proveito dos relacionamentos que foram desenvolvidos na fase do projeto do banco de dados.
Uma *JUNÇÃO* é o termo utilizado para descrever o ato em que uma ou mais tabelas são "unidas" entre si para recuperar dados necessários com base nos relacionamentos que são compartilhados entre elas.

Criando uma junção com INNER JOIN

A seguinte sintaxe cria uma junção:

SELECT tabela1.coluna, tabela2.coluna FROM tabela1 INNER JOIN tabela2 on tabela1.coluna_de_valor_identico=tabela2.coluna_de_valor_identico;

INNER JOIN's são provavelmente as mais comuns de todas as junções.
Uma **INNER JOIN** significa que todos que todos os registros que estão sem correspondência são descartados. Somente as linhas correspondidas serão exibidas no conjunto de resultados. Os dados aparecem na ordem em que você especifica.

Chaves variadas do MySQL

O que é uma chave?

Uma chave em uma tabela em um banco de dados fornece um meio de localizar rapidamente informações específicas. Embora uma chave não precise significar qualquer coisa para o usuário humano do banco de dados, as chaves são uma parte vital da arquitetura de banco de dados e pode influenciar significativamente o desempenho.

Princípios da Chave

Imagine que você tem uma coleção muito simples de dados em que armazena apenas os dados "úteis" simples. Por exemplo, você talvez crie uma tabela de Clientes semelhante a um antigo arquivo de índice de fixas, com nome e detalhes de um cliente em cada ficha. Quando quiser pesquisar um cliente, você procura o arquivo e lê cada ficha sucessivamente. Ao ver a(s) ficha(s) que quer, você lê essas informações úteis – como o nome, endereço e número de telefone do cliente.

Convencionalmente, você talvez classifique o arquivo de índice de ficha por ordem de sobrenome. Isso ajuda se você sabe o nome da pessoa cujos dados está examinando. Mas e se você quiser localizar as pessoas por algum outro critério.
Naturalmente, você pode configurar o banco de dados MySQL da mesma maneira. Mas logo ficaria com dificuldades.
Com a quantidade de dados o seu banco de dados teria de ler as informações sucessivamente, o que torna uma operação ineficiente.
Você chamaria essa operação de varredura de tabela. Essa é a mais demorada das operações em um banco de dados.
É nesse momento em que uma chave se torna útil.

Como as chaves funcionam

Uma chave existe como uma tabela extra no banco de dados, embora pertença à sua tabela pai. Ela ocupa espaço físico no disco rígido (ou outras áreas de armazenamento) do banco de dados. Pode ser tão grande quanto a tabela principal e, teoricamente, até maior.
Você define a chave para se relacionar com uma ou várias colunas em uma tabela específica. Como os dados em uma chave são totalmente derivados da tabela, você pode eliminar e recriar uma chave sem qualquer perda de dados.

Benefícios de usar uma chave

A utilização adequada de chaves pode aprimorar significativamente o desempenho do banco de dados. Para utilizar a analogia de um índice de livro, considere o pouco número de páginas que é necessário no índice de um livro para dar visão rápida dos temas importantes. Compare quanto tempo você levaria se estivesse pesquisando pelo volume, página por página.

Suporte de chave do MySQL

O MySQL suporta os seguintes comandos para criar chaves nas tabelas existentes:

ALTER TABLE nome_tabela ADD (KEY | INDEX) nome_do_índice (nome_da_coluna [,...]);

ALTER TABLE nome_tabela ADD UNIQUE nome_do_índice (nome_da_coluna[,...]);

ALTER TABLE nome_tabela ADD PRIMARY KEY nome_do_índice (nome_da_coluna[,...]);

Observe que no MySQL, chave e índice são sinônimos.
Esses são os formatos preferidos para adicionar chaves a tabelas existentes. Para compatibilidade com outras implementações de SQL, o MySQL também suporta os seguintes:

CREATE INDEX nome_do_índice ON nome_tabela (nome_da_coluna[,...]);

CREATE UNIQUE INDEX [nome_do_índice] ON nome_tabela (nome_da_coluna[,...]);

CREATE PRIMARY KEY ON nome_tabela (nome_da_coluna,...);

Você pode definir as chaves quando cria uma tabela:

CREATE TABLE nome_da_tabela (nome_da_coluna tipo_de_campo [NULL | NOT NULL], KEY col_index (nome_da_coluna));

Chaves primárias

Uma chave primária é semelhante em princípio a uma chave única, seus dados devem ser únicos, mas a chave primária de uma tabela tem um status mais privilegiado. Apenas uma chave primária pode existir para cada tabela e seus valores de campo nunca podem ser nulos.
Uma chave primária é geralmente utilizada como um link estrutural no banco de dados, definindo o relacionamento entre as tabelas diferentes. Sempre que quiser unir uma tabela a outra, você deve ter a chave primária dessa tabela.
O MySQL não requer que você especifique que a coluna em que estiver a chave primária seja NOT NULL(não nula) *****, mas porém se tentar colocar um valor idêntico na coluna chave, esta retornará um erro que não pode haver duplicação.

***** Este caso é somente para chaves primárias em tabelas cuja coluna selecionada seja INT ou semelhante. Em casos de ser VARCHAR, CHAR e etc, é exigida a utilização do NOT NULL. Caso isso não ocorra, você terá como resultado um erro. Se desejar que seja nulo o campo, coloque uma chave UNIQUE.

Chaves estrangeiras

As chaves estrangeiras são atualmente suportadas no MySQL em outro formato de tabela. A mais usada e recomenda para transações é chamada de InnoDB.
A seguir você tem o comando necessário para criar uma chave estrangeira no seu banco de dados:

```
ALTER TABLE publicacao ADD CONSTRAINT FK_publicacao
FOREIGN KEY (isbn) REFERENCES
livros (isbn) ON DELETE CASCADE ON UPDATE CASCADE;
```

Excluindo uma chave estrangeira

Para excluir uma chave estrangeira, use o comando:

```
ALTER TABLE publicacao DROP FOREIGN KEY FK_publicacao;
```

Transações

Transações são mecanismos que asseguram consistência no banco de dados, especialmente no caso de erro ou queda no servidor.

Usando transações no MySQL

Por padrão, o MySQL na versão atual roda em **autocommit mode**. Isto significa que cada declaração que você executa é escrita imediatamente ao banco de dados (**commit**). Se você desejar trabalhar com transações no MySQL, você terá que executar o comando:

```
set autocommit=0;
```

Inicie a transação com o comando:

```
start transaction;
```

Para ilustrar uma transação, você vai executar uma EXCLUSÃO na tabela LIVROS:

```
DELETE FROM livros;
```

Dê um SELECT para ter certeza de que os dados foram todos excluídos da tabela:

```
SELECT * FROM livros;
```

Após confirmar que os dados foram realmente excluídos, execute o comando:

```
rollback;
```

O comando **ROLLBACK** retorna o estado anterior. Mas se você desejar confirmar o comando, utilize o comando a seguir:

```
commit;
```

Stored Procedures

Stored Procedures (procedimentos armazenados) passou a ser suportado pelo MySQL na versão 5. Agora, além do poder da velocidade desse incrível banco de dados, você também pode armazenar rotinas e chamá-las quando necessitar.
As stored procedures pode consistir em declarações de SQL e várias estruturas de controle especiais. Isto pode ser útil quando você quiser executar a mesma função de aplicações diferentes ou plataformas, ou como um modo de encapsular funcionalidade.

Criando um Stored Procedure com resultados de um JOIN

Aqui você aprenderá a criar um stored procedure simples para executar uma query com join entre as tabelas **livros, publicacao, autores** e **editora**:

```
DELIMITER //
CREATE PROCEDURE sp_m_liv_e_aut( )
BEGIN
  SELECT nome as `Nome do Autor`, titulo as `Título`,
     ano_publicacao as `Ano Publicado`, editora_nome as Editora
     FROM autores
     INNER JOIN publicacao USING (autor_id, autor_id)
     INNER JOIN livros USING (isbn, isbn)
     INNER JOIN editora USING (editora_id, editora_id);
END
//
DELIMITER ;
```

Para chamar a procedure, utilize o comando:

```
call  sp_m_liv_e_aut( );
```

Visualizando procedures criadas

Para visualizar as Stored Procedures criadas, execute o comando a seguir:

```
SHOW PROCEDURE STATUS;
```

Visualizando a criação da procedure

Para visualizar o código da Stored procedure, execute o comando:

```
SHOW CREATE PROCEDURE sp_m_liv;
```

Criando um Stored Procedure com parâmetros

A idéia agora é transmitir um parâmetro para que você possa filtrar os livros, chamando apenas uma procedure com o ISBN do livro:

```
DELIMITER //
CREATE PROCEDURE sp_m_liv_isbn(IN vIsbn VARCHAR(100))
BEGIN
  SELECT * FROM livros WHERE isbn=vIsbn;
END
//
DELIMITER ;
```

Para chamar essa procedure, execute o comando a seguir:

```
call  sp_m_liv_isbn('85-7393-436-0');
```

Criando um procedure com a cláusula LIKE

O stored procedure seguinte demonstra como usar a cláusula LIKE para retornar valores do banco de dados MySQL:

```
DELIMITER $$
CREATE PROCEDURE sp_m_liv(IN vTitulo VARCHAR(100))
BEGIN
  SELECT * FROM livros WHERE titulo LIKE CONCAT('%',vTitulo,'%')
  ORDER By titulo;
END $$
DELIMITER;
```

Para chamar essa procedure, execute o comando a seguir:

```
call  sp_m_liv('JBuilder');
```

Para se excluir uma stored procedure execute o seguinte comando:

```
DROP PROCEDURE sp_m_liv;
```

Criando Views

Uma view pode ser pensada de como uma " query " armazenada. Uma view definida cria essencialmente uma definição nomeada para uma declaração SQL que pode ser então referenciada como uma tabela em outras declarações SQL.
É uma complexa definição para uma situação simples, aqui você armazena a query e depois a chama pelo nome (fácil né?).
Para criar uma view, execute o comando a seguir:

```
CREATE OR REPLACE VIEW
  vw_livros(nome,titulo,ano_publicacao,editora_nome) AS
        SELECT nome, titulo,
        ano_publicacao, editora_nome
        FROM autores
```

```
    INNER JOIN publicacao USING (autor_id, autor_id)
    INNER JOIN livros USING (isbn, isbn)
    INNER JOIN editora USING (editora_id, editora_id);
```

Para chamar a VIEW, basta usar o comando como se estivesse fazendo uma query:

```
SELECT * FROM vw_livros;
```

Ou se preferir:

```
SELECT titulo, editora_nome, nome FROM vw_livros WHERE nome='Edson';
```

Visualizando a estrutura de uma view

Assim como nas tabelas, execute o comando:

```
DESC vw_livros;
```

Visualizando a criação da view

Para visualizar o código da view, execute o comando:

```
SHOW CREATE VIEW vw_livros;
```

Como você pode ver, as VIEWS são tratadas como se fossem uma outra tabela, armazenando apenas a query que você deseja executar, podendo ser manipulada depois.

Excluindo uma view

Para se excluir a view, execute o seguinte comando:

```
DROP VIEW vw_livros;
```

Criando Triggers (gatilhos)

Um trigger é um tipo especial de programa armazenado que dispara quando uma tabela é modificada por uma declaração INSERT, UPDATE, ou DELETE (DML). Triggers implementam funcionalidade que tem que acontecer sempre que uma certa mudança ocorre na tabela. Porque gatilhos são diretamente fixos à tabela, código de aplicação não pode evitar gatilhos no banco de dados.

Usos típicos de gatilhos incluem a implementação lógica de negócios críticos, a denormalização de dados para reações de desempenho, e os auditando nas mudanças feitas a uma tabela. Podem ser definidos gatilhos para disparar antes ou depois que uma declaração DML específica execute.

Para criar a Trigger, primeiramente crie a seguinte tabela:

```
CREATE TABLE livros_backup (
  isbn CHAR(13) NOT NULL,
  titulo VARCHAR(50) default NULL,
  edicao_num TINYINT(2) default NULL,
  ano_publicacao YEAR(4) default NULL,
  descricao TEXT,
  dt_exclusao DATETIME
) ENGINE=InnoDB;
```

Essa tabela será uma tabela de backup da tabela de livros, assim sendo, quando você excluir da tabela de livros, essa tabela armazenará os dados excluídos. Para que isso seja possível, você terá que criar um gatilho (trigger):

```
  DELIMITER $$
    CREATE TRIGGER tr_livbackup BEFORE DELETE ON livros
```

```
  FOR EACH ROW
  BEGIN
     DECLARE DATA DATE;
     SET DATA = NOW();
     INSERT INTO livros_backup SET isbn = OLD.isbn,
     titulo=OLD.titulo,
     edicao_num=OLD.edicao_num,
     ano_publicacao=OLD.ano_publicacao,
     descricao=OLD.descricao,
    dt_exclusao = DATA;
  END $$
DELIMITER;
```

Você pode criar gatilhos no MySQL para comandos INSERT, UPDATE e DELETE, com já havia sido dito. No caso, o comando fora feito para a execução do DELETE.
Note a palavra **OLD** sendo usada antes de cada COLUNA. Isso significa que ele receberá os dados antigos. Como a execução deve ser feita antes da exclusão dos dados na tabela livros, você vê na declaração da TRIGGER o comando **BEFORE** e seguido do comando SQL que o fará disparar **DELETE** e em que tabela.
Um loop **FOR EACH** varrerá os dados e os colocará na tabela de backup.

Visualizando as triggers criadas

Execute o comando para visualizar as triggers criadas, bem como sua estrutura:

```
show triggers;
```

Excluindo uma trigger

Para se excluir a trigger, utilize o comando a seguir:

```
DROP TRIGGER tr_livbackup
```

Administrando o MySQL

Um sistema de MySQL pode ter muitos usuários. O usuário root geralmente deve ser utilizado somente para propósitos de administração, por razões de segurança. Para cada usuário que precisar utilizar o sistema, você precisará configurar uma conta e senha.
Não é obrigatório configurar senhas para usuários, mas recomendo que você configure senhas para todos os usuários que forem criados.

Entendendo o sistema de privilégios do MySQL

O MySQL suporta um sofisticado sistema de privilégios. Um privilégio é um direito que um usuário tem para realizar uma ação particular em um objeto particular.
Quando você cria um usuário no MySQL, você concede a ele um conjunto de privilégios para especificar o que ele pode e não pode fazer dentro do sistema.

Configurando usuários

Os comandos GRANT e REVOKE são utilizados para fornecer e retirar direitos dos usuários do MySQL. Ele pode ser concedido nos seguintes níveis:

- Global

- Banco de dados
- Tabela
- Coluna

O comando para criar um usuário com privilégios é como mostrado a seguir:

GRANT privilégios [colunas] ON item
TO nome_do_usuario [IDENTIFIED BY 'senha']
[WITH GRANT OPTION]

As cláusulas entre colchetes são opcionais.
Para conceder privilégios ao um usuário no banco livraria, você deve criar um usuário com os seguintes privilégios:

```
mysql> grant all
    -> on livraria.*
    -> to edson identified by 'integrator';
```

Com isso você concede todos os privilégios de manipulação do banco de dados livraria somente ao usuário **edson**, com a senha **integrator**.

Confirmando o novo usuário

Para confirmar a criação do novo usuário, você deve executar o comando a seguir:

```
flush privileges;
```

Revogando privilégios

Para revogar esse privilégio você deve fazer o seguinte comando:

```
mysql> revoke all
    -> on livraria.*
    -> from edson;
```

Obtendo informações com SHOW

Se você desejar visualizar todos os privilégios que um usuário tem, execute o seguinte comando:

```
SHOW GRANTS FOR edson;
```

Para visualizar todos os usuários existentes no seu MySQL execute;

```
SHOW GRANTS;
```

Integrando PHP e MYSQL

O poder do PHP5 unido ao poderio do MySQL 5 torna o desenvolvimento mais rápido e preciso, possibilitando uma gama de situações antes não vistas graças às limitações do MySQL.

Acessando seu banco de dados pelo PHP

A partir de agora você irá trabalhar com o banco de dados em MySQL e o PHP, onde terá uma integração completa de inserção, alteração, seleção e exclusão de dados.

Conectando ao MySQL e visualizando dados

```
<?php
	$conexao=mysql_connect("localhost","edson","integrator");
	if(!$conexao){
		echo "Erro ao se conectar";
		exit;
	}
	$banco=mysql_select_db("livraria");
	if(!$banco){
		echo "O Banco de dados não foi encontrado";
		exit;
	}

$rs=mysql_query("SELECT * FROM livros");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Visualizando dados da tabela livros</title>
</head>

<body>
<table width="489" border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th width="109" align="left">ISBN</th>
    <th width="202" align="left">T&iacute;tulo</th>
    <th width="82" align="left">Edi&ccedil;&atilde;o N.&ordm;
</th>
    <th width="96" align="left">Publicado em: </th>
  </tr>
<?php while($row=mysql_fetch_array($rs)){ ?>
  <tr>
    <td><?php echo $row['isbn']?></td>
    <td><?php echo $row['titulo']?></td>
    <td align="center"><?php echo $row['edicao_num']?></td>
    <td align="center"><?php echo $row['ano_publicacao']?></td>
  </tr>
<?php }//end if ?>
```

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

vis_dados_mysql.php

A conexão com o servidor de dados em MySQL é feita utilizando a função **mysql_connect( )** que tem as seguintes passagens de informações:
mysql_connect (“local”, “usuário”, “senha”);
Com condição **IF**, você verifica se foi possível ou não a conexão. Caso não seja possível de se conectar ao servidor de dados específico, o construtor **exit( )** termina a execução da página, não permitindo que apareça mais nem um erro. Antes, temos um **echo** que irá imprimir na tela do usuário uma mensagem.
A função utilizada para selecionar o banco de dados aqui é a **mysql_select_db( )**. A função tem a seguinte passagem de parâmetro:
mysql_select_db(“banco_de_dados”)
A linha utilizada para a execução do SQL é onde se encontra a função **mysql_query( ).** Essa função tem o seguinte parâmetro:
mysql_query(“string_sql”)
Com a função **mysql_fetch_array( )** você consegue recuperar os dados vindos do banco de dados através da função **mysql_query( )** e transportá-lo para o PHP em formato de array (por isso **array** ao final do nome da função).
O loop **WHILE** existe para que se faça uma varredura em todas as linhas, imprimindo os resultado através da estrutura **echo**.
Note que uma variável, chamada **$row** se transforma em array, acessando assim os dados vindos do banco de dados através de uma chamada associativa, com $row**['nome_da_coluna_da_tabela']**.

Parâmetros opcionais de mysql_fetch_array

Usando **MYSQL_BOTH**, você terá uma matriz com os índices associativos e numéricos. Usando **MYSQL_ASSOC**, você terá apenas os índices associativos, usando **MYSQL_NUM**, você terá apenas os índices numéricos.
Onde você poderá acessar os resultados da seguinte maneira, com **MYSQL_BOTH**:

```
<?php while($row=mysql_fetch_array($rs, MYSQL_BOTH )){ ?>
  <tr>
    <td><?php echo $row[ 'isbn' ]?></td>
    <td><?php echo $row[ 1 ]?></td>
    <td align="center"><?php echo $row[ 2 ]?></td>
    <td align="center"><?php echo $row[ 'ano_publicacao' ]?></td>
  </tr>
<?php }//end if ?>
```

Esse parâmetro visto anteriormente é o padrão, não precisando ser declarado.

Com o parâmetro **MYSQL_ASSOC**:

```
<?php while($row=mysql_fetch_array($rs, MYSQL_ASSOC )){ ?>
  <tr>
    <td><?php echo $row[ 'isbn' ]?></td>
    <td><?php echo $row[ 'titulo' ]?></td>
    <td align="center"><?php echo $row[ 'edicao_num' ]?></td>
```

```
   <td align="center"><?php echo $row[ 'ano_publicacao' ]?></td>
  </tr>
<?php }//end if ?>
```

E por fim com o parâmetro **MYSQL_NUM**:

```
<?php while($row=mysql_fetch_array($rs, MYSQL_ASSOC )){ ?>
  <tr>
    <td><?php echo $row[ 0 ]?></td>
    <td><?php echo $row[ 1 ]?></td>
    <td align="center"><?php echo $row[ 2 ]?></td>
    <td align="center"><?php echo $row[ 3 ]?></td>
  </tr>
<?php }//end if ?>
```

A escolha de qualquer uma dessas formas não implica em diferença na velocidade de acesso aos dados.

Inserindo dados na tabela livros

O código a seguir será responsável pelo cadastro de livros:

```
<?php
        //conecta ao banco de dados
        $conexao=mysql_connect("localhost","edson","integrator");
        //acessa o banco de dados desejado
        $banco=mysql_select_db("livraria");

        //captura os dados vindos do formulário HTML
        $isbn=trim( $_POST['isbn'] );
        $titulo=trim( $_POST['titulo'] );
        $edicao=trim( $_POST['edicao'] );
        $publicacao=trim( $_POST['publicacao'] );
        $descricao=trim( $_POST['descricao'] );
        //verifica se os dados recebidos não são vazios com empty
        if(!empty( $isbn ) && !empty( $titulo )
                && !empty( $edicao ) && !empty( $publicacao )
                && !empty( $descricao ) )
        {
                $query="INSERT INTO livros
                                VALUES('$isbn',
                                        '$titulo',
                                        '$edicao',
                                        '$publicacao',
                                        '$descricao')";
                $ins=mysql_query( $query );
                //verifica se o resultado dado é falso
                if( $ins===FALSE )
                        $msg= "Erro na query... " . mysql_error( ) . "<br/>";
                else{
                        $msg= "Foi inserida " . mysql_affected_rows( ) . " linha <br/>";
                        //destrói as variáveis criadas para receber os dados
```

```
                        unset( $isbn,$titulo,$edicao,$publicacao,$descricao );
                }

        }//end if

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Cadastro de Livros</title>
</head>

<body>
<?php if( isset( $msg ) )
                echo $msg;
?>
<form id="form1" name="form1"
                method="post"
                action="<?php echo $_SERVER['PHP_SELF']?>">
 <table width="259" border="0" cellspacing="2" cellpadding="0">
   <tr>
    <th colspan="2">Cadastro de Livros </th>
   </tr>
   <tr>
    <td width="87" align="right">ISBN:</td>
    <td width="166">
      <input name="isbn" type="text" id="isbn" value="<?php echo $isbn?>" />
    </td>
   </tr>
   <tr>
    <td align="right">Título:</td>
    <td>
      <input name="titulo" type="text" id="titulo" value="<?php echo $titulo?>" />
     </td>
   </tr>
   <tr>
    <td align="right">Edição N.º;: </td>
    <td>
      <input name="edicao" type="text" id="edicao" value="<?php echo $edicao?>" />
     </td>
   </tr>
   <tr>
    <td align="right">Publicaçãoo:</td>
    <td>
      <input name="publicacao" type="text" id="publicacao"
            value="<?php echo $publicacao?>" />
     </td>
   </tr>
```

```
   <tr>
    <td align="right" valign="top">Descrição:</td>
    <td>
     <textarea name="descricao" rows="5" id="descricao">
     <?php echo $descricao?></textarea>
     </td>
   </tr>
   <tr>
    <td colspan="2" align="center">
         <input name="bt_cad" type="submit" id="bt_cad" value="Cadastrar" />
         <input name="bt_limp" type="reset" id="bt_limp" value="Limpar" />
   </td>
   </tr>
  </table>
</form>
</body>
</html>
```

cad_livros.php

No começo desse exemplo você capturará os dados vindos do formulário através do método **$_POST**. Você percebe os valores recebidos envoltos na função **trim( )**. Essa função retira espaço no inicio e final de uma string.
Logo após através da condição **IF** você verifica se os valores não (**!**) estão vazios com a função **empty( )**.
Caso não estejam, é construído uma query com a instrução SQL **INSERT**, que possibilita a inserção de dados no banco de dados, como já visto anteriormente.
A variável **$ins** tem como atribuição o valor resultante de **mysql_query( )**. Através dessa atribuição, você pode verificar se o valor foi executado ou não. Isso é feito através do **IF** que compara a variável **$ins** é **IDÊNTICO** a **FALSE**. Caso seja, significa que a query apresentou um erro, ou na construção da sintaxe ou no recebimento dos dados que estão tentando serem inseridos.
O resultado e **FALSE** é a variável **$msg** com o valor de uma String **CONCATENADA** com **mysql_error( )**. A função **mysql_error( )** retorna o erro vindo do banco de dados MySQL, dando ao desenvolvedor a possibilidade de depurá-lo, caso seja um problema da instrução SQL.
Caso contrário, **ELSE**, a variável **$msg** receberá a o valor de uma String **CONCATENADA** com a função **mysql_affected_rows( )**. Essa função resulta no número de linhas afetadas pela instrução SQL (que no caso sempre será com o valor 1).
Ainda na condição **ELSE**, um construtor de linguagem aparece: o **unset( )**. O **unset( )** destrói variáveis, que, mais do que necessário nessa ocasião, as variáveis usadas para inserir os dados serão destruídas, evitando assim com que seus dados sejam acessíveis em qualquer outra parte do código a seguir.
Se **unset( )** não é chamado, veja que se executou um erro no formulário, o que permite que os dados preenchidos anteriormente sejam impressos nos valores dos campos (**VALUE**) desse formulário.

Alterando o cadastro de livros

O cadastro de livros agora precisa ser alterado, pois os dados foram inseridos e precisam ser acessíveis para possíveis alterações de erros ou outras informações.

O código a seguir unirá muitos dos conhecimento adquiridos no PHP 5:

```
<?php
   //conecta ao banco de dados
   $conexao=mysql_connect("localhost","edson","integrator");
   //acessa o banco de dados desejado
   $banco=mysql_select_db("livraria");

   //método que verifica se os campos estão preenchidos
   function preenchido( )
   {
        //verifica o número de argumentos
        $numargs = func_num_args( );
        //pega os argumentos passados
        $arg_list = func_get_args( );
        //no loop verifica se os argumentos estão vazios com empty
        for ($i = 0; $i < $numargs; $i++){
                        if(empty($arg_list[$i]))
                                return false;//retorna false caso estejam
        }
        return true;//retorna true caso não estejam

   }
   //método que coloca aspas nas strings para execução da instrução SQL
   function valor_string($valor)
   {
     //verifica se está ativo as cotas mágicas
     //se não estiver, usa addslashes
     $valor = (!get_magic_quotes_gpc( )) ? addslashes($valor) : $valor;
     $valor = ($valor != "") ? "'" . $valor . "'" : "NULL";
     return $valor;
   }

   //captura os dados vindos do formulário HTML
   $isbn=trim($_REQUEST['isbn']);
   $titulo=trim($_POST['titulo']);
   $edicao=trim($_POST['edicao']);
   $publicacao=trim($_POST['publicacao']);
   $descricao=trim($_POST['descricao']);
   //verifica se foi postado o formulário de atualização
   if(isset($_POST['bt_atu'])){

     try{
        //verifica se os dados recebidos não são vazios
        if( !preenchido($titulo,$edicao,$publicacao,$descricao) ){
          //se forem vazios dispara uma exceção e encerra a execução
          //pulando para a cláusula catch
          throw new Exception('Você não preencheu os campos corretamente<br />');
        }
        //com sprintf você pode trabalhar com strings
        //essa função retorna string formatado
```

```
            $query=sprintf("UPDATE livros
                                SET    titulo = %s,
                                       edicao_num =  %s,
                                       ano_publicacao = %s,
                                       descricao = %s
                                WHERE isbn=%s",
                                valor_string($titulo),
                                valor_string($edicao),
                                valor_string($publicacao),
                                valor_string($descricao),
                                valor_string($isbn));

        $update=mysql_query($query);
        //verifica se o resultado dado é falso dispara uma exceção e encerra
        if($update===FALSE) {
                throw new Exception("Erro na atualização... " . mysql_error() . "<br />");
        }
        else{
                $msg= "Foi atualizada " . mysql_affected_rows() . " linha<br />";
                //destrói as variáveis criadas para receber os dados
                unset($isbn,$titulo,$edicao,$publicacao,$descricao);
        }
    }
    catch (Exception $e)
    {
        //caso haja uma exceção à mensagem é capturada e atribuída a $msg
        $msg = $e->getMessage( );
    }
  }
$rs=mysql_query("SELECT * FROM livros");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Atualização de Livros</title>
</head>
<body>
<?php
        //verifica se existe a variável $msg
        if(isset($msg))
                echo $msg;
?>

<p>
<table width="489" border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th width="109" align="left">ISBN</th>
    <th width="202" align="left">Título</th>
```

```
   <th width="82" align="left">Edição N.º </th>
   <th width="96" align="left">Publicado em: </th>
        <th width="96" align="left">Atualizar </th>
  </tr>
  <?php
  //varre todos os dados da tabela
  while($row=mysql_fetch_array($rs)){
  ?>
  <tr>
    <td><?php echo $row['isbn']?></td>
    <td><?php echo $row['titulo']?></td>
    <td align="center"><?php echo $row['edicao_num']?></td>
    <td align="center"><?php echo $row['ano_publicacao']?></td>
     <td align="center">
     <a href="<?php echo $_SERVER['PHP_SELF']?>?isbn=
          <?php echo urlencode($row['isbn'])?>">
     Clique aqui
     </a>
     </td>
  </tr>
  <?php }//end if?>
</table>
</p>

<?php
//verifica se você clicou no link e se não está vazio o valor de isbn
if( ( !empty($_SERVER['QUERY_STRING']) && !empty($isbn) ) ) :
    try{
          $result=mysql_query("SELECT * FROM livros WHERE isbn='$isbn'");
          //se a query resultar em um erro dispara essa exceção
          if($result===FALSE) {
                throw new Exception("Erro na consulta... " . mysql_error( ) . "<br />");
          }
          //se o resultado for igual a zero, ou seja, não existir o ISBN
          //dispara essa exceção
          if(mysql_num_rows($result)==0)
                throw new Exception("Não existem dados no ISBN procurado");

          $row = mysql_fetch_assoc($result);
?>
<form id="form1" name="form1" method="post"
          action="<?php echo $_SERVER['PHP_SELF']?>">
  <table width="259" border="0" cellspacing="2" cellpadding="0">
    <tr>
      <th colspan="2">Atualização de Livros </th>
    </tr>
    <tr>
      <td width="87" align="right">ISBN:</td>
      <td width="166">
          <strong><?php echo $isbn?></strong>
```

```
        <input name="isbn" type="hidden" id="isbn"
            value="<?php echo $row['isbn']?>" />
      </td>
    </tr>
    <tr>
      <td align="right">Título:</td>
      <td>
        <input name="titulo" type="text" id="titulo"
            value="<?php echo $row['titulo']?>" />
        </td>
    </tr>
    <tr>
      <td align="right">Edição N.º;: </td>
      <td>
        <input name="edicao" type="text" id="edicao"
            value="<?php echo $row['edicao_num']?>" />
        </td>
    </tr>
    <tr>
      <td align="right">Publicação:</td>
      <td>
        <input name="publicacao" type="text" id="publicacao"
            value="<?php echo $row['ano_publicacao']?>" />
        </td>
    </tr>
    <tr>
      <td align="right" valign="top">Descrição:</td>
      <td>
     <textarea name="descricao" rows="5" id="descricao">
        <?php echo $row['descricao']?>
     </textarea>
      </td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input name="bt_atu" type="submit" id="bt_cad" value="Atualizar" />
     </td>
    </tr>
  </table>
</form>
<?php
        }
        catch (Exception $e)
        {
                //caso haja uma exceção à mensagem é ecoada na tela
                echo "<strong>{$e->getMessage( )}</strong>";
        }
endif;

?>
```

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

atu_livros.php

Dois métodos foram criados no começo do código, sendo o primeiro chamado de **preenchido( )**. O método **preenchido( )** foi desenvolvido para receber diversos argumentos, não possuindo um número exato. Se algum desses argumentos passados estiverem vazios, o método retorna **FALSE**.
O método **valor_string( )** verifica se as cotas mágicas estão habilitadas no PHP.INI, através de **get_magic_quotes_gpc( )**. Caso não estejam, o método utilizará **addslashes()**.
Em seguida, a variável recebe os strings **'aspas simples'** para que seja possível usar mais adiante na instrução SQL.
A manipulação de exceção ocorre quando você utiliza o bloco **try...catch**. O bloco **try** é o responsável por verificar se existem um problema ocorrendo. É nesse ponto que você tem um **IF** verificando se não estão preenchidos os valores vindos do formulário de atualização. Se não estiverem, uma exceção é disparada em **throw new Exception( )**. Caso haja um disparo, imediatamente o código muda para o bloco **catch** e exibe o valor disparado.
Caso não haja um erro, o código segue, onde é executada a instrução SQL **UPDATE**, que dessa vez fora chamada com a função **sprintf**, que captura os strings e os retorna formatados.
Na execução da **QUERY**, novamente uma verificação é feita para evitar erros, que se houver, dispara também uma **EXCEÇÃO**. Caso não dispare, uma mensagem é dada ao usuário indicando que os dados foram atualizados com sucesso.
Abaixo da **TABELA** é feita uma verificação para que seja exibido ou não o **FORMULÁRIO**. A verificação é feita através de **$_SERVER['QUERY_STRING']** e pelo valor da variável **$isbn**. Se você se lembrar, **QUERY_STRING** exibe os resultados passados pelo cabeçalho HTTP, usados após o símbolo de consulta '**?**'. Esse é o método usado para o envio do código ISBN do livro para executar a atualização.
Um outro método aparecera, chamado de **mysql_num_rows( )**, capaz de capturar o número de linhas vindos de uma instrução SQL **SELECT**. O método **mysql_affected_rows( )** não foi utilizado porque ele não funciona com a instrução **SELECT**. Se o resultado for igual a **ZERO**, quer dizer que não existe o ISBN procurado, fazendo com que seja disparada uma exceção.

Trabalhando com MySQL Improved Extension

Com o avanço do MySQL, principalmente em sua nova versão, a 5, o PHP precisava também melhorar seu processo de comunicação com o MySQL nesta versão. Sendo assim, fora criado uma extensão chamada de **mysqli**. A extensão mysqli (improved - aperfeiçoado) permite a você acessar a funcionalidade provida pelo MySQL na versão 4.1.2 ou superior.
O **mysqli** possui uma semelhança com a extensão **mysql**, mas com muito mais funcionalidades.

Obs.: Para ter estas funções disponíveis, você deverá compilar o PHP com suporte para a extensão mysqli. No Windows, você precisará habilitar, se não estiver habilitada, a extensão **php_mysqli.dll**.

Criando a conexão com o mysqli

Nessa etapa você criará uma conexão com o mysqli em um arquivo externo, com a intenção de se incluir nas demais páginas.

```
<?php

function conexao( )
{
    //executa a conexão com mysqli pela interface orientada a objeto
    $result = new mysqli('localhost', 'edson', 'integrator', 'livraria');
    //se não for possível se conectar dispara uma exceção
    if (!$result)
        throw new Exception('Não foi possível se conectar com o banco de dados');
    else
        return $result;//caso contrário retorna o resultado
}

?>
```

conexao.php

O método **mysqli** é chamado aqui com uma interface orientada a objeto, mas você também pode optar pelo modo procedural, como é feito em **mysql_connect( )**:

$result = mysqli_connect('localhost', 'edson', 'integrator', 'livraria');

Criando o arquivo de funções

Embora a idéia seja colocar mais de um método, no início você apenas colocará um, o mesmo usado no de atualização de livros, para que se verifique se foi preenchido os campos do formulário:

```
<?php
        //método que verifica se os campos estão preenchidos
        function preenchido( )
        {
                //verifica o número de argumentos
                $numargs = func_num_args( );
                //pega os argumentos passados
                $arg_list = func_get_args( );
                //no loop verifica se os argumentos estão vazios com empty
                for ($i = 0; $i < $numargs; $i++){
                                if(empty($arg_list[$i]))
                                        return false;//retorna false caso estejam
                }
                return true;//retorna true caso não estejam

        }
?>
```

func.php

Criando o Stored Procedure para inserir autores

Para criar o Stored Procedure para inserir autores, execute o comando a seguir no terminal ou prompt de comando:

```
DELIMITER $$
  CREATE PROCEDURE sp_ins_autor(in_nome VARCHAR(100), in_dt_nasc DATE)
  BEGIN

  IF in_nome IS NOT NULL AND in_dt_nasc IS NOT NULL THEN

     INSERT INTO autores (nome, dt_nasc)  VALUES (in_nome, in_dt_nasc );

  END IF;

  END $$
DELIMITER ;
```

Criando o cadastro de autores

O código a seguir requisitará tanto a conexão como as funções para que seja possível de ser manipuladas pelo arquivo, chamando-as por **require_once**( ).

```
<?php
   require_once("conexao.php");
   require_once("func.php");

   $autor=trim($_POST['autor']);
   $dt_nasc=trim($_POST['dt_nasc']);

   //verifica se foi postado o formulário de inserção
if(isset($_POST['bt_cad'])){
   try{
      $con=conexao( );
      if( !preenchido($autor,$dt_nasc) )
         throw new Exception('Você não preencheu os campos corretamente<br />');

      $dt=explode("/",$dt_nasc);
      $dt_nasc="{$dt[2]}-{$dt[1]}-{$dt[0]}";

      $result = $con->query("CALL sp_ins_autor('$autor','$dt_nasc')");

      if(!$result)
         throw new Exception('Problemas: '.$con->errno.' --- '.$con->error.'<br />');
      else
         $msg="O cadastro foi inserido com sucesso!";

      unset($autor,$dt_nasc);


   }
```

```
    catch(Exception $e){
        //caso haja uma exceção à mensagem é capturada e atribuída a $msg
        $msg = $e->getMessage( );
    }
}


?>


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Cadastro de Autores</title>
</head>

<body>
<?php
    //verifica se existe a variável $msg
    if(isset($msg))
        echo $msg;
?>
<form id="form1" name="form1" method="post"
     action="<?php echo $_SERVER['PHP_SELF']?>">
  <table width="265" border="0" cellspacing="2" cellpadding="0">
    <tr>
      <th colspan="2">Cadastro de Autores </th>
    </tr>
    <tr>
      <td width="107" align="right">Nome do Autor: </td>
      <td width="152">
        <input name="autor" type="text" id="autor" value="<?php echo $autor?>" />
    </td>
    </tr>
    <tr>
      <td align="right">Data de Nasc.: </td>
      <td>
    <input name="dt_nasc" type="text" id="dt_nasc" value="<?php echo $dt_nasc?>" />
    </td>
    </tr>
    <tr>
      <td colspan="2">
        <input name="bt_cad" type="submit" id="bt_cad" value="Cadastrar" />
    </td>
    </tr>
  </table>
</form>
</body>
```

```
</html>
```

cad_autores.php

Muitas das situações aqui vistas já são conhecidas por exemplos anteriores. Dentro da cláusula **try** você chama o método **conexao( )**, atribuindo-o à variável **$con**. A partir desse momento, você pode acessar com **$con** os métodos da classe **mysqli**, existente no arquivo **conexao.php**.
Como a data de nascimento será preenchida como em nosso idioma, é chamado **explode**( ) que retorna uma matriz de Strings. A variável **$dt** capta os dados quebrados em forma de matriz e os torna acessível como uma matriz linear.
A variável **$dt_nasc** recebe essa matriz reorganizando-a no formato de data aceito pelo MySQL.
Com o método **query( )** de **mysqli**, você executa a Stored Procedure **sp_ins_autor**( ), inserindo assim os valores recebidos.

Selecionando os autores através de Stored Procedure

Crie a procedure no terminal ou no prompt de comando:

```
DELIMITER $$
  CREATE PROCEDURE sp_sel_autores( )
  BEGIN

     SELECT autor_id as id, nome,
       DATE_FORMAT(dt_nasc,"%d/%m/%Y") as nascimento
     FROM autores;

  END $$
DELIMITER ;
```

Chamando a Stored Procedure para visualizar os autores

O código a seguir chamará a Stored Procedure criada para exibir seus resultados na página:

```
<?php
   require_once("conexao.php");
   require_once("func.php");

   try{
      $con=conexao( );

      $results = $con->query("CALL sp_sel_autores( )");

      if(!$results)
         throw new Exception('Problemas: '.$con->errno.' --- '.$con->error.'<br />');

   }
   catch(Exception $e){
      //caso haja uma exceção à mensagem é capturada e atribuída a $msg
      $msg = $e->getMessage( );
```

```
    }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Visualizar Autores</title>
</head>

<body>
<table width="317" border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th width="190" align="left">Autor</th>
    <th width="121" align="left">Data Nascimento </th>
  </tr>
  <?php
  try{
    if( $results->num_rows==0 )
        throw new Exception('Não existem dados cadastrados no momento<br />');

    while($row = $results->fetch_assoc( )) :

  ?>
  <tr>
    <td><?php echo $row['nome']?></td>
    <td><?php echo $row['nascimento']?></td>
  </tr>
  <?php
    endwhile;
  }
  catch(Exception $e){
        //caso haja uma exceção à mensagem é exibida em echo
        echo $e->getMessage( );
    }
    //fecha os resultados
    $results->close( );
    //fecha a conexão
    $con->close( );

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

vis_autores.php

Embora o método de trabalho seja diferente, é praticamente a mesma coisa que fazer com o método já visto anteriormente.
Você tem em **$results** a capacidade de acessar os métodos vindos da query, que no caso

fora utilizado o **fetch_assoc( )**, similar ao **mysql_fetch_assoc( )** visto anteriormente.
O acesso a cada campo é de acordo com o **ALIAS** criado na query armazenada na Stored Procedure.
Depois de pegos os resultados, você fecha a conexão e os results com o método **close( )**.

Atualizando os autores por Stored Procedure

O código a seguir demonstra como atualizar um cadastro usando Stored Procedure:

```
DELIMITER $$
  CREATE PROCEDURE sp_aut_autor(in_nome VARCHAR(100), in_dt_nasc DATE, in_id INT)
  BEGIN

    IF in_nome IS NOT NULL AND in_dt_nasc IS NOT NULL AND in_id IS NOT NULL THEN

      UPDATE autores SET nome=in_nome, dt_nasc=in_dt_nasc WHERE autor_id=in_id;

    END IF;

  END $$
DELIMITER ;
```

Criando a procedure que seleciona um autor em específico

A Stored Procedure a seguir seleciona somente um autor pelo código do cadastro:

```
DELIMITER $$
  CREATE PROCEDURE sp_sel_autor(in_id INT)
  BEGIN

    SELECT autor_id, nome, DATE_FORMAT(dt_nasc,'%d/%m/%Y') as dt_nasc
    FROM autores WHERE autor_id=in_id;

  END $$
DELIMITER ;
```

Criando a página de atualização de autores

A página de atualização de autores contém todos os meios já vistos em uma página só:

```
<?php
  require_once("conexao.php");
  require_once("func.php");

  $autor=trim($_POST['autor']);
  $dt_nasc=trim($_POST['dt_nasc']);
  $id=trim($_REQUEST['id']);

  try{
    $con=conexao( );
```

```
        if(isset($_POST['bt_atu'])){

            if( !preenchido($autor,$dt_nasc) )
                throw new Exception('Você não preencheu os campos corretamente<br />');

            $dt=explode("/",$dt_nasc);
            $dt_nasc="{$dt[2]}-{$dt[1]}-{$dt[0]}";

            $up = $con->query("CALL sp_aut_autor('$autor','$dt_nasc', $id)");

            if($up===FALSE)
                throw new Exception('Problemas: '.$con->errno.' --- '.$con->error.'<br />');
            else
                $msg="O cadastro foi atualizado com sucesso!";

            unset($autor,$dt_nasc);

        }//end if

        $results = $con->query("CALL sp_sel_autores( )");

        if($results===FALSE)
            throw new Exception('Problemas: '.$con->errno.' --- '.$con->error.'<br />');

    }
    catch(Exception $e){
        //caso haja uma exceção à mensagem é capturada e atribuída a $msg
        $msg = $e->getMessage( );
    }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Atualizar Autores</title>
</head>
<body>
<?php
    //verifica se existe a variável $msg
    if(isset($msg))
        echo $msg;
?>
<table width="410" border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th width="129" align="left">Autor</th>
    <th width="165" align="left">Data Nascimento </th>
    <th width="108" align="left">Atualizar</th>
  </tr>
  <?php
```

```
 try{

   if( $results->num_rows==0 )
       throw new Exception('Não existem dados cadastrados no momento<br />');

   while($row = $results->fetch_assoc( )):

 ?>
 <tr>
  <td><?php echo $row['nome']?></td>
  <td><?php echo $row['nascimento']?></td>
  <td>
      <a href="<?php echo $_SERVER['PHP_SELF']?>?id=<?php echo $row['id']?>">
         Clique aqui
      </a>
   </td>
 </tr>
 <?php
   endwhile;
 ?>
</table>
<?php
  $results->close( );

 }
 catch(Exception $e){
       //caso haja uma exceção à mensagem é exibida em echo
       echo $e->getMessage( );
 }
?>
<p>
<?php

if(  !empty($_SERVER['QUERY_STRING']) && !empty($id)  ) :
    try{

       $result = $con->query("CALL sp_sel_autor($id)");

         if ($result === FALSE)
           throw new Exception("Erro no resultado... " . $con->error . "<br />");

       //se o resultado for igual a zero, ou seja, não existir o ISBN
       //dispara essa exceção
   if( $result->num_rows==0 )
       throw new Exception("Não existem dados no AUTOR procurado");

   $row = $result->fetch_object( );

?>
<form id="form1" name="form1" method="post" action="<?php echo
```

```
$_SERVER['PHP_SELF']?>">
  <table width="265" border="0" cellspacing="2" cellpadding="0">
    <tr>
      <th colspan="2">Cadastro de Autores </th>
    </tr>
    <tr>
      <td width="107" align="right">Nome do Autor: </td>
      <td width="152"><input name="autor" type="text" id="autor" value="<?php echo
$row->nome?>" /></td>
    </tr>
    <tr>
      <td align="right">Data de Nasc.: </td>
      <td><input name="dt_nasc" type="text" id="dt_nasc" value="<?php echo $row-
>dt_nasc?>" /></td>
    </tr>
    <tr>
      <td colspan="2"><input name="bt_atu" type="submit" id="bt_atu"
value="Atualizar" /></td>
    </tr>
  </table>
  <input name="id" type="hidden" id="id" value="<?php echo $row->autor_id?>" />
</form>
</p>
<?php
        $result->close( );
    }//end try
    catch (Exception $e)
    {
        //caso haja uma exceção à mensagem é ecoada na tela
        echo "<strong>{$e->getMessage( )}</strong>";
    }
endif;

$con->close( );
?>
</body>
</html>
```

atu_autores.php

Excluindo autores

Para excluir os autores, você irá criar um procedimento para executar essa ação:

```
DELIMITER $$
      CREATE PROCEDURE sp_del_autor(in_id INT)
      BEGIN

        DELETE FROM autores WHERE autor_id=in_id;

      END $$
DELIMITER ;
```

A página a seguir faz o trabalho de exclusão de autores:

```
<?php
    require_once("conexao.php");
    require_once("func.php");

    $id=trim($_REQUEST['id']);

    try{
        $con=conexao( );

        if(isset($id) && preenchido($id)){

            if( !preenchido($id) )
                throw new Exception('Você não clicou em um autor<br />');

            $del = $con->query("CALL sp_del_autor($id)");

            if($del===FALSE)
                throw new Exception('Problemas: '.$con->errno.' --- '.$con->error.'<br />');
            else
                $msg="O cadastro foi EXCLUIDO com sucesso!";

            unset($id);

        }//end if

        $results = $con->query("CALL sp_sel_autores( )");


    }
    catch(Exception $e){
        //caso haja uma exceção à mensagem é capturada e atribuída a $msg
        $msg = $e->getMessage();
    }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Excluir Autores</title>
</head>

<body>
<?php
    //verifica se existe a variável $msg
    if(isset($msg))
        echo $msg;
?>
```

```
<table width="410" border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th width="129" align="left">Autor</th>
    <th width="165" align="left">Data Nascimento </th>
    <th width="108" align="left">Excluir</th>
  </tr>
  <?php
  try{

     if( $results->num_rows==0 )
         throw new Exception('Não existem dados cadastrados no momento<br />');

     while($row = $results->fetch_assoc( )):
  ?>
  <tr>
    <td><?php echo $row['nome']?></td>
    <td><?php echo $row['nascimento']?></td>
    <td>
         <a href="<?php echo $_SERVER['PHP_SELF']?>?id=<?php echo $row['id']?>">
            Clique aqui
         </a>
    </td>
  </tr>
  <?php
     endwhile;
  ?>
</table>
<?php
   $results->close( );

  }
  catch(Exception $e){
         //caso haja uma exceção à mensagem é exibida em echo
         echo $e->getMessage( );
  }
     $con->close( );
?>
</body>
</html>
```

exc_autores.php

Por ser uma página com situações semelhantes as já vistas, não serão apresentados detalhes ou explicações.

Lendo e manipulando o conteúdo de um diretório

Algumas vezes, você terá que verificar o conteúdo de um diretório e até mesmo alterá-lo, excluindo.

O exemplo abaixo mostra o método **dir( )** que faz a abertura do diretório. Com o método **read ( )** o loop WHILE faz uma varredura em todos os arquivos contidos dentro do diretório.
A função **realpath( )** retorna o caminho físico real de um determinado diretório.
A função **unlink( )** faz com que você possa excluir arquivos.

```
<?php
$arq=trim($_GET['arq']);
$path = realpath("arquivos");
if(!empty($arq))
     unlink($path.'/'.$arq);
?>
<html>
<head><title>Trabalhando com arquivos</title></head>
<body>
<?php
     $_dir = dir($path);
     echo "Diretório: " . $_dir->path . "<br/>Conteúdo:<br/>";
  while (($file=$_dir->read())!==FALSE) {

     if($file!="." && $file!=".."){
        echo "<a href=\"{$_SERVER['PHP_SELF']}?arq=$file\">
                 $file</a><br/>";
     }
  }
  $_dir->close();
?>
</body>
</html>
```

manip_arq.php

Criando arquivos em PDF

Usado para o desenvolvimento de relatórios, você tem no PHP diversas formas de desenvolver PDF's.
A forma escolhida dessa apostila é com a utilização da biblioteca FPDF, escrita totalmente em PHP.
Como característica positiva, essa biblioteca não depende da instalação de nenhuma extensão extra do PHP.
Como negativa, a extensão de PDF do PHP é mais rápida que essa biblioteca, mas não tão atratível como a biblioteca FPDF, pois é considerada mais complexa.
A biblioteca FPDF pode ser encontrada no site oficial: **http://www.fpdf.org**.
A seguir você tem o código que cria um relatório em PDF utilizando a biblioteca FPDF e com acesso ao banco de dados MySQL:

```
<?php
   //PDF SCRIPT By EDSON - INTEGRATOR
   define('FPDF_FONTPATH','fpdf/font/');
   require_once('fpdf/fpdf.php');

   //Arquivo que se conecta com o banco de dados
   require_once("conexao.php");

class PDF extends FPDF
{
   //Cabeçalho da página
```

```
    function Header( )
    {
        //Logo
        $this->Image('fpdf/tutorial/logo_pb.png',10,8,33);
        //Arial bold 15
        $this->SetFont('Arial','B',15);

        //Move para a direita
        $this->Cell(80);
        //Título
        $this->Cell(60,10,'Cabeçalho da Página',0,0,'C');
        //Quebra de linha
        $this->Ln(20);
    }

    //Rodapé da página
    function Footer( )
    {
        //Posição de  1.5 cm da borda inferior
        $this->SetY(-15);
        //Arial italic 8
        $this->SetFont('Arial','I',8);
        //Número da página
        $this->Cell(0,10,'Página '.$this->PageNo().'/{nb}',0,0,'C');
        //name
        $this->SetTextColor(0,0,255);
        $this->SetFont('','U');
        $this->SetY(-5);
        $this->SetX(90);
        $this->Write(5,'www.integrator.com.br','http://www.integrator.com.br');
    }

}

    //Criando um novo arquivo de PDF
    //na classe, você pode definir a visualização (“L” em minúsculo) indica Paisagem,
    //o default é Retrato, as  medidas usadas na página e o formato
    //da página: A3, A4 e etc
    $pdf=new PDF('l','mm','A4');
    $pdf->AliasNbPages( );

    //Abre o arquivo
    $pdf->Open( );

    //Desabilita a quebra automática de páginas
    $pdf->SetAutoPageBreak(false);

    //Adiciona a primeira página
    $pdf->AddPage( );
```

```
    //coloca o valor do eixo y na posição por página
    $y_axis = 30;
    //coloca a altura da linha
    $row_height = 6;

    //Imprime os títulos para a página atual
    //coloca a cor de fundo
    $pdf->SetFillColor(232,232,232);
    //coloca o cor da fonte
    $pdf->SetTextColor(0,0,160);
    $pdf->SetFont('Arial','B',12);
    $pdf->SetY($y_axis);
    $pdf->SetX(25);
    $pdf->Cell(180,$row_height,'Livros Cadastrados',1,0,'C',1);
    //adiciona a altura  da linha seguinte
    $y_axis = $y_axis + $row_height;
    $pdf->SetY($y_axis);
    $pdf->SetX(25);
    $pdf->Cell(50,$row_height,'ISBN',1,0,'C',1);
    $pdf->Cell(100,$row_height,'Titulo',1,0,'C',1);
    $pdf->Cell(30,$row_height,'Edição',1,0,'C',1);

    $y_axis = $y_axis + $row_height;

    try{
        $con=conexao( );

            //seleciona os livros para serem mostrados no seu arquivo PDF
        $result=$con->query('select isbn,titulo,edicao_num from livros ORDER BY
titulo');

        if($result===FALSE)
            throw new Exception('Problemas: '.$con->errno.' --- '.$con->error.'<br />');

        //Inicializa o contador
        $i = 0;

        //Coloca o máximo de linhas por página
        $max = 5;

        if( $result->num_rows==0 )
            throw new Exception('Não existem dados cadastrados no momento');

        while($row = $result->fetch_array( ))
        {
            //Se a linha atual é da próxima página, é criada uma nova página e é
            //impressa os títulos novamente
            if ($i == $max)
            {
                    $pdf->AddPage( );
```

```
                $y_axis = 30;
                //Imprime os títulos das colunas para a página atual
                $pdf->SetY($y_axis);
                $pdf->SetX(25);
                $pdf->SetFillColor(232,232,232);
                $pdf->SetTextColor(0,0,160);
                $pdf->Cell(50,$row_height,'ISBN',1,0,'C',1);
                $pdf->Cell(100,$row_height,'Titulo',1,0,'C',1);
                $pdf->Cell(30,$row_height,'Edição',1,0,'C',1);

                //Vai para a próxima linha
                $y_axis = $y_axis + $row_height;

                //Pega a variável $i e coloca o valor 0 (primeira linha)
                $i = 0;
        }
        //coloca o efeito zebra nas linhas
        if($i%2)
                $pdf->SetFillColor(255,255,255);
        else
                $pdf->SetFillColor(245,245,245);
        $isbn = $row['isbn'];
        $titulo = $row['titulo'];
        $edicao = $row['edicao_num'];
        //coloca a cor da fonte
        $pdf->SetTextColor(0,0,0);
        $pdf->SetY($y_axis);
        $pdf->SetX(25);
        //adiciona os valores do banco nas células
        $pdf->Cell(50,$row_height,$isbn,1,0,'L',1);
        $pdf->Cell(100,$row_height,$titulo,1,0,'L',1);
        $pdf->Cell(30,$row_height,$edicao,1,0,'R',1);

        //vai para a próxima linha
        $y_axis = $y_axis + $row_height;
        $i ++;
    }

    //Cria o arquivo
    //abaixo você tem a possibilidade de enviar para o browser para salvar como
    //$pdf->Output('arquivo.pdf','D');
    //o exemplo abaixo é para ser exibido diretamente no browser
    $pdf->Output( );//abre o plug-in padrão do PDF para visualizar o arquivo de saída
    Header('Pragma: public'); //Deve ser colocado para exibição no Internet Explorer
    $result->close( );
}
catch(Exception $e){
    //caso haja uma exceção a mensagem é capturada e atribuida a $msg
    echo $e->getMessage( );
}
```

```
    $con->close( );
```

gerar_pdf.php

Arquivos de Excel

Para criar um arquivo que tenha como saída o formato do Excel, ou xls, você inicialmente precisa ter a formatação padrão usada por ele. Isso é fácil de se conhecer, bastando apenas exportá-lo no formato XML. Com os header's a seguir, você tem a possibilidade de ter a saída para download.

```
<?php
      header ( "Expires: Mon, 1 Apr 1974 05:00:00 GMT" );
      header ( "Last-Modified: " . gmdate("D,d M YH:i:s") . " GMT" );
      header ( "Pragma: no-cache" );
      header ( "Content-type: application/x-msexcel" );
      header ( "Content-Disposition: attachment; filename=arquivo.xls" );
      header ( "Content-Description: PHP Generated XLS Data" );
?>
```

O código a seguir demonstra como funciona um documento com saída em formato XLS (MS Excel), com dados vindos do banco de dados.

Uma situação crítica que deve ser observada está em **`ss:ExpandedRowCount`**, em destaque a seguir. Isso porque você tem que passar ao arquivo o número de linhas correto. Caso não o faça, um erro ocorre.

```
<?php
header( "content-type: text/xml" );
header ( "Expires: Mon, 1 Apr 1974 05:00:00 GMT" );
header ( "Last-Modified: " . gmdate("D,d M YH:i:s") . " GMT" );
header ( "Pragma: no-cache" );
header ( "Content-type: application/x-msexcel" );
header ( "Content-Disposition: attachment; filename=arquivo.xls" );
header ( "Content-Description: PHP Generated XLS Data" );

          //conecta ao banco de dados
          $conexao=mysql_connect("localhost","edson","integrator");
          //acessa o banco de dados desejado
          $banco=mysql_select_db("livraria");
  //seleciona os dados da tabela
  $sql = "SELECT * FROM livros";
  $resultado = mysql_query($sql);
  $nr=mysql_num_rows($resultado); //conta o número de linhas encontradas

//colocado em PHP para evitar erros de interpretação pelo servidor
echo( "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>\n" );
echo( "<?mso-application progid=\"Excel.Sheet\"?>\n" );
?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:x="urn:schemas-microsoft-com:office:excel"
xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:html="http://www.w3.org/TR/REC-html40">
```

```
<DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
 <Author>Edson Gonçalves</Author>
 <LastAuthor>Edson Gonçalves</LastAuthor>
 <Created>2005-04-30T14:08:07Z</Created>
 <LastSaved>2005-04-30T14:09:14Z</LastSaved>
 <Company>Integrator Tecnology and Design</Company>
 <Version>10.2625</Version>
</DocumentProperties>
 <OfficeDocumentSettings xmlns="urn:schemas-microsoft-com:office:office">
 <DownloadComponents/>
 <LocationOfComponents HRef=""/>
 </OfficeDocumentSettings>
 <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
 <WindowHeight>15930</WindowHeight>
 <WindowWidth>20025</WindowWidth>
 <WindowTopX>480</WindowTopX>
 <WindowTopY>105</WindowTopY>
 <ProtectStructure>False</ProtectStructure>
 <ProtectWindows>False</ProtectWindows>
 </ExcelWorkbook>
 <Styles>
 <Style ss:ID="Default" ss:Name="Normal">
 <Alignment ss:Vertical="Bottom"/>
 <Borders/>
 <Font/>
 <Interior/>
<NumberFormat/>
<Protection/>
</Style>
</Styles>
<Worksheet ss:Name="Livros no Excel">
<Table ss:ExpandedColumnCount="3"
ss:ExpandedRowCount="<? print( $nr + 1 ) ?>"
x:FullColumns="1"
x:FullRows="1">
<Column ss:AutoFitWidth="0" ss:Width="117"/>
<Row>
<Cell><Data ss:Type="String">Título</Data></Cell>
<Cell><Data ss:Type="String">Edição N.</Data></Cell>
<Cell><Data ss:Type="String">Publicado em</Data></Cell>
</Row>
<?php while($row = mysql_fetch_array($resultado)){ ?>
<Row>
<Cell><Data ss:Type="String"><? print( $row["titulo"] ) ?></Data></Cell>
<Cell><Data ss:Type="Number"><? print( $row["edicao_num"] ) ?></Data></Cell>
<Cell><Data ss:Type="Number"><? print( $row["ano_publicacao"] ) ?></Data></Cell>
</Row>
<? } ?>
</Table>
<WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
<Selected/>
<Panes>
<Pane>
```

```
<Number>3</Number>

<ActiveRow>2</ActiveRow>
</Pane>
</Panes>
<ProtectObjects>False</ProtectObjects>
<ProtectScenarios>False</ProtectScenarios>
</WorksheetOptions>
</Worksheet>
</Workbook>
```

trab_excel.php

Criando arquivos em RTF

Criar arquivos em formato RTF com o PHP é relativamente simples. Primeiro você precisa usar um editor de arquivos como o BrOffice.org Writer ou MS Word e criar um arquivo com o formato mostrado a seguir:

Formato do arquivo chamado de arquivoRTF.rtf

Data do relatório: <<data>>
A seguir você tem uma lista de Livros encontrados no banco de dados:

Título	Edição N.º	Publicado em
<<titulo>>	<<edicao>>	<<publicado>>

A seguir você tem um exemplo de listagem dos livros simples, para que seja através desse arquivo a chamada para o outro que irá gerar o RTF com os novos dados:

```
<?php

	try{
	//conecta ao banco de dados
	$conexao=mysql_connect("localhost","edson","integrator");
	//acessa o banco de dados desejado
	$banco=mysql_select_db("livraria");

		$rs = mysql_query("SELECT * FROM livros");

		if(!$rs)
			throw new Exception('Problemas: '.mysql_error().'<br />');

	}
	catch(Exception $e){
		//caso haja uma exceção a mensagem é capturada e atribuida a $msg
		$msg = $e->getMessage();
	}
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Visualizar Livros</title>
</head>

<body>
<?php
        //verifica se existe a variável $msg
        if(isset($msg))
                echo $msg;
?>
<table width="550" border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th width="109" align="left">ISBN</th>
    <th width="177" align="left">T&iacute;tulo</th>
    <th width="79" align="left">Edi&ccedil;&atilde;o N.&ordm; </th>
    <th width="96" align="left">Publicado em: </th>
    <th width="77" align="left">Gerar RTF  </th>
  </tr>
  <?php
  //varre todos os dados da tabela
  while($row=mysql_fetch_array($rs)){
  ?>
  <tr>
    <td><?php echo $row['isbn']?></td>
    <td><?php echo $row['titulo']?></td>
    <td align="center"><?php echo $row['edicao_num']?></td>
    <td align="center"><?php echo $row['ano_publicacao']?></td>
    <td align="center"><a href="geraRTF.php?isbn=<?php echo
urlencode($row['isbn'])?>"> Clique aqui </a> </td>
  </tr>
  <?php }//end if?>
</table>
</body>
</html>
```

list_livros.php

A seguir você fará o arquivo que receberá o ISBN do livro para gerar o arquivo RTF com os dados.

```
<?php
        try{
              //conecta ao banco de dados
              $conexao=mysql_connect("localhost","edson","integrator");
               //acessa o banco de dados desejado
              $banco=mysql_select_db("livraria");
              //seleciona o livro escolhido na lista
```

```
            $sql = "SELECT * FROM livros WHERE ISBN='{$_GET['isbn']}'";
            $resultado = mysql_query($sql);
            //se houver problemas na query, lança uma exceção
            if(!$resultado)
                        throw new Exception('Problemas: '.mysql_error( ).'<br />');

            $nr=mysql_num_rows($resultado);
  }
        catch(Exception $e){
                //caso haja uma exceção a mensagem é capturada e ecoada na tela
                echo $e->getMessage( );
        }


//gera os cabeçalhos corretos para a saída do documento RTF
header( "Content-Type: application/msword" );
header( "Content-Disposition: inline, filename=arquivoRTFModificado.rtf");
$data = date( "d/m/Y" );
// abre o arquivo RTF criado como template
$filename = "arquivoRTF.rtf";
$output = file_get_contents($filename);
// altera as informações dentro do arquivo template para os dados vindos do banco
$row = mysql_fetch_array($resultado);
        $output = str_replace( "<<data>>", $data, $output );
        $output = str_replace( "<<titulo>>", $row["titulo"] , $output );
        $output = str_replace( "<<edicao>>", $row["edicao_num"], $output );
        $output = str_replace( "<<publicado>>", $row["ano_publicacao"], $output );


//envia a saída gerada para o browser
echo $output;

?>
```

geraRTF.php

O método **file_get_contents( )** é o preferível para se ler o conteúdo de um arquivo em uma string. Ela usa técnicas de mapeamento de memória suportadas em seu sistema operacional para melhorar a performance. Mas você poderia ter usado o conhecido **fopen( )**, **fwrite( )** e **fclose( )** se desejasse.
O exemplo a seguir faz exatamente isso e gera um novo arquivo, usando o método **copy( )** para copiá-lo com o novo nome no diretório especificado.

```
<?php

  try{
    //conecta ao banco de dados
    $conexao=mysql_connect("localhost","edson","integrator");
    //acessa o banco de dados desejado
    $banco=mysql_select_db("livraria");
```

```
  $sql = "SELECT * FROM livros WHERE ISBN='{$_GET['isbn']}'";
  $resultado = mysql_query($sql);
  if(!$resultado)
    throw new Exception('Problemas: '.mysql_error().'<br />');

  $nr=mysql_num_rows($resultado);

  $data = date( "d/m/Y" );
  // open our template file
  $arquivo = "arquivoRTF.rtf";
  $novoarq="arquivoRTFGerado.rtf";
  $dir = './';
  if(!copy($arquivo,$dir.$novoarq))
    throw new Exception('Não foi possível copiar o arquivo<br />');

  if(file_exists($arquivo)) {
    $abre=fopen($arquivo,"r");//abre o arquivo original
    $abre_novo=fopen($novoarq,"w+");//abre o arquivo novo
    $saida=fread($abre,filesize($arquivo));

    $row = mysql_fetch_array($resultado);
    $saida = str_replace( "<<data>>", $data, $saida );
    $saida = str_replace( "<<titulo>>", $row["titulo"] , $saida );
    $saida = str_replace( "<<edicao>>", $row["edicao_num"], $saida );
    $saida = str_replace( "<<publicado>>", $row["ano_publicacao"], $saida );

    $grava=fwrite($abre_novo,$saida);//escreve a saída no arquivo novo
    fclose($abre);//fecha o original
    fclose($abre_novo);//fecha o novo
  }
  else
    throw new Exception('Não foi possível encontrar o arquivo<br />');

    echo "Arquivo gerado com sucesso!";
 }
  catch(Exception $e){
  //caso haja uma exceção a mensagem é capturada e atribuida a $msg
  echo $e->getMessage();
 }


?>
```

gerarRTF.php

Desenvolvendo aplicações Web com PHP e Ajax

Nessa etapa da apostila você aprenderá a criar aplicações escritas em PHP5 que acessam dados sem o famoso reload da página. Para que isso seja possível, um conjunto de

tecnologias entram em ação, possibilitando a você desenvolver aplicações Web com características similares a encontrada nas aplicações para desktop.

O que é AJAX?

AJAX é carregar e renderizar uma página, utilizando recursos de scripts rodando pelo lado cliente, buscando e carregando dados em background sem a necessidade de reload da página.

AJAX é acrônimo para: **Asynchronous JavaScript And XML** e foi gerado por Jesse James Garret, em um artigo no site http://www.adaptivepath.com/publications/essays/archives/000385.php, da sua empresa Adaptive Path, em fevereiro de 2005.

Ajax não é uma tecnologia, mas sim um conjunto de tecnologias. O conceito de AJAX se resume em conhecer bem JavaScript, trabalhando com DOM (Document Object Model), CSS (Cascading Style Sheets) e XML.

Como o AJAX trabalha

Enquanto em uma aplicação Web clássica o navegador tem que ir buscar as informações no servidor e retornar para o cliente, no Ajax ocorre de forma diferente. No carregamento da página, toda a lógica de processamento de dados é passado ao cliente. Quando o usuário faz uma requisição, quem busca e trás essas informações é o JavaScript, de forma assíncrona, não causando assim o chamado "reload" na tela.

O tratamento dos dados, seu formato e exibição fica toda por conta do script que foi carregado inicialmente quando se acessou a página. O processo inicial de carregamento é mais lento que de uma aplicação comum, pois muitas informações são pré-carregadas. Mas depois, somente os dados são carregados, tornando assim o site mais rápido.

Criando uma página com Ajax

Com o uso do objeto XMLHttpRequest, que faz parte do padrão ECMA e está presente em todas as boas versões do Javascript. Os browsers que suportam esse padrão são:

- Opera 8
- Mozilla e Firefox
- Konqueror
- Safari

Além disso o Internet Explorer, desde a versão 5, suporta o Microsoft XMLHTTP, um substituto para o XMLHttpRequest.

Há duas maneiras de se fazer uma requisição com um objeto XMLHttpRequest, uma é síncrona, outra assíncrona. No modo síncrono, quando você manda o objeto fazer uma requisição, o seu script é interrompido esperando pelo retorno. No modo assíncrono a requisição é feita em segundo plano e seu script continua a ser executado. Em modo síncrono, você tem o problema de ter seu navegador congelado enquanto seu script é executado. E isso é ruim, pois podem ser que seja rápida a requisição e pode ser que não, ai você pergunta, será que está funcionando ou travou? O negócio é evitar esse método.

ajax.php

```
<?php
try{
  //conecta ao banco de dados
  $conexao=mysql_connect("localhost","edson","integrator");
  //acessa o banco de dados desejado
  $banco=mysql_select_db("livraria");
  $rs = mysql_query("SELECT * FROM livros");
  if(!$rs)
  throw new Exception('Problemas: '.mysql_error().'<br />');
}
catch(Exception $e){
  //caso haja uma exceção a mensagem é capturada e atribuida a $msg
  $msg = $e->getMessage();
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Trabalhando com Ajax</title>
<script language="JavaScript">

function Dados(isbn) {
  //verifica se o browser tem suporte a ajax
  try {
    ajax = new ActiveXObject("Microsoft.XMLHTTP");
  }
  catch(e) {
    try {
      ajax = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch(ex) {
      try {
          ajax = new XMLHttpRequest();
      }
      catch(exc) {
          alert("Esse browser não tem recursos para uso do Ajax");
          ajax = null;
      }
    }
  }
  //se tiver suporte ajax
  if(ajax) {
    ajax.open("GET", "livro.php?isbn="+isbn, true);
    ajax.onreadystatechange = function() {
    //enquanto estiver processando...emite a msg de carregando
    if(ajax.readyState == 1) {
```

```
      mensagem( "Carregando...!" );
    }
    //após ser processado - chama função processXML que vai varrer os dados
  if(ajax.readyState == 4 ) {
    if(ajax.responseXML) {
      processXML(ajax.responseXML);
    }
    else {
      //caso não seja um arquivo XML emite a mensagem abaixo
      mensagem( "Erro ao carregar" );
    }
  }
}
  ajax.send(null);
}
}//end function Dados

function processXML(obj){
    //pega a tag livro do XML
    var dataArray = obj.getElementsByTagName("livro");
    //total de elementos contidos na tag livro
    if(dataArray.length > 0) {
      //percorre o arquivo XML paara extrair os dados
      for(var i = 0 ; i < dataArray.length ; i++) {
          var item = dataArray[i];
          //contéudo dos campos no arquivo XML
          var isbn = item.getElementsByTagName("isbn")[0].firstChild.nodeValue;
          var titulo = item.getElementsByTagName("titulo")[0].firstChild.nodeValue;
          var edicao = item.getElementsByTagName("edicao")[0].firstChild.nodeValue;
          var publicacao = item.getElementsByTagName("publicacao")[0].firstChild.nodeValue;
      }
      mensagem( "Dados carregados" );
      document.getElementById('isbn').innerHTML=isbn;
      document.getElementById('titulo').innerHTML=titulo;
      document.getElementById('edicao').innerHTML=edicao;
      document.getElementById('publicacao').innerHTML=publicacao;
    }
}//end function processXML

function mensagem(msg){
    document.getElementById('mensagem').innerHTML=msg;
}//end function mensagem

</script>
</head>
<body>
<?php
//verifica se existe a variável $msg
if(isset($msg))
  echo $msg;
```

```
?>
<table width="253" border="1" cellspacing="0" cellpadding="0">
<tr>
<th width="137" align="left">ISBN</th>
<th width="110" align="left">Exibir dados </th>
</tr>
<?php
  //varre todos os dados da tabela
  while($row=mysql_fetch_array($rs)){
?>
<tr>
<td><?php echo $row['isbn']?></td>
<td align="center">
<a href="#" onclick="Dados('<?php echo urlencode($row['isbn'])?>')">
Clique aqui </a> </td>
</tr>
<?php }//end if?>
</table>
<p>
<div id="mensagem"></div>
<table width="295" height="92" border="0" cellpadding="2" cellspacing="0">
<tr>
<td width="75">ISBN:</td>
<td width="212"><span id="isbn"></span></td>
</tr>
<tr>
<td>Título:</td>
<td><span id="titulo"></span></td>
</tr>
<tr>
<td>Edição N.º</td>
<td><span id="edicao"></span></td>
</tr>
<tr>
<td>Publicação:</td>
<td><span id="publicacao"></span></td>
</tr>
</table>
</p>
</body>
</html>
```

livro.php

```
<?php
  //conexao ao mysql
  $conexao=mysql_connect("localhost","edson","integrator");
  //acessa o banco de dados desejado
  $banco=mysql_select_db("livraria");
```

```
  //recebendo o parâmetro
  $isbn = $_GET["isbn"];
  //executa a query
  $rs = mysql_query("SELECT * FROM livros WHERE isbn='$isbn'");
  //conta a quantidade de linhas encontradas
  $row = mysql_num_rows($rs);
  //se existem dados
  if($row>0) {
    //gera o xml
    $xml = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n";
    $xml .= "<livros>\n";
    //percorre os dados encontrados
    while($l=mysql_fetch_array($rs)){
          $xml .= "<livro>\n";
          $xml .= "<isbn>".$l['isbn']."</isbn>\n";
          $xml .= "<titulo>".$l['titulo']."</titulo>\n";
          $xml .= "<edicao>".$l['edicao_num']."</edicao>\n";
          $xml .= "<publicacao>".$l['ano_publicacao']."</publicacao>\n";
          $xml .= "</livro>\n";
    }//end while
    $xml.= "</livros>\n";
    //saída para o navegador
    header("Content-type: application/xml; charset=iso-8859-1");
  }//end if
  //echo do resultado
  echo $xml;
?>
```

Entendendo o AJAX

Para ter a forma com que o objeto ***XMLHttpRequest*** vai trabalhar, você tem que alterar o terceiro parâmetro do método **open**.

Com esse parâmetro em *true*, no terceiro parâmetro do método *open*, coloca o objeto em modo assíncrono.

O método **open** do objeto **XMLHttpRequest** permite abrir um documento, passar argumentos para ele e capturar uma resposta.

Com apenas dois métodos possíveis de se utilizar para acessar um documento: **GET** e **POST**, o método usado no exemplo é **GET**. No entanto se a quantidade de informações a ser passada for muito grande você deverá alterar para o método **POST**.

O método **send** ativa a conexão e faz a requisição de informações ao documento aberto pelo método **open**. Este método possui somente um parâmetro que serve para enviar dados extras ao documento que está sendo acessado.

O browser *Internet Explorer* não o obriga a passar nenhum parâmetro, mas outros navegadores como o *Mozilla*, exige algum dado, neste caso, a solução foi enviar **null**, mesmo não havendo necessidade de passar nenhum parâmetro.

Ao fazer a requisição o objeto vai executar o método **onreadystatechange**.

Esse código vai ser executado várias vezes durante a requisição, por isso é testado **readyState**. Quando **readyState** tiver o valor **4**, significa que a requisição foi concluída e que é possível ler o retorno e trabalhar com ele.

Para capturar a resposta do documento web acessado, você tem duas propriedades do objeto ***XMLHttpRequest***: **responseText** e **responseXML**.

A propriedade **responseText** contém o retorno do documento web acessado na forma de texto. Já a propriedade **responseXML** retorna um objeto DOM, em formato XML, podendo ser manipulado facilmente.

Apêndice A
A configuração do Apache (httpd.conf)

O servidor httpd Apache tem a função de servir páginas html para intranet ou Internet e possui grandes qualidades. Entre elas, e de onde o nome do aplicativo se originou, a de ser um programa bastante "patcheável" ("patchy" em inglês, logo o nome Apache veio de "a patchy server") estendendo suas capacidades iniciais por fazer uso de módulos diversos. Estes, adicionam o suporte a php, cgi e outros, e são de suma importância para um servidor http seguro e versátil.
A configuração básica abordada neste documento, tem 3 seções importantes:

```
Configuração do servidor principal
Configuração de um servidor em um domínio virtual
Habilitação de uma página hospedada no home de usuários
```

Configuração principal (editando o arquivo de configuração)
Os arquivos responsáveis por toda a configuração do httpd são três (acess.conf, srm.conf e httpd.conf), mas apenas um deles importa realmente (httpd.conf), sendo os outros dois considerados dispensáveis, já que é possível colocar tudo que poderia estar nestes, no principal.
As opções neste arquivo já estão comentadas, mas, mesmo assim, aqui vai uma descrição das mesmas. As principais são:

```
    # ServerType is either inetd, or standalone.  Inetd mode is
only supported
    # on
    # Unix platforms.
    #
    ServerType standalone
```

Diz ao sistema se o httpd vai ser rodado via script próprio (standalone), ou a partir do arquivo inetd.conf (inetd). (em "inetd" o httpd fica ocioso, enquanto o inetd fica monitorando as requisições, quando houver alguma, ele avisa e o serviço começa a funcionar)

```
    #
    # ServerRoot: The top of the directory tree under which the
server's
    # configuration, error, and log files are kept.
    #
    # NOTE!  If you intend to place this on an NFS (or otherwise
network)
    # mounted filesystem then please read the LockFile
documentation
    # (available at
http://www.apache.org/docs/mod/core.html#lockfile);
    # you will save yourself a lot of trouble.
    #
    # Do NOT add a slash at the end of the directory path.
```

```
    #
    ServerRoot /etc/httpd
```

Esta cuida do caminho do diretório onde irão ficar os arquivos de configuração. Pode ser mudado se necessário.

```
    #
    # Timeout: The number of seconds before receives and sends
time out.
    #
    Timeout 300
```

Tempo máximo (em segundos) que o servidor esperará, mantendo uma conexão aberta com o cliente. Se o limite for excedido, ele terá de criar uma nova conexão com o mesmo.

```
    #
    # KeepAlive: Whether or not to allow persistent connections
(more than
    # one request per connection). Set to "Off" to deactivate.
    #
    KeepAlive On
```

Diretamente ligado com a opção acima, define se o processo de manter a conexão com o cliente está ativo ou não.

```
    #
    # MaxKeepAliveRequests: The maximum number of requests to
allow
    # during a persistent connection. Set to 0 to allow an
unlimited amount.
    # We recommend you leave this number high, for maximum
performance.
    #
    MaxKeepAliveRequests 100
```

Número máximo de conexões mantidas, sem necessidade de renovação. Quanto mais alto o número, melhor a performance (com o hardware adequado).

```
    #
    # KeepAliveTimeout: Number of seconds to wait for the next
request from the
    # same client on the same connection.
    #
    KeepAliveTimeout 15
```

Máximo (de segundos) a espera de nova requisição.

```
    #
    # Number of servers to start initially --- should be a
reasonable ballpark
    # figure.
```

```
    #
    StartServers 10
```

Número de servers iniciais, ou seja, logo no início do processo, o httpd poderia responder a 10 conexões simultâneas ao mesmo site.

```
    #
    # Limit on total number of servers running, i.e., limit on
the number
    # of clients who can simultaneously connect --- if this
limit is ever
    # reached, clients will be LOCKED OUT, so it should NOT BE
SET TOO LOW.
    # It is intended mainly as a brake to keep a runaway server
from taking
    # the system with it as it spirals down...
    #
    MaxClients 150
```

Número máximo de conexões simultâneas por clientes ao site. Se for ultrapassada, mostrará a infame mensagem "http server busy".

```
    #
    # Listen: Allows you to bind Apache to specific IP addresses
and/or
    # ports, in addition to the default. See also the
<VirtualHost>
    # directive.
    #
    #Listen 3000
    #Listen 12.34.56.78:80
```

Permite ao __principal__ httpd server, responder em mais de um ip (descomentando o 12.34.56.78:80 por exemplo, habilitaria ao server http escutar em um ip além de seu ip normal (o da própria máquina))

```
    #
    # BindAddress: You can support virtual hosts with this
option. This
    # directive
    # is used to tell the server which IP address to listen to.
It can either
    # contain "*", an IP address, or a fully qualified Internet
domain name.
    # See also the <VirtualHost> and Listen directives.
    #
    BindAddress 192.168.255.108:80
```

Esse é importante. Por default a linha BindAddress vem comentada pois como está apresentada acima, habilita o acesso a um domínio virtual (em nosso caso, o ip 192.168.255.108:80 (o :80 seria indicando a porta 80) que será explicado mais além. Para cada virtual host, é necessária uma entrada "BindAddress e um número ip"

```
    #
    # Dynamic Shared Object (DSO) Support
    #
    # To be able to use the functionality of a module which was
built as a DSO
    # you
    # have to place corresponding `LoadModule' lines at this
location so the
    # directives contained in it are actually available _before_
they are used.
    # Please read the file README.DSO in the Apache 1.3
distribution for more
    # details about the DSO mechanism and run `httpd -l' for the
list of already
    # built-in (statically linked and thus always available)
modules in your
    # httpd
    # binary.
    #
    # Note: The order is which modules are loaded is important.
Don't change
    # the order below without expert advice.
    #

    # Example:
    # LoadModule foo_module modules/mod_foo.so
    #
    #LoadModule mmap_static_module modules/mod_mmap_static.so
    LoadModule env_module         modules/mod_env.so
    (seguido de uma lista de LoadModule e mais além, AddModule)
```

Descomentando quaisquer das linhas que comecem com LoadModule ou AddModule, valida o carregamento de módulos feito na inicialização do httpd. Estes funcionam como opções, por exemplo, habilitar ou não o suporte a arquivos cgi no server, etc

```
    #
    # Port: The port to which the standalone server listens. For
    # ports < 1023, you will need httpd to be run as root
initially.
    #
    Port 80
```

O httpd responde por default na porta 80, neste campo você poderá modificá-la se quiser.

```
    #
    # ServerAdmin: Your address, where problems with the server
should be
    # e-mailed.  This address appears on some server-generated
pages, such
    # as error documents.
    #
    ServerAdmin root@localhost
```

O endereço de email para onde será mandado algo se o server acusar erro ou anormalidades

```
    #
    # ServerName allows you to set a host name which is sent
back to clients for
    # your server if it's different than the one the program
would get (i.e.,
    # use
    # "www" instead of the host's real name).
    #
    # Note: You cannot just invent host names and hope they
work. The name you
    # define here must be a valid DNS name for your host. If you
don't
    # understand
    # this, ask your network administrator.
    # If your host doesn't have a registered DNS name, enter its
IP address
    # here.
    # You will have to access it by its address (e.g.,
http://123.45.67.89/)
    # anyway, and this will make redirections work in a sensible
way.
    #
    ServerName vader.suptel
```

Outro importante. Determina o nome do server __principal__. Importante: o nome tem que obrigatoriamente constar em DNS (um ip associado a um nome) pois apenas inventando um, não irá adiantar. O modo de chamá-lo seria http://nome mas se o mesmo não estiver em nenhum DNS, coloque o ip (seria http://numero-ip para chamá-lo então, neste caso).

```
    #
    # DocumentRoot: The directory out of which you will serve
your
    # documents. By default, all requests are taken from this
directory, but
    # symbolic links and aliases may be used to point to other
locations.
    #
    DocumentRoot "/html"
```

Determina o caminho onde estarão os arquivos html do servidor __principal__. IMPORTANTE: o diretório deve estar com permissão 755 (chmod 755, sendo leitura, escrita e execução para o dono, leitura e execução para grupo e outros que não sejam do grupo nem donos (others).

```
    #
    # This should be changed to whatever you set DocumentRoot
to.
    #
    <Directory "/html">
```

```
    #
    # This may also be "None", "All", or any combination of
"Indexes",
    # "Includes", "FollowSymLinks", "ExecCGI", or "MultiViews".
    #
    # Note that "MultiViews" must be named *explicitly* ---
"Options All"
    # doesn't give it to you.
    #
        Options Indexes FollowSymLinks Includes

    #
    # This controls which options the .htaccess files in
directories can
    # override. Can also be "All", or any combination of
"Options", "FileInfo",
    # "AuthConfig", and "Limit"
    #
        AllowOverride None

    #
    # Controls who can get stuff from this server.
    #
        Order allow,deny
        Allow from all
    </Directory>
```

Este conjunto de campos determinam as opções que os diretórios onde contém documentos htmls a serem acessados irão ter. A primeira "# This should.." deve conter o mesmo diretório que o "DocumentRoot" tem (o /html). IMPORTANTE: TODAS ESTAS DEVERÃO SER COPIADAS (E EDITADAS SE PRECISO) PARA AS PASTAS PRINCIPAIS QUE CONTIVEREM ARQUIVOS HTML DO SERVIDOR PRINCIPAL OU DOMINIO VIRTUAL. Se por exemplo você tiver apenas um servidor virtual alem do principal que responda no diretorio /vh (veremos como fazer essa associação mais além), você terá de ter as entradas <Directory "/vh"> e todas as abaixo desta, nem que sejam apenas copiadas, abaixo do término da ultima (# Controls who can get stuff from this server.).

```
    #
    # UserDir: The name of the directory which is appended onto
a user's home
    # directory if a ~user request is received.
    #
    UserDir public_html
```

Esta opção é bem útil. Cuida de qual diretório o usuário terá de fazer, se quiser ter uma página em seu home. No caso, como está configurado, ele precisará criar um diretório public_html (o nome pode ser alterado no campo acima) e colocar algo em html ali, podendo ser acessado com http://nome.da.maquina/~nome-do-usuário. IMPORTANTE: COMO MENCIONADO ANTERIORMENTE, ESTE E TODOS OS DIRETÓRIO ANTERIORES PRECISAM TER PERMISSÕES 755 AFIM DE GARANTIR ACESSO.

```
    #
    # Control access to UserDir directories.  The following is
an example
    # for a site where these directories are restricted to read-
only.
    #

    <Directory /home/*/public_html>
        AllowOverride FileInfo AuthConfig Limit
        Options MultiViews Indexes SymLinksIfOwnerMatch
        <Limit GET POST OPTIONS PROPFIND>
            Order allow,deny
            Allow from all
        </Limit>
        <Limit PUT DELETE PATCH PROPPATCH MKCOL COPY MOVE LOCK
UNLOCK>
    Order deny,allow
        Deny from all
        </Limit>
    </Directory>
```

Esta opção coordena os direitos de acesso ao diretório public_html dos usuários e vem por padrão, comentada. No caso você deve descomenta-la e modificá-la de acordo com o diretório home de seus usuários (por exemplo, o campo <Directory /home/*/public_html> diz que, no diretório /home, todos que existem dentro dele e que tenham public_html vão ser passíveis de acesso, sob as regras configuradas abaixo desta linha.

```
    #
    # DirectoryIndex: Name of the file or files to use as a pre-
written HTML
    # directory index.  Separate multiple entries with spaces.
    #
    DirectoryIndex index.html index.htm index.cgi
```

Esta é bastante importante também pois determina quais nomes de arquivos serão válidos para realizar-se a abertura dos mesmos em um browser http. No caso da configuração acima, o server aceitará arquivos de nome index.html, index.htm e index.cgi como arquivos iniciais de uma home page.

```
    #
    # UseCanonicalName:  (new for 1.3)  With this setting turned
on, whenever
    # Apache needs to construct a self-referencing URL (a URL
that refers back
    # to the server the response is coming from) it will use
ServerName and
    # Port to form a "canonical" name.  With this setting off,
Apache will
    # use the hostname:port that the client supplied, when
possible.  This
    # also affects SERVER_NAME and SERVER_PORT in CGI scripts.
    #
    UseCanonicalName On
```

Se ligada, uma pagina que por exemplo se chame http://www.teste.com/teste/ e seja acessada como http://www.teste.com/teste (sem o / no final) seja válida. Se desligada, ele não irá achar.

```
    #
    # LogLevel: Control the number of messages logged to the
error_log.
    # Possible values include: debug, info, notice, warn, error,
crit,
    # alert, emerg.
    #
    LogLevel warn
```

Determina em que nível o httpd irá rodar. A recomendada é a warn pois não causa acúmulo de atividades no apache e é uma das mais usadas.

```
    #
    # If you want to use name-based virtual hosts you need to
define at
    # least one IP address (and port number) for them.
    #
    NameVirtualHost 192.168.255.108:80
```

Neste, configuramos o ip e porta que o servidor virtual terá. A definição deste é que você não precisa ter vários computadores rodando http servers neles, com apenas um você pode ter www.teste.com e www.teste1.com, cada um abrindo uma página diferente (em diferentes diretórios do cpu) e cada um possuindo um ip (mas ambos apontarão para o mesmo cpu, isso se chama IP ALIAS).

```
    #
    # VirtualHost example:
    # Almost any Apache directive may go into a VirtualHost
container.
    #
    #<VirtualHost ip.address.of.host.some_domain.com>
    #    ServerAdmin webmaster@host.some_domain.com
    #    DocumentRoot /www/docs/host.some_domain.com
    #    ServerName host.some_domain.com
    #    ErrorLog logs/host.some_domain.com-error_log
    #    CustomLog logs/host.some_domain.com-access_log common
    #</VirtualHost>

    <VirtualHost teste:80>
        ServerAdmin webmaster@host.some_domain.com
            DocumentRoot /vh
        ServerName teste.suptel
        ErrorLog logs/host.some_domain.com-error_log
        CustomLog logs/host.some_domain.com-access_log common
    </VirtualHost>
```

Esta é a última mas não menos importante das configurações básicas do apache. Ela cuida do servidor virtual e de suas configurações, sendo que o que está comentado (com um # na frente) dá um exemplo do que deverá ser feito (as linhas descomentadas não estão escritas por padrão, estas apresentadas foram digitadas com base no exemplo). Explicarei cada uma delas:

```
    <VirtualHost teste:80>
```

Valida o nome "teste" ao servidor virtual e fala em que porta ele irá atender (80)

```
    ServerAdmin webmaster@host.some_domain.com
```

Se o sistema detectar algo de anômalo, um mail será enviado a webmaster@host.some_domain.com

```
    DocumentRoot /vh
```

Designa a pasta onde os arquivos html do servidor virtual serão colocados (LEMBRANDO QUE ESTA DEVERÁ TER PERMISSÕES 755 EM SEU CAMINHO INTEIRO COMO MENCIONADO ANTERIORMENTE).

```
    ServerName teste.suptel
```

Nome e domínio do servidor virtual.

```
    ErrorLog logs/host.some_domain.com-error_log
```

O relatório de erros do servidor virtual vai ser escrito em logs/host.some_domain.com-error_log

```
    CustomLog logs/host.some_domain.com-access_log common
```

Log de acessos vai ser escrito em logs/host.some_domain.com-access_log common

```
    </VirtualHost>
```

Indica o final da configuração do virtual host.
IMPROTANTE: DEPOIS DE QUALQUER MODIFICAÇÃO NESTE ARQUIVO (INCLUSIVE NA PRIMEIRA), O HTTPD DEVERÁ SER REINICIADO DIGITANDO COMO ROOT:

```
    [root@localhost /root]# cds
```

e após:

```
    [root@localhost /init.d]# ./httpd stop
    [root@localhost /init.d]# ./httpd start
```

Depois, é necessário informar ao sistema que o mesmo precisa responder num outro ip (192.168.255.108 como definido nas configurações do virtual host) além do ip verdadeiro (pois um virtual host não é nada mais do que fazer um computador responder em outro ip (e outro nome, se assim especificado no dns), direcionando o pedido http para este ip "falso" e associando a pasta de htmls referida ao mesmo).
Vamos utilizar então o linuxconf para adicionar este ip "falso" (técnica chamada de IP ALIAS, anteriormente mencionada).
Entre como root no linuxconf:

```
    [root@localhost]# linuxconf
```

Vá em:

```
    Ambiente de Rede -->
        Apelidos de IP para máquinas virtuais -->
                                                eth0 -->

    configure o ip virtual (no caso do nosso, seria
192.168.255.108) e sua
    máscara.
    Depois:

          Aceitar
         Sair
        Sair
       Sair
      Ativar as mudanças
```

Confirme se o novo ip está realmente online usando o comando ping:

[root@localhost]# ping 192.168.255.108

Coloque algum documento html no diretório /html (que de acordo com a configuração feita, é a pasta do server principal, podendo ter subpastas dentro desta) e em /vh (configurada para o virtualhost "teste" e que também pode ter subpastas), todas com as devidas permissões 755 previamente mencionadas.

E, para acessá-los, digite em um browser:

```
    http://vader.suptel (para o principal já que o nome é o do
servidor principal)

    ou:

    http://teste (que é o nome do virtual host)
```

O primeiro tem que estar respondendo no documento html válido na pasta /html e o segundo na /vh.

Para uma página no home do usuário, digite (após o usuário ter criado a pasta public_html em seu home e ter dado permissões 755 para a mesma).

```
    http://nome-da-maquina/~nome-do-usuário
```

deve responder no html do diretório /home/nome-do-usuário/public_html

Como faço para o apache abrir um diretório, sem conter o index.html?

Basta editar o arquivo de configuração do a pache /etc/httpd/conf/httpd.conf, que pode ser aberto com o comando:

```
    [root@localhost]# mcedit /etc/httpd/conf/httpd.conf
```

Inclua a opção Indexes, conforme o exemplo abaixo:

```
    <Directory "/html">

        Options Indexes FollowSymLinks Includes

    </Directory>
```

Bibliografia

Manual on-line do MySQL5 em www.mysql.com
Manual on-line do PHP em www.php.net
Manual FPDF em www.fpdf.org