APOSTILA PHP

2017

Profª Patrícia Gagliardo de Campos
Departamento de Processamento de Dados
Colégio Técnico de Campinas
Revisão: 1.4

Índice

Capítulo 1 – Introdução

PHP significa *PHP: Hypertext Preprocessor*. O PHP é uma linguagem de programação de código aberto, ou seja, não há necessidade de aquisição de licença ou outras permissões para uso da linguagem. O PHP é uma linguagem para criação de scripts Web do lado servidor. O script pode ser incorporado ao HTML ou pode ser um arquivo binário independente. Outras opções de linguagens para criação de scripts são o ASP (Active Server Pages) da Microsoft, o ColdFusion da Macromedia e o JSP (Java Server Pages) da Sun, sendo todos eles "proprietários".

Antes de começarmos a criar scripts, é importante entender o funcionamento do PHP no conjunto de arquivos que compõem um site, e isso faremos nesse capítulo. O PHP pouco interfere na formatação ou qualquer outro aspecto ligado à aparência de uma página Web. O trabalho do PHP é transparente ao usuário final, pois o resultado final do PHP é próprio HTML.

História

A primeira versão do PHP surgiu em 1994, por Rasmus Lerdorf (engenheiro de software, membro da equipe Apache), com o nome *Personal Home Page Tools.* Mais tarde, com o crescimento ascendente, Zeev Suraski e Andi Gutmans, ambos programadores israelenses, contribuíram cirando os analisadores de sintaxe do PHP3 e do PHP4.

O PHP 4 apresentava problemas na criação de cópias de objetos, sendo resolvido apenas na versão atual que é o PHP 5.

O PHP provê suporte a um grande número de bases de dados: Oracle, Sybase, PostgreSQL, InterBase, MySQL, SQLite, MSSQL, Firebird etc, podendo abstrair o banco com a biblioteca ADOdb, entre outras. Também tem suporte aos protocolos: IMAP, SNMP, NNTP, POP3, HTTP, LDAP, XML-RPC, SOAP. É possível abrir sockets e interagir com outros protocolos. E as bibliotecas de terceiros expandem ainda mais estas funcionalidades. Mais detalhes podem ser conferidos no site do PHP (www.php.net).

Características

Vamos comentar sobre algumas características do PHP que o faz ser uma linguagem com grande aceitação na parte de programação em ambientes Web.

- **O PHP é gratuito**

  O PHP é uma linguagem de código-fonte aberto, e por isso não há custo na obtenção de uma licença. Geralmente é possível obter o PHP através de um download gratuito ou através de sistemas operacionais, como o Linux, ou ainda juntamente no CD-ROM de livros técnicos.

  O site abaixo possui o contrato a respeito das licenças:

  - http://www.php.net/license.html

- **O PHP é fácil**

  O PHP é uma linguagem fácil de aprender quando comparado a outras linguagens de programação. O PHP tem uma sintaxe fácil de analisar e de fácil utilização.

  O PHP não possui um editor próprio, onde ao criar o código em ambientes gráficos basicamente apenas arrastando objetos. O PHP é uma linguagem manual, assim como o HTML, e que é possível escolher qualquer editor de texto para criar um programa. Alguns editores com suporte ao PHP para facilitar a criação do script.

- **O PHP é incorporado na HTML**

  O PHP é incorporado dentro de um código HTML. Vejamos um exemplo:

```
<HTML>
<HEAD>
<meta charset="windows-1252">
<TITLE> Bem-vindo Exemplo</TITLE>
</HEAD>

<BODY>
<P> Olá,

<?php

// Agora vamos inserir código PHP
// Usaremos variáveis estáticas, mas poderíamos obter as informações por // uma
base de dados ou cookies.

$var_firstname = "Patrícia";
$var_lastname = "Campos";
$var_titulo = "Sra.";

echo "$var_titulo $var_lastname";

// Agora continuaremos com o HTML!
?>

. Nós conhecemos quem você é! Seu primeiro nome é <?php echo $var_firstname;
?>. </P>


<P> Você está visitando esse site as <?php echo date('d-m-Y H:i:s'); ?>. </P>

</BODY>
</HTML>
```

Exemplo 1.1

Ao solicitar essa página, antes do conteúdo chegar ao browser do cliente, o servidor Web a pré-processa. Isso significa que o servidor Web percorre a página de cima para baixo, procurando por seções do PHP que tentará resolver.

- **O PHP é multiplataforma**

  O PHP é compatível com os três mais importantes servidores Web: o **Apache HTTP Server** para Unix e Windows, o **Microsoft Internet Information Server** e o **Netscape Enterprise Server** (também conhecido como iPlanet Server). Consultar Anexo I para conhecer a lista de servidores Web disponíveis.

  O PHP também executa em Macintosh, tornando praticamente multiplataforma.

- **O PHP não é baseado em tags**

  O PHP é uma linguagem de programação com as principais características das demais linguagens tradicionais. Diferentemente do ColdFusion (que é baseado em tags), no PHP é possível definir funções apenas digitando um nome e uma definição.

- **O PHP é rápido**

  O PHP é extremamente rápido em sua execução, principalmente quando compilado como um módulo do Apache no Unix.

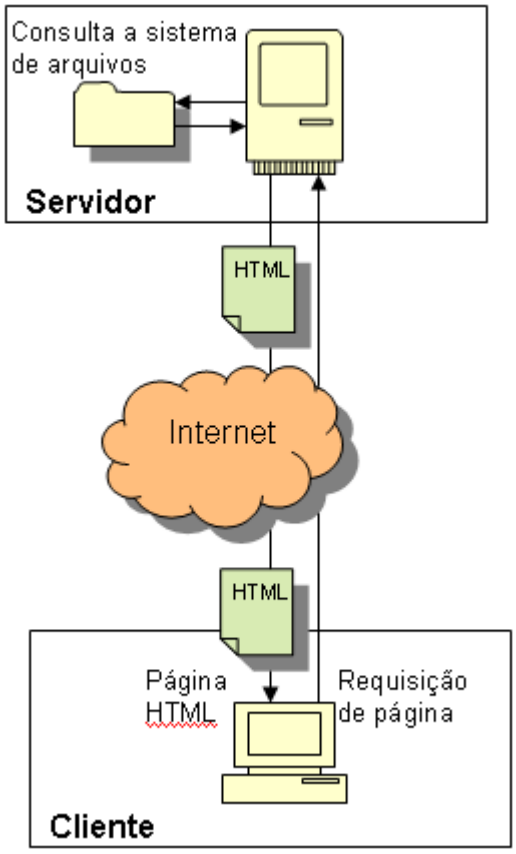
  O PHP perde apenas para scripts CGI (*Common Gateway Interface*). Apesar dos scripts CGI serem bastante velozes, eles perdem pelo fato de cada solicitação precisa gerar um processo independente e depois passado para o programa residente (*daemon*) http.

  O PHP, sendo um módulo do Apache, ele torna-se parte do próprio programa residente http, aumentando seu tempo de resposta economizando recursos da máquina.

- **O PHP facilita comunicação**

O PHP possui suporte a vários programas e protocolos, tornando-o bastante flexível. O PHP possui conectividade com os principais bancos de dados do mercado, além do ODBC. Também possui suporte a protocolos de rede como POP3, IMAP e LDAP. Possui suporte para Java e arquiteturas distribuídas (COM e CORBA).

Criando Script Web



A forma mais simples de uma página Web é uma página completamente estática, baseada apenas nos recursos (tags) da linguagem HTML.

A maioria dos recursos executados no lado cliente estão relacionados à formatação e ajustes à página para viabilizar uma determinada tarefa. Um fator a considerar é que as tecnologias do lado cliente dependem inteiramente do navegador. Como por exemplo, muitos clientes desativam o JavaScript por razões de segurança, o que torna impossível visualizar sites que utilizem excessivamente JavaScript para navegação.

Em outros casos, o navegador pode não estar atualizado, sendo assim não entenderá determinados recursos. O cliente pode optar por usar algum navegador menos popular e não ter suporte aos recursos desejados. Enfim, ao mesmo tempo que encontramos vantagens, outras inúmeras desvantagens nos faz atentarmos para alguns cuidados ao inserirmos recursos para serem executado no lado cliente.

No caso dos scripts do lado servidor, eles procuram tratar principalmente da conexão de sites Web aos servidores de *back-end*, como banco de dados. Essa comunicação pode-se dar de duas formas:

- **Servidor para cliente:** páginas Web podem ser criadas a partir de dados fornecidos por outros servidores.
- **Cliente para servidor:** informações inseridas pelo cliente podem ser manipuladas.

Exemplos de interação entre cliente e servidor são formulários e listas suspensas (exigem que se clique num botão para exibir conteúdo).

Abaixo temos um exemplo de uma página dinâmica. Começando pelo arquivo contendo o código-fonte PHP onde poderiam ser buscadas as informações num banco dados para serem inseridos à página.

```
<?php require_once($_SERVER['DOCUMENT_ROOT'] . "/cabecalho.inc.php"); ?>

<P> Este é o corpo de uma página em HTML! </P>

<?php require_once($_SERVER['DOCUMENT_ROOT'] . "/rodape.inc.php"); ?>
```

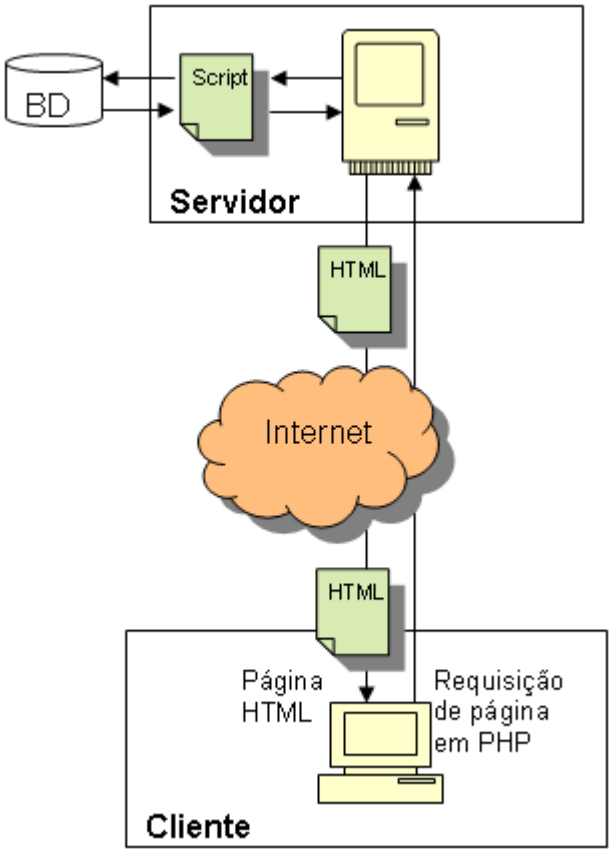
Exemplo 1.2

Agora temos o HTML gerado pelo script acima que será transmitido ao cliente:

```
<HTML>
<HEAD>
<TITLE> Exemplo de include! </TITLE>
</HEAD>
```

```
<BODY>

<P> Este é o corpo de uma página em HTML! </P>

<HR>
<P>Colégio Técnico de Campinas</P>
</BODY>
</HTML>
```

Exemplo 1.3

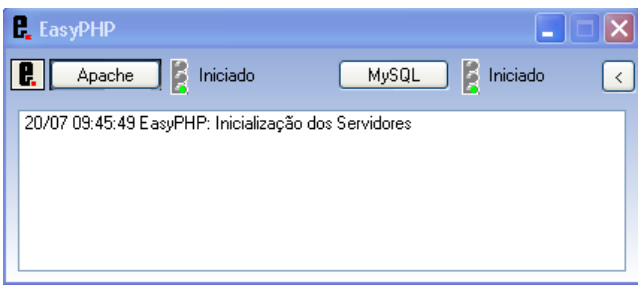


Como é possível perceber, o cliente não tem como visualizar o código do script que criou o HTML.

Instalando o PHP

O PHP está disponível para download no site http://www.php.net em diversos formatos e para diversas plataformas.

No site do PHP, existe documentação detalhada para instalação e configuração do PHP no servidor Web (consultar http://www.php.net/manual/pt_BR/install.windows.php).



Para instalar o PHP num PC pessoal (não servidor Web de uma instituição ou empresa), podemos fazê-lo através de programas instaladores que realizam a instalação de um servidor Web, um banco de dados, o PHP e já os configura. Temos como exemplo desses

instaladores o EasyPHP, o WampServer, entre outros. Consultar em http://en.wikipedia.org/wiki/List_of_AMP_packages uma lista mais completa.

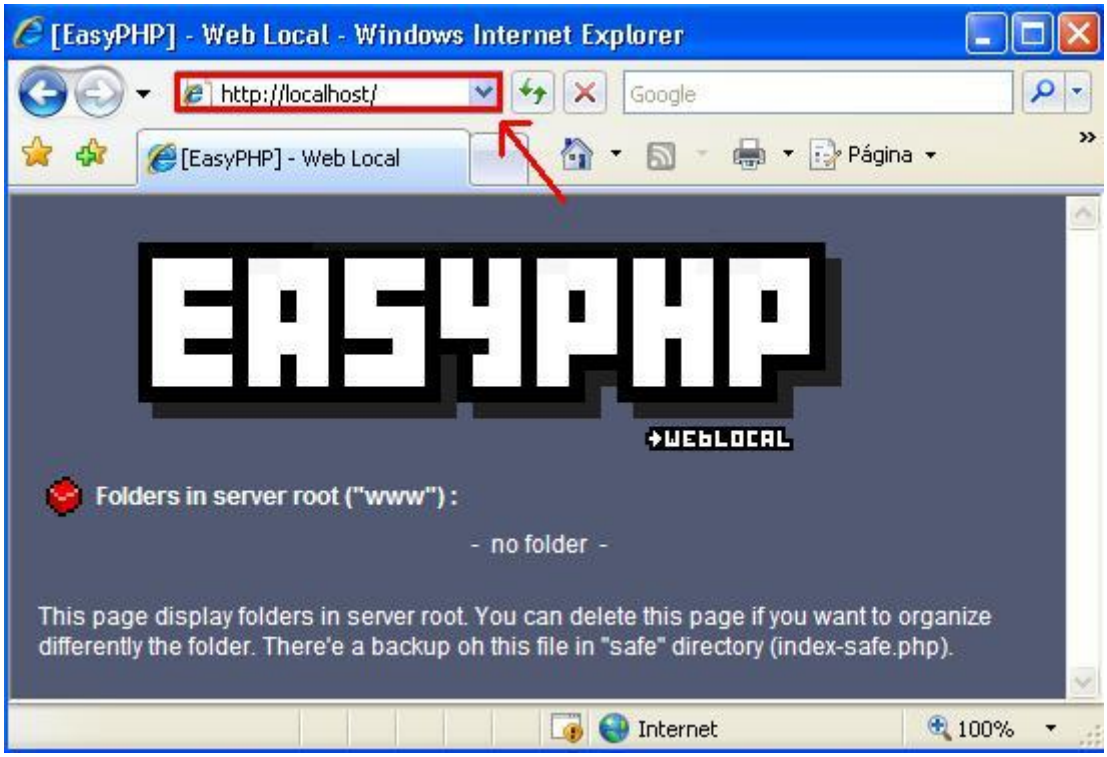
Tanto o EasyPHP como o WampServer, instalam e configuram o servidor Web Apache , o PHP e o banco de dados MySQL. Sua instalação é bastante simples e é aplicada uma configuração básica para os seus serviços.

No caso do EasyPHP, depois de instalado, será exibida a janela abaixo, informando o status dos serviços Apache e MySQL. Para que o PHP seja interpretado, o indicador do Apache deve estar iniciado. O MySQL só será necessário se estiver fazendo uso de bando de dados.

Quando os serviços forem iniciados, no rodapé também aparecerá o ícone referente ao EasyPHP.

Se o EasyPHP será usado apenas para testes, ele não precisará permanecer iniciado o tempo todo. É possível iniciá-lo somente quando for executar páginas com código em HTML.

As páginas em HTML e PHP criados no computador local deverão ser gravadas na pasta **C:\Arquivos de programas\EasyPHP1-8\www** e referenciados no browser como http://localhost/nome-do-seu-arquivo.html.



Exercícios

1. Digitar o código abaixo. Depois salvar em Z: dentro da pasta WEB com o nome **EXEMPLO1.PHP.** Para testar, abrir o navegador (Internet Explorer, Mozilla Firefox, etc) e digitar o endereço http://www2/*u-e-seu-RA*/exemplo1.php (exemplo http://www2/u11000/exemplo1.php) .

```
<?php

phpinfo();

?>
```

Depois de rodar o arquivo acima, responda as questões abaixo:

a) Ao visualizar essa página no navegador, o que aparece?

_____

b) Através desse exemplo, é possível descobrir a versão da linguagem PHP? Se sim, qual é a versão?

_____

2. Digitar o exemplo da página 6. Salvar no Z: dentro da pasta WEB com o nome **EXEMPLO2.PHP.** Para testar, abrir o navegador (Internet Explorer, Mozilla Firefox, etc) e digitar o endereço http://www2/u-e-seu-RA/exemplo2.php.

3. Criar duas páginas em HTML, sendo uma página principal com um link chamando uma página secundária. A página principal deve ter o nome **PRINCIPAL.PHP** e a página secundária deve ter o nome **SECUNDARIA.PHP**. Pode ser qualquer assunto. Tanto a página principal como a página secundária devem iniciar o HTML com a tag <BODY>, ter o corpo da página e finalizar o arquivo com a tag </BODY>. Nas duas páginas, acrescentar o código abaixo no início e no final do arquivo, como indicado abaixo:

```
<?php include("cabecalho.inc.php"); ?>

<body>
.
.
.
</body>

<?php include("rodape.inc.php"); ?>
```

Criar os arquivos CABECALHO.inc.php e RODAPE.inc.php, listados abaixo:

```
// Cabecalho.inc.php

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<title> Primeiro Exemplo de Página Dinâmica </title>
<style type="text/css">
```

```
<!--
body {background-color:#ffffee; margin-left:10px; font-family: verdana;
font-size: 10px}
</style>
</head>
<body>
```

```
// RODAPE.inc.php

<br /><br />
<a href="principal.php">Voltar</a>
<hr />
Colégio Técnico de Campinas <br />
Depto. Processamento de Dados

</body>

</html>
```

Depois de testar os arquivos acima, responda as questões abaixo:

c) Ao visualizar o código-fonte pelo navegador da página principal, a sequência de tags do HTML está correta, ou seja, aparecem as tags <HTML> e </HTML>, assim como as tags <HEAD></HEAD>?

_____

d) Aparece algum texto na barra de título do navegador ao carregar a página principal e a secundária? Se sim, qual o texto que aparece?

_____

_____

e) Foi aplicada alguma formatação de CSS à sua página principal e/ou secundária? Por quê?

_____

_____

Capítulo 2 – Conceitos Básicos

Já comentamos anteriormente que o PHP pode ser inserido num arquivo HTML ou ser um arquivo independente contendo o programa em PHP e chamado por uma (ou várias) página em HTML.

O PHP pode ser inserido num arquivo HTML através das tags abaixo:

```
<?php

?>
```

Exemplo 2.1

As tags indicarão que entre elas haverá comandos da linguagem PHP. Podem ser inseridas em qualquer parte da página HTML ou ainda iniciar e finalizar uma página e todo conteúdo HTML pode ser obtido pelos comandos do PHP.

Todo arquivo que contenha algum código PHP deve ter a extensão do arquivo .PHP.

Exemplo

Como exemplo, vamos criar um arquivo com extensão .PHP. Se estiver usando uma instalação no computador local do Apache (por exemplo, instalado através do EasyPHP) salve-o na pasta **C:\Arquivos de programas\EasyPHP1-8\www.** Senão verifique o local correto para localização desse arquivo para que o servidor Web o interprete.

Observações:

1. Se estiver utilizando o EasyPHP, este deve estar iniciado para que o arquivo abaixo abra normalmente.
2. Se você estiver testando o PHP num computador particular, o arquivo deverá ser gravado na pasta do servidor web (ou do EasyPHP, como descrito acima) e no browser deverá ser colocado o endereço da máquina local http://localhost/exemplo1.html
   Se estiver usando o arquivo em outro local, verificar onde o arquivo deverá ser gravado e qual o nome do servidor web, e no browser usar o endereço do servidor web para testar seu programa: http://servidor-web/exemplo1.html.

```
<HTML>
<HEAD>
<TITLE> Meu primeiro programa PHP </TITLE>
</HEAD>
<BODY>

<?php
   print("Hello, world!<BR><BR>");
   phpinfo();
?>

</BODY>
</HTML>
```

Exemplo 2.2

O resultado do programa acima deve ser algo do tipo:

PHP Version 5.6.23

System	Windows NT VENUS 6.1 build 7601 (Windows Server 2008 R2 Enterprise Edition Service Pack 1) i586
Build Date	Jun 22 2016 12:07:43
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "--disable-isapi" "--disable-nsapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--with-enchant=shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\php-5.6.23-nts-Win32-VC11-x86\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,NTS,VC11
PHP Extension Build	API20131226,NTS,VC11
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar, sqlsrv
Registered Stream Socket Transports	tcp, udp, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	convert.iconv.*, mcrypt.*, mdecrypt.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

Comentários

No PHP, podemos usar // para comentar uma linha apenas, ou /* */ para comentar um bloco de linhas. Exemplo:

```
<html>
<body>

<?php
//Este é um comentário

/*
Este é
um bloco
de comentário
*/
?>

</body>
</html>
```

Exemplo 2.3

Extensão de Arquivos

A seguir são listadas as extensões mais utilizadas para programas em PHP.

Extensão	Significado
.php	Arquivo PHP contendo um programa.
.class.php	Arquivo PHP contendo uma classe.
.inc.php	Arquivo PHP a ser incluído, pode incluir constantes ou configurações.

Incluindo Arquivos

É possível também adicionar PHP ao HTML através de um arquivo separado. Há quatro maneiras de se fazer isso:

- `include("/filepath/filename")`
- `require("/filepath/filename")`
- `include_once("/filepath/filename")`
- `require_once("/filepath/filename")`

As funções acima diferem apenas no tipo de erro. As funções `include()` e `include_once()` somente gerarão um aviso de falha, enquanto que as funções `require()` e `require_once()` causarão um erro fatal e terminarão o script.

A diferença das funções `include_once()` e `require_once()` para as demias é que só aceitarão a inclusão de um arquivo uma única vez pelo PHP. Isso é útil quando se estiver incluindo arquivos com funções do PHP, porque redeclarar funções resulta em erro fatal automático. Em sistemas grandes, às vezes pode ser difícil lembrar se uma função foi incluída ou não. Mas quando usamos o `include_once()` e `require_once()` isso não acontece.

Dependendo da necessidade, pode-se optar para que a função de inserção de arquivos seja mais rigorosa em relação a alguns erros, que podem surgir no código e implicando num erro fatal. Ou executar o script mesmo com alguns erros. A melhor alternativa é `require()`, que travará tudo o que não for perfeito no código. A pior alternativa é o `include_once()`, que ocultará as conseqüências de alguns erros na codificação.

A aplicação mais comum do uso do `include` do PHP é adicionar cabeçalhos e rodapés comuns a todas as páginas Web de um site.

Por exemplo, abaixo temos um arquivo contendo um cabeçalho simples (intuitivamente chamado de `header.inc.php`, ou `cabecalho.inc.php`!):

```
<HTML>
<HEAD>
<TITLE> Exemplo de include! </TITLE>
</HEAD>

<BODY>
```

Exemplo 2.4

Para o rodapé teremos o seguinte arquivo (chamado `footer.inc.php`, ou ainda `rodape.inc.php`):

```
<HR>
```

```
<P>Colégio Técnico de Campinas - COTUCA</P>
</BODY>
</HTML>
```

Exemplo 2.5

Na página de PHP, serão chamados da seguinte forma:

```
<?php require_once($_SERVER['DOCUMENT_ROOT'] . "/cabecalho.inc.php"); ?>

<P> Este é o corpo de uma página em HTML! </P>

<?php require_once($_SERVER['DOCUMENT_ROOT'] . "/rodape.inc.php"); ?>
```

Exemplo 2.6

Ao incluirmos um arquivo, o conteúdo será adicionado como um texto ao arquivo principal. No exemplo acima, foram incluídos partes de HTML (cabeçalho e o rodapé), mas se no arquivo a ser incluído, existir código em PHP, obrigatoriamente deverá ter as tags do PHP válidas (`<?php` e `?>`).

Em relação ao arquivo `database.inc.php` abaixo:

```
$db = mysql_connect('localhost', 'db_user', 'db_password');
mysql_select_db('my_database');
```

Exemplo 2.7

Ao incluirmos o arquivo a partir de um script PHP, as variáveis de banco de dados seriam visíveis como textos. Para corrigir esse problema, devemos acrescentar as tags do PHP:

```
<?php
$db = mysql_connect('locahost', 'db_user', 'db_password');
mysql_select_db('my_database');
?>
```

Exemplo 2.8

Observe e teste os exemplos abaixo:

hello.php

```
<?php
echo "Hello";
?>
```

Exemplo 2.9

teste.php

```
<?php
require('hello.php');
require_once('hello.php');
?>
```

Exemplo 2.10

teste2.php

```
<?php
require('hello.php');
require('hello.php');
?>
```

Exemplo 2.11

teste3.php

```
<?php
require_once('hello.php');
require('hello.php');
?>
```

Exemplo 2.11

Sintaxe Geral

A sintaxe do PHP é bastante parecida com a da linguagem C. Diante disso devemos tomar alguns cuidados ao criarmos os scripts. A seguir destacaremos algumas características:

- ***O PHP não considera mais de um espaço em branco***

O espaço em branco inclui caracteres de espaço, tabulações e CR (*carriage return*, que são caracteres de final de linha). A seqüência de vários caracteres de espaços em branco é considerada pelo PHP como apenas um caractere. Isso é útil, pois em algumas situações não será preciso forçar o PHP a desconsiderar excessos de espaços em branco.

Abaixo temos um exemplo de um código somando 2 + 2 e atribuindo à variável $four de diferentes disposições de espaços em branco:

```
$four = 2 + 2;  // espaços únicos
$four <tab><tab>2<tab><tab>+<tab>2 ;  // espaços e tabulações
$four
2
+
2;   // múltiplas linhas
```

Exemplo 2.12

- ***O PHP às vezes faz distinção entre letras maiúsculas e minúsculas***

Apesar do PHP não ser uma linguagem exigente, em algumas situações ele faz distinção entre letras maiúsculas e minúsculas. No caso das variáveis, essa característica é presente. Veja o exemplo abaixo:

```
<?php
   $capital = 67;
   print("A variável capital é $capital<BR>");
   print("A variável CaPital é $CaPital<BR>");
?>
```

Exemplo 2.13

A saída será:

```
   A variável capital é 67
   A variável CaPital é
```

Exemplo 2.14

No PHP a variável `$capital` é diferente de `$CaPital`. Portanto possuem valores diferentes. Nesse caso, ao executar o script, não aparecerá nenhum erro por estar usando a segunda variável pela primeira vez na função `print()` sem ter tido uma atribuição prévia. No PHP há é possível criar variáveis em qualquer parte do script. Veremos essa característica mais adiante.

Diferentemente de C, nomes de função não fazem distinção de letras maiúsculas e minúsculas bem como a maioria das construções básicas da linguagem (`if`, `then`, `else`, `while`, etc).

- ***Instruções são expressões terminadas por ponto-e-vírgula***

Apesar da orientação para que se use ponto-e-vírgula, o PHP não abortará o script caso não se use. Exemplo:

```
$boasvindas = "Seja bem vindo ao PHP!";
```

Exemplo 2.15

- ***Chaves constituem blocos***

Dentro de alguns comandos, como o `if`, para que várias instruções sejam executadas caso a condição seja verdadeira, devemos usar as chaves para agrupar essas instruções. Exemplo:

```
if (3 == 2 + 1)
   print("Ok, condição verdadeira!<BR>");

if (3 == 2 + 1)
{
   print("Ok,");
   print (" condição verdadeira!<BR>");
}
```

Exemplo 2.16

- ***Comentários***

Um comentário é uma parte do programa que será ignorado na compilação. Um programa deve ter vários comentários a fim de esclarecer a lógica aplicada. O PHP suporta várias maneiras de inclusão de comentários:

- **Comentários multilinha no estilo C**

  Um comentário inicia com o par de caracteres /* e termina com o par */. Exemplo:

```
/* Isto é
   um comentário no
   PHP */
```

Exemplo 2.17

Deve-se tomar cuidado apenas para não aninhar blocos de comentários (colocar /* e */ dentro de um bloco de comentários já aberto por /* e */).

- **Comentários de uma única linha: # e //**

  O PHP também suporta duas formas de comentários no final de uma linha. Exemplo:

```
# Esta é uma linha de comentário
// Este também é um comentário
$a= $b + $c;  # também posso colocar comentários aqui
```

```
$a= $b + $c;  // posso colocar comentários aqui dessa outra forma
```

Exemplo 2.18

Variáveis

Algumas características das variáveis no PHP:

- Todas as variáveis no PHP começam pelo caractere de cifrão ($)
- As variáveis são atribuídas com o operador =
- As variáveis não precisam ser declaradas antes de uma atribuição

Depois do sinal de $, os nomes das variáveis devem ser compostos por letras (maiúsculas ou minúsculas), dígitos (0-9) e caracteres sublinhados (_). O primeiro caractere depois do $ não pode ser um número.

Atribuição de variáveis

A atribuição de variáveis é bastante simples:

```
$nome = "Maria";
$pi = 3 + 0.14159;
```

Exemplo 2.19

No PHP é possível reatribuir uma variável mesmo com valores de tipos diferentes.

```
$my_num = "Isto poderá ser um número";
$my_num = 5;
```

Exemplo 2.20

No PHP é possível utilizar variáveis sem terem sido atribuídas anteriormente. Em outras linguagens, será manipulado um lixo qualquer da memória do computador. No PHP será atribuído um valor padrão e não será exibida nenhuma mensagem de erro.

No PHP as variáveis não têm tipos pré-definidos. Não há como saber o tipo antes de uma atribuição. De acordo com o valor esperado (se será um número ou uma string), a variável receberá valor padrão 0 ou uma string vazia (string com zero caracteres).

Constantes

Constantes são variáveis que permanecerão com seu valor inalterado durante toda a execução do script. Elas possuem escopo global, ou seja, é visível em todos os níveis e funções do script.

As constantes só podem conter valores escalares (boolean, inteiro, ponto flutuante e string). Um valor escalar não pode ser composto de outros valores, como vetores ou objetos.

As constantes não têm um "$" no início do nome e por convenção os nomes de constantes são definidos em letras maiúsculas. Para criar uma constante, usamos a função `define()`. Veja o exemplo abaixo:

```
<?php
define("MINHACONSTANTE", "Hello world.");
echo MINHACONSTANTE; // imprime "Hello world."
?>
```

Exemplo 2.21

Saída

Os scripts PHP só exibirão algum conteúdo caso sejam usadas funções para esse fim. Essa exibição se dará no navegador do usuário.

As duas funções mais comuns para essa função são as funções `echo` e `print`. A função `echo` possui uma estrutura mais simples, aceitando somente um argumento e entende aspas ou parênteses. Exemplo:

```
echo "Este texto será exibido no browser do usuário";
echo ("Este texto será exibido no browser do usuário");
echo "Este texto será exibido", " no browser do usuário";
```

Exemplo 2.22

A função `print` é bem semelhante à função `echo`, mas possui algumas diferenças:

- Aceita somente um argumento
- A função `print` retorna um valor que informa se a instrução foi bem sucedida (retornará 1 se foi bem sucedida e 0 caso contrário)

```
print("3.14159");  //imprime uma string
print(3.14159);    //imprime um número

$animal = "antílope";
$animal_cabeca = 1;
$animal_patas = 4;
print("O $animal possui $animal_cabeca cabeça(s).<BR>");
print("O $animal possui $animal_patas pata(s).<BR>");
```

Exemplo 2.23

Aspas simples e aspas duplas

No PHP o uso das aspas gera comportamentos diferentes na função `print`. No exemplo acima, as variáveis foram substituídas pelo seu conteúdo no momento da exibição. Ao substituir as aspas duplas pelas aspas simples, os nomes das variáveis não serão convertidos pelos seus respectivos valores.

Variáveis Pré-definidas

O PHP oferece varáveis predefinidas de informações vindas do servidor e do browser, que podem ser usadas pelo programador. São as chamadas **superglobais**. A maior parte das supervariáveis consiste em arrays, cujos valores são acessados a partir de chaves (índices) em formato texto, com o nome da informação a ser obtida. As principais supervariáveis em formatos de array do PHP são as seguintes:

$GLOBALS
Contém referências para todas as variáveis que estão disponíveis dentro do script.

$_SERVER
Variáveis criadas pelo servidor web e relacionadas com o ambiente de execução.

$_GET
Variáveis enviadas por script para o servidor via método HTTP GET. O que é enviado pelo formulário é mostrado na barra de endereço.

$_POST
Variáveis enviadas por script para o servidor via método HTTP POST. O que é enviado pelo formulário não é mostrado na barra de endereço do navegador.

$_COOKIE
Variáveis enviadas por script para o servidor via cookies HTTP.

$_FILES
Variáveis enviadas para o script com todas as informações relativa aos arquivos enviados via HTTP, por upload.

$_ENV
Variáveis disponíveis no script do ambiente em execução. Em Web é igual ao $_SERVER.

$_REQUEST
Possui todas as variáveis globais em $_GET, $_POST e $_COOKIE. Em desuso devido à demora na realização do script!

$_SESSION
Variáveis que estão diretamente relacionadas no registro da sessão no script.

$php_errormsg
Exibe a última mensagem de erro ocorrida.

$argc
Número de argumentos passados para o script.

$argv
Array de argumentos passados para o script.

Exercícios

1. Crie uma página em PHP para exibir os valores das variáveis superglobais do PHP.
2. Criar os arquivos abaixo e verificar como se comportam ao executá-los:

hello.php

```
<?php
echo "Hello";
?>
```

teste.php

```
<?php
require('hello.php');
require_once('hello.php');
?>
```

teste2.php

```
<?php
require('hello.php');
require('hello.php');
?>
```

teste3.php

```
<?php
require_once('hello.php');
require('hello.php');
?>
```

3. Testar o exemplo 2.21 e 2.22 e responder as questões abaixo:

   a. Ocorre alguma diferença na execução do programa se trocar a última linha de `echo MinhaConstante; // imprime "Hello world."` por `echo minhamonstante; // imprime "Hello world."`? Se sim, qual a diferença?

   b. No exemplo 2.22, ocorre alguma diferença entre os três comandos `echo`? Se sim, qual a diferença?

   c. No exemplo 2.22, o que acontece se trocar os três comandos `echo` por `print`?

4. Executar o exemplo 2.22 trocando as aspas duplas das funções `print` por aspas simples. Ocorre alguma diferença? Se sim, qual?

Capítulo 3 - Tipos de Dados e Operadores

O PHP possui um sistema de representação de dados na memória bastante simples, isolando o tratamento de baixo nível para que o programador não precise ter esse tipo de preocupação.

O tipo de uma variável não precisa ser declarado previamente. Isso pode ser realizado numa atribuição de valor e o tipo será identificado de acordo com o valor atribuído.

```
$var1 = 55.5;
$var2 = "Bom dia!";
```

Exemplo 3.1

O PHP também pressupõe os tipos dos dados atribuídos. No exemplo abaixo, o tipo do dado atribuído à variável não está explicitamente definido. A função `substr()` espera por três argumentos: uma string, a posição inicial e o tamanho da *substring* que será retornado. Ao passar um número no lugar de uma string, o PHP faz essa conversão automática de modo que a função não retornará um erro por isso. Como o resultado da função é uma string, o tipo da variável `$sub` será string.

```
<?php

$sub = substr(12345,2,2);
print("sub é $sub<BR>");

?>
```

Exemplo 3.2

Veremos agora os tipos de dados suportados pelo PHP, sendo que os cinco primeiros são tipos simples e os dois seguintes são tipos compostos.

Tipo Inteiro

Inteiros representam valores numéricos simples, tanto positivos como negativos. Os inteiros podem ser lidos das bases decimal, octal e hexadecimal. O formato decimal é o padrão, os inteiros octais são representados com um zero no início do número e o hexadecimal com um 0x. Exemplos:

```
<?php
$a = 1234; # número decimal
$b = -123; # um número negativo
$c = 0123; # número octal (equivalente a 83 em decimal)
$d = 0x1A; # número hexadecimal (equivalente a 26 em decimal)
print("Valor de a: $a <BR>");
print("Valor de b: $b <BR>");
```

```
print("Valor de c: $c <BR>");
print("Valor de d: $d <BR>");
?>
```

Exemplo 3.3

O tamanho que um inteiro pode ter vai depender da plataforma. Para a maioria das plataformas mais comuns, o maior número inteiro é de 2^{31}-1 (ou 2.147.483.647) e o menor número inteiro é $-(2^{31} - 1)$.

Tipo Numérico de dupla precisão (doubles)

Números de dupla precisão são números de ponto flutuante e podemos utilizar qualquer uma das sintaxes abaixo:

```
<?php
$a = 1.234;
$b = 1.2e3;   # 1.2 x 10^3
$c = 7E-10;   # 7 x 10^-10
$d = 8e9;
print("Valor de a: $a <BR>");
print("Valor de b: $b <BR>");
print("Valor de c: $c <BR>");
print("Valor de d: $d <BR>");
?>
```

Exemplo 3.4

Tipo Booleano

Tipos booleanos representam valores lógicos verdadeiro ou falso. São usados em construções de controle para teste de uma condição.

O PHP fornece variáveis constantes para utilização como booleanos; TRUE e FALSE, que podem ser usados como no exemplo abaixo:

```
<?php
$var = True; // assimila o valor TRUE para $var

if (TRUE)
   print("Se essa condição for colocada num comando de repetição, resultará num
loop infinito!");

?>
```

Exemplo 3.5

É possível interpretar valores booleanos através de outros conteúdos:

- Se o valor for um número igual a zero, é um valor falso; caso contrário verdadeiro
- Se o valor for uma string vazia (zeros caracteres) ou for a string "0", é um valor falso; caso contrário verdadeiro
- Os valores tipo NULL são sempre falsos
- Se o valor for um tipo composto (um array ou um objeto), ele é falso se não tiver outro valor e, caso contrário, verdadeiro. Para um objeto, *conter um valor* significa ter uma variável de membro que recebeu um valor
- Os recursos válidos são verdadeiros (embora algumas funções que retornem recursos quando são bem-sucedidas retornarão FALSE quando mal sucedidas).

```
$true_num = 3 + 0.14159;
$true_str = "Bom dia!";
```

```
$true_array[49] = "Um elemento do array";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

Exemplo 3.6

Tipo NULL

O valor especial NULL representa que a variável não tem valor. NULL é o único valor possível do tipo NULL. A variável é considerada NULL nos casos:

- ser assimilada com a constante NULL
- ainda não tenha recebido nenhum valor
- ter sido apagada com a função `unset()`

Tipo String

Strings são seqüências de caracteres. Elas podem ser incluídas tanto entre aspas simples como entre aspas duplas, resultando em algumas diferenças na exibição no browser. As strings entre aspas simples são tratadas literalmente, enquanto as strings entre aspas duplas convertem as variáveis pelos seus valores. Exemplo:

```
<?php
echo 'isto é uma string comum <BR>';

echo 'Você pode incluir novas linhas em strings,
dessa maneira que estará
tudo bem <BR>';

// Imprime: Arnold disse uma vez: "I\'ll be back"
echo 'Arnold disse uma vez: "I\'ll be back" <BR>';

// Imprime: Você tem certeza em apagar C:\*.*?
echo 'Você tem certeza em apagar C:\\*.*? <BR>';

// Imprime: Você tem certeza em apagar C:\*.*?
echo 'Você tem certeza em apagar C:\*.*? <BR>';

// Imprime: Isto não será substituído: \n uma nova linha
echo 'Isto não será substituído: \n uma nova linha <BR>';

// Imprime: Variáveis $também não $expandem
echo 'Variáveis $também não $expandem <BR>';?>
```

Exemplo 3.7

Ao usarmos as aspas duplas, todas as variáveis serão convertidas e as seqüências de escape serão convertidas de acordo com seus significados:

Seqüência	**Significado**
\n	Caractere de nova linha
\r	Caractere CD (carriage-return, retorno de carro)
\t	Caractere de tabulação
\$	Sinal de cifrão
\"	Aspa dupla
\\	Barra invertida

Tipo Array

O tipo array permite agrupar valores diferentes e indexá-los por uma seqüência numérica ou por strings. Exemplos:

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

echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

Exemplo 3.8

Em outro capítulo abordaremos de forma mais detalhada o uso de arrays.

Tipo Objeto

Outro tipo de dados do PHP é o objeto. Como um array, um objeto é um tipo composto que permite empacotar outros valores dos dados em uma única unidade. Os objetos têm propriedades adicionais mais complexas, incluindo a capacidade de empacotar funções bem como dados e a capacidade de definir novos tipos de objetos (herança).

Tipo Recurso

Recurso (resource) é uma variável especial que mantém uma referência de recurso externo. Recursos são criados e utilizados por funções especiais, como uma conexão ao banco de dados. Um exemplo é a função mysql_connect(), que, ao conectar-se ao banco de dados, retorna uma variável de referência ao tipo recurso.

Testando tipos

O PHP possui algumas funções a fim de verificarmos o tipo de uma variável. A seguir temos um resumo de algumas funções com essa finalidade:

Função	**Comportamento**
`gettype(arg)`	Retorna uma string identificando o tipo de arg

`is_int(arg)`
`is_integer(arg)`
`is_long(arg)`
	Retorna um valor verdadeiro se arg for um inteiro, e falso, caso contrário

`is_double(arg)`
`is_float(arg)`
`is_real(arg)`
	Retorna um valor verdadeiro se arg for um número de dupla precisão, e falso, caso contrário

`is_bool(arg)`	Retorna um valor verdadeiro se arg for um do tipo booleano, e falso, caso contrário
`is_null(arg)`	Retorna um valor verdadeiro se arg for do tipo NULL, e falso, caso contrário
`is_string(arg)`	Retorna um valor verdadeiro se arg for uma string, e falso, caso contrário
`is_array(arg)`	Retorna um valor verdadeiro se arg for um array, e falso, caso contrário
`is_object(arg)`	Retorna um valor verdadeiro se arg for um objeto, e falso, caso contrário
`is_resource(arg)`	Retorna um valor verdadeiro se arg for um recurso, e falso, caso contrário

Operadores Aritméticos

Na tabela a seguir temos uma lista dos operadores aritméticos suportados pelo PHP:

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 o igual (=). Existem também os operadores combinados compatíveis com todos os operadores aritméticos e os de string. Exemplos:

```
<?php

$a = 3;
$a += 5; // configura $a para 8, como se disséssemos: $a = $a + 5;
$b = "Bom ";
$b .= "Dia!"; // configura $b para "Bom Dia!", como em $b = $b . "Dia!";

?>
```

Exemplo 3.9

Operadores de Incremento/Decremento

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

Exemplo	**Nome**	**Efeito**
`++$a`	Pré-incremento	Incrementa $a em um, e então retorna $a.
`$a++`	Pós-incremento	Retorna $a, e então incrementa $a em um.
`--$a`	Pré-decremento	Decrementa $a em um, e então retorna $a.
`$a--`	Pós-decremento	Retorna $a, e então decrementa $a em um.

```
<?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";
?>
```

Exemplo 3.10

Operadores de String

Há dois operadores de string. O primeiro é o operador de concatenação ('.'), que retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação ('.='), que acrescenta o argumento do lado direito no argumento do lado esquerdo.

```
<?php
$a = "Olá ";
$b = $a . "mundo!"; // agora $b contém "Olá mundo!"

$a = "Olá ";
$a .= "mundo!";     // agora $a contém "Olá mundo!"
?>
```

Exemplo 3.11

Operadores Lógicos

Os operadores lógicos são usados nas combinações de testes lógicos das estruturas de controle. Abaixo temos uma lista dos operadores suportados pelo PHP:

Operador	**Comportamento**
`and`	É verdadeiro se e somente se todos os argumentos forem verdadeiros.
`or`	É verdadeiro se pelo menos um argumento for verdadeiro.
`!`	Negação do resultado de um argumento.
`xor`	É verdadeiro se qualquer um, mas não ambos os argumentos, for verdadeiro.
`&&`	Idêntico ao `and`
`||`	Idêntico ao `or`

Os operadores lógicos apresentam precedência em relação a outros. Na tabela abaixo vemos a lista de operadores (alguns serão comentados futuramente) e sua precedência do maior para o menor:

Associação	**Operador**
não associativo	`new`
direita	`[`
direita	`! ~ ++ -- (int) (float) (string) (array) (object) @`
esquerda	`* / %`
esquerda	`+ - .`
esquerda	`<< >>`
não associativo	`< <= > >=`
não associativo	`== != === !==`
esquerda	`&`
esquerda	`^`
esquerda	`|`
esquerda	`&&`
esquerda	`||`
esquerda	`? :`
direita	`= += -= *= /= .= %= &= |= ^= <<= >>=`
direita	`print`
esquerda	`and`
esquerda	`xor`
esquerda	`or`
esquerda	`,`

Podemos forçar a alteração na ordem da precedência utilizando parênteses. Na expressão 1 + 5 * 3, o resultado será 16. Mas se quisermos mudar a precedência teremos que usar parênteses deixando a expressão assim: (1 + 5) * 3, resultando no valor 18.

É importante saber que os operadores booleanos são associados da esquerda para a direita, sendo que podem não avaliar o segundo argumento caso o primeiro argumento já determine resultado falso para a estrutura de controle. Por exemplo, imagine que seja necessário determinar uma relação aproximada para dois números, mas sem permitir que um possível erro da divisão por zero aconteça. È possível testar primeiro se o denominador não é igual a zero, utilizando o operador != (não igual a):

```
if ($denom != 0 && $number / $denom > 2)
   print("Divisão bem sucedida!");
```

Exemplo 3.12

Quando `$denom` for zero, o operador `&&` deverá retornar falso, independentemente de a segunda expressão ser verdadeira ou falsa, evitando um erro de divisão por zero.

Operadores de Comparação

Abaixo temos uma lista dos operadores de comparação:

Exemplo	Nome	Resultado
`$a == $b`	Igual	Verdadeiro (TRUE) se $a é igual a $b.
`$a === $b`	Idêntico	Verdadeiro (TRUE) 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`	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.

Os operadores de comparação podem ser utilizados para comparar strings e números. Vejamos um exemplo:

```
if (("Marx" < "Mary")) and (("Mary" < "Marzipan"))
{
   print("Between Marx and Marzipan in the");
   print("dictionary, there was Mary.<BR>");
}
```

Exemplo 3.13

As comparações fazem distinção de maiúsculas e minúsculas. No exemplo acima, todas as strings começam com uma letra em maiúsculo.

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 $b tem os mesmos elementos na mesma ordem.
`$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.

Exercícios

1. Execute o exemplo 3.7. Qual o efeito real do caracter /n?
2. Troque as aspas do exemplo 3.7 por aspas duplas. Ocorre alguma diferença?
3. Execute o exemplo 3.10 e anote o que é exibido no browser.
4. A que operadores equivalem && e || ?
5. Qual a diferença dos operadores =, == e === ?

Capítulo 4 – Comandos Condicionais e de Repetição

Estudaremos nesse capítulo dois tipos de estruturas de controle: os *desvios* e *repetição*. Um desvio é uma bifurcação no caminho da execução de um programa. O loop é um tipo especial de desvio no qual um dos caminhos de execução volta ao início do desvio, repete o teste e, possivelmente, o corpo do loop.

Estruturas de Desvio

As estruturas mais importantes de desvio são o `if` e o `switch`. Veremos a seguir as características de cada uma delas.

If-else

A sintaxe para if é:

```
if (teste)
   instrução-1

if (teste)
   instrução-1
else
   instrução-2
```

Vejamos alguns exemplos:

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

Exemplo 4.1

Abaixo, temos o exemplo que um bloco de comando dentro da estrutura do if. Nesse caso, devemos identificar esse bloco através das chaves `{ }`:

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

Exemplo 4.2

Ao necessitarmos de um comportamento caso a condição principal falhe, podemos usar o `else` como no exemplo abaixo:

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

Exemplo 4.3

Também temos a opção `elseif`, para analisar uma condição alternativa:

```
<?php
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";
}
?>
```

Exemplo 4.4

Os próximos dois exemplos são logicamente iguais, mas com o uso do `elseif` a leitura do script se torna mais simplificada:

```
if ($day == 5)
   print("Quinta-feira");
else
   if ($day == 4)
      print("Quarta-feira");
   else
      if ($day == 3)
         print("Terça-feira");
      else
         if ($day == 2)
            print("Segunda-feira");
         else
            if ($day == 1)
               print("Domingo");
```

Exemplo 4.5

```
if ($day == 5)
   print("Quinta-feira");
elseif ($day == 4)
   print("Quarta-feira");
elseif ($day == 3)
   print("Terça-feira");
elseif ($day == 2)
   print("Segunda-feira");
elseif ($day == 1)
   print("Domingo");
```

Exemplo 4.6

Switch

O switch possui o mesmo comportamento do if-else. Porém ele é bastante útil quando há necessidade de comparar uma única variável com vários valores. A sintaxe é a seguinte:

```
switch (expressão)
{
   case valor-1:
      instrução-1;
      [break;]

   case valor-2:
      instrução-2;
      [break;]
```

```
    ...
    [default:
       instrução-padrão;]
}
```

Vejamos alguns exemplos:

```
switch ($day)
{
   case 5:
      print("Quinta-feira");
      break;
   case 4:
      print("Quarta-feira");
      break;
   case 3:
      print("Terça-feira");
      break;
   case 2:
      print("Segunda-feira");
      break;
   case 1:
      print("Domingo");
      break;
}
```

Exemplo 4.7

Exemplo com `default`:

```
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";
}
```

Exemplo 4.8

Estruturas de Repetição (Loops)

Existem dois tipos de loop: os limitados e os não-limitados. Os loops limitados executarão um número fixo de vezes. Os loops não-limitados são aqueles que se repetem até que uma condição se torne verdadeira ou falsa, e essa condição depende da ação do código dentro do loop.

While

O `while` é a estrutura de repetição mais simples do PHP. Sua sintaxe é a seguinte:

```
while (condição)
   instrução
```

O `while` avalia a expressão *condição* como um booleano, ou seja, se for verdadeiro, executa a *instrução* e então volta a avaliar a *condição*. Se a *condição* for falsa, o loop é interrompido. Se o corpo do while for formado por um bloco de instruções devemos acrescentar as chaves indicando o início e o término do bloco. Exemplo:

```
<?php

$i = 0;
while ($i <= 10){
  echo $i;
}
echo "<BR> Fim do script!";

?>
```

Exemplo 4.9

Do-while

O `do-while` é bastante parecido com o `while`, porém o teste ocorrerá no final do loop. Assim sendo, a instrução será executada pelo menos uma vez. A sintaxe é:

```
do instrução
   while (expressão);
```

Exemplo:

```
<?php

$i = 0;
do {
  echo $i;
} while ($i <= 10);
echo "<BR> Fim do script!";

?>
```

Exemplo 4.10

For

O `for` é uma estrutura mais complexa. Sua sintaxe é a seguinte:

```
 for (expressão-inicial; verificação-de-término; expressão-de-final-do-loop)
   instrução
```

A *expressão-inicial* é uma expressão que será executada pelo `for` para depois ser comparada através da *verificação-de-término*. A cada repetição a *expressão-de-final-do-loop* será executada.

```
<?php

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

Exemplo 4.11

Foreach

O comando `foreach` é uma estrutura que irá comparar elementos de um array.

```
foreach ($array as $value)
 {
 instrução
 }
```

Em cada *loop*, cada elemento do array será atribuído à variável declarada no `foreach`. Exemplo:

```
<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>
```

Exemplo 4.12

Break e Continue

Normalmente um loop é interrompido quando a condição deixa de ser verdadeira. Os comandos `break` e `continue` são opções para quebra do loop.

No exemplo abaixo vemos que o `break` aborta o comando de repetição:

```
<?php

$i = 0;
while ($i <= 10){
  echo $i;
  if ($i == 5)
     break;
}
echo "<BR> Fim do script!";

?>
```

Exemplo 4.13

O comando continue ignorará as instruções seguintes dentro de uma estrutura de repetição, reiniciando o loop. Exemplo:

```
<?php
  for ($i = 0; $i < 5; ++$i) {
      if ($i == 2)
          continue;
      print ("$i<BR>");
  }
?>
```

Exemplo 4.14

Tanto o `break` como o `continue` aceitam um argumento numérico que indica qual estrutura se aplica o comando.

Exercícios

1. Execute o exemplo abaixo:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

2. Se tirarmos o ponto-e-vírgula depois do comando `continue` do exemplo 4.14, o que acontecerá? Por quê?

_____

Capítulo 5 – Funções

Funções definidas pelo Usuário

As funções podem ser definidas de acordo com a sintaxe abaixo:

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

```
<?php
function exemplo ($arg_1, $arg_2, $arg_n)
{
    echo "Exemplo de função.\n";
    return $valor_retornado;
}
?>
```

Exemplo 5.1

Dentro de uma função podemos inserir qualquer código do PHP, bem como criação de outras funções e classes.

```
<?php
function exemplo1()
{
  function exemplo2()
  {
    echo "Eu não existo até exemplo1() ser chamada.\n";
  }
}

// Nós não podemos chamar exemplo2() ainda porque ela ainda não foi definida.

exemplo1();

/* Agora nós podemos chamar exemplo2(), porque o processamento de exemplo1()
tornou a primeira acessível */

exemplo2();

?>
```

Exemplo 5.2

Os nomes das funções seguem o mesmo padrão de nomeação dos demais recursos do PHP: compostos por letras (maiúsculas ou minúsculas), dígitos (0-9) e caracteres sublinhados (_). Todas as funções possuem escopo global, isso implica em que uma vez criada a função, é possível executá-la em qualquer parte do script.

Os tipos das funções e dos parâmetros não precisam ser declarados, uma vez que o PHP não obriga a declaração de tipos. O cuidado que é necessário ter é no momento em que os parâmetros serão usados dentro da função (se a operação corresponde ao valor passado pela função) e em relação ao tipo do valor retornado (se a função retornar valor, se esse valor corresponde ao processamento fora da função). De qualquer modo, é importante documentar o programa, de forma a esclarecer os tipos dos valores passados e retornado a fim de evitar erros durante a execução do script.

Passagem de parâmetros

Os parâmetros são passados para as funções através de uma lista de variáveis e/ou constantes delimitados por vírgulas.

O PHP suporta a passagem de parâmetros por valor (o padrão), por referência e valores padrão de parâmetros. Listas de parâmetros de comprimento variável são suportadas apenas no PHP4 e posterior que veremos adiante.

Quando um parâmetro é passado por valor, o conteúdo é trabalhado dentro da função, mas a variável que continha o valor antes do processamento da função não sofre nenhuma alteração. Exemplo:

```
<?php
$str = 'Isto é uma string,';
function concatena($string)
{
    $string .= ' e alguma coisa mais.';
}
concatena($str);
echo $str;    // imprime 'Isto é uma string,'
?>
```

Exemplo 5.3

Quando passamos um parâmetro por referência, o conteúdo da variável é alterado pela função permanentemente. Ou seja, é passado para a função o endereço (a referência) do conteúdo da variável na memória do computador. Portanto, a função estará alterando diretamente na memória onde está armazenada aquela variável. Quando passamos o parâmetro por valor, na realidade passamos uma cópia do conteúdo para a função. Exemplo:

```
<?php
function add_some_extra(&$string)
{
    $string .= ' e alguma coisa mais.';
}
$str = 'Isto é uma string,';
add_some_extra($str);
echo $str;    // imprime 'Isto é uma string, e alguma coisa mais.'
?>
```

Exemplo 5.4

Também podemos definir valores padrão no estilo C++, como mostra o exemplo:

```
<?php
function cafeteira ($tipo = "cappuccino")
{
    return "Fazendo uma xícara de café $tipo.\n";
}
echo cafeteira ();
echo cafeteira ("expresso");
?>
```

Exemplo 5.5

A saída do script acima será:

```
Fazendo uma xícara de café cappucino.
Fazendo uma xícara de café expresso.
```

Exemplo 5.6

Se uma função possuir parâmetros normais e parâmetros com valor padrão, é necessário que os parâmetros com valor padrão sejam definidos após os parâmetros sem valor padrão. Compare os exemplos abaixo:

```
<?php
function calendario ($mes = "Agosto", $ano)
{
    return "Estamos em $mes de $ano.\n";
}
echo calendario ("2007");   // não funciona como esperado
?>
```

Exemplo 5.7

```
<?php
function calendario ($ano, $mes = "Agosto")
{
    return "Estamos em $mes de $ano.\n";
}
echo calendario ("2007");
?>
```

Exemplo 5.8

O PHP permite que a quantidade de parâmetros de uma função seja variável. Para isso usamos as funções `func_num_args()`, `func_get_arg()` e `func_get_args()`.

A função `func_num_args()` retornará o número de parâmetro passados para uma função. Exemplo:

```
<?php
function exemplo() {
    $num_par = func_num_args();
    echo "Número de parâmetros: $num_par\n";
}
exemplo (1, 2, 3, "oi");
?>
```

Exemplo 5.9

A função `func_get_arg()` retornará um parâmetro, de acordo com a referência passada para a função começando em 0 para o primeiro parâmetro. Exemplo:

```
<?php
function exemplo() {
     $num_par = func_num_args();
     echo "Número de parâmetros: $num_par<br>\n";
     if ($num_par >= 2) {
     echo "O segundo parâmetro é: " . func_get_arg (1) . "<br>\n";
     }
}
exemplo(1, 2, 3, "oi");
?>
```

Exemplo 5.10

A função `func_get_args()` retornará um array contendo os parâmetros passados para a função. Exemplo:

```
<?php
function exemplo() {
    $num_par = func_num_args();
```

```
    echo "Número de parâmetros: $num_par<br>\n";
    if ($num_par >= 2) {
    echo "O segundo parâmetro é: " . func_get_arg (1) . "<br>\n";
    }
    $par_list = func_get_args();
    for ($i = 0; $i < $num_par; $i++) {
    echo "Parâmetro $i é: " . $par_list[$i] . "<br>\n";
    }
}
exemplo(1, 2, 3, "oi");
?>
```

Exemplo 5.11

Retornando Valores

Como mencionado anteriormente, uma função pode retornar valores. Para isso usamos o comando `return`. Os valores retornados podem ser de qualquer tipo, inclusive arrays e objetos. Exemplo:

```
<?php
function quadrado ($num)
{
    return $num * $num;
}
echo quadrado (4);
?>
```

Exemplo 5.12

Para retornar vários valores, devemos usar arrays. Exemplo:

```
<?php
function numeros_pequenos()
{
    return array (0, 1, 2);
}
list ($zero, $um, $dois) = numeros_pequenos();
?>
```

Exemplo 5.13

Exercícios

1. Crie um script em PHP para validar número de CPF. O Segue o algoritmo para verificação:

```
function ValidarCPF(cpf: int[11])
    var v: int[2]
    //Nota: Calcula o primeiro dígito de verificação.
    v1 := 10×cpf1 + 9×cpf2 + 8×cpf3
    v1 +=  7×cpf4 + 6×cpf5 + 5×cpf6
    v1 +=  4×cpf7 + 3×cpf8 + 2×cpf9
    v1 := 11 - v1 mod 11
    v1 := 0 if v1 ≥ 10
    //Nota: Calcula o segundo dígito de verificação.
    v2 := 11×cpf1 + 10×cpf2 + 9×cpf3
    v2 +=  8×cpf4 +  7×cpf5 + 6×cpf6
    v2 +=  5×cpf7 +  4×cpf8 + 3×cpf9
    v2 +=  2×v1
    v2 := 11 - v2 mod 11
    v2 := 0 if v2 ≥ 10
    //Nota: Verdadeiro se os dígitos de verificação são os esperados.
    return v1 = cpf10 and v2 = cpf11
```

Capítulo 6 – Passagem de Informações entre Páginas

É importante termos em mente que as páginas HTML são carregadas independentemente umas das outras. Mesmo quando uma página é acessada por um link, nenhuma informação da página origem, ou seja, nenhuma informação de estado é passada para a página seguinte apenas pelo processo de abrir o HTML.

O HTML por si só não possui a característica de passar informações entre páginas. Isso só é possível através do uso de programas para fazer com que as informações sejam passadas entre páginas. Veremos que o PHP é uma linguagem bastante eficiente para esse caso.

Basicamente, temos duas técnicas de passagem de informações entre páginas Web, que utilizam os métodos GET e POST em HTML.

Argumentos GET

O método GET passa argumentos para uma página inserindo-os na URL (Uniform Resource Locator). Quando utilizado para tratamento de formulário, GET acrescenta o nome indicado de variáveis e valores ao URL designado no atributo ACTION com um separador de ponto de interrogação e envia a tudo para o agente de processamento, que é, neste caso, um servidor Web. Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo do método GET</TITLE>
</HEAD>
<BODY>

<FORM ACTION=http://localhost/futebol.php METHOD=”GET”>
<P>Selecione o clube campeão da Copa Libertadores da América:<BR>

<SELECT NAME=Time SIZE=3>
   <OPTION VALUE=Gremio>GRÊMIO</OPTION>
   <OPTION VALUE=SaoPaulo>SÃO PAULO</OPTION>
   <OPTION VALUE=Corinthians>CORINTHIANS</OPTION>
</SELECT>

<P>Digite seu nome: <INPUT TYPE=text NAME=Nome>
<P><INPUT TYPE=submit NAME=Enviar VALUE=Seleciona></P>
</FORM>
</BODY>
</HTML>
```

Exemplo 6.1

Quando o usuário faz uma seleção e clica no botão **Seleciona**, o navegador agrupa esses elementos na seguinte ordem:

- A URL entre aspas depois da palavra ACTION (”`http://localhost/futebol.php`”)
- Um ponto de interrogação (?) que significa que os caracteres a seguir foram passados através do método GET
- Uma variável e um valor (Time=Corinthians)
- Um “&” seguido de outro par variável+valor (isso se repetirá para todos os campos preenchidos no formulário)

O navegador construirá a seguinte URL:
http://localhost/”http://localhost/futebol.php”?Time=Corinthians&Nome=Maria&Enviar=Seleciona

O script PHP `futebol.php` receberá os dados do formulário e realizará o processamento devido. Exemplo:

```
<HTML>

<HEAD>
<TITLE>futebol.php</TITLE>
<STYLE TYPE="text/css">
<!--
BODY {font-size:24pt; }
-->
</style>
</HEAD>

<BODY>

<P>Parabéns,
<?php echo $_GET['Time']; ?>

<P>Valeu a torcida,
<?php echo $_GET['Nome']; ?>


</BODY>
</HTML>
```

Exemplo 6.2

Uma vantagem do método GET em relação ao POST é que uma vez tendo a URL formada a partir do formulário, a página é possível ser acessada sem ter que preencher o formulário novamente, mas apenas passando a URL completa com os respectivos campos.

As desvantagens do GET são mais substanciais, tornando-o obsoleto de acordo com a especificação de HTML 4.0. São elas:

- O método GET não é adequado para logins por expor o nome de usuário e a senha
- Cada envio de GET é registrado no log de servidor Web, incluindo o conjunto de dados
- Como o método GET atribui dados a uma variável de ambiente de servidor, o comprimento da URL é limitado

Depois de muitas indagações sobre o uso do GET, recomenda-se sua utilização em situações onde não haverá nenhum tipo de problema ao passar dados através da URL. Resumindo, o caso mais apropriado para o uso do GET é para caixa de pesquisa. Para os demais casos, recomenda-se o uso do método POST.

Argumentos POST

POST é o método de envio de formulário preferido, particularmente em usos que resultarão em alterações permanentes, como adicionar informações a um banco de dados.

Vantagens do método POST:

- É mais seguro que GET porque informações inseridas pelo usuário nunca são visíveis na string de consulta da URL, nos logs do servidor, ou na telam se forem tomadas certas precauções, como sempre utilizar o tipo de entrada HTML `password` para senhas.
- Há um limite muito maior na quantidade de dados que pode ser passada (cerca de 2 kbytes em vez de aproximadamente 200 caracteres).

Desvantagens do método POST:

- Os resultados em um determinado momento não podem ser marcados

- Esse método pode ser incompatível com vertas configurações de firewall, o que elimina os dados de formulários como uma medida de segurança.

Formatando Variáveis de Formulário

O PHP atribui automaticamente as variáveis correspondentes aos dados inseridos por um formulário HTML através dos métodos GET ou POST. Para isso, é importante nomear claramente cada INPUT através do atributo NAME, pois esse atributo será o nome da variável no PHP. Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo</TITLE>
</HEAD>
<BODY>

<FORM ACTION=”http://localhost/inclusao.php” METHOD=”POST”>
<P>Digite seus dados:<BR>
<P>Nome: <INPUT TYPE="text" NAME="Nome">
<P>Idade: <INPUT TYPE="text" NAME="Idade">
<P><INPUT TYPE="submit" NAME="Enviar" VALUE="Seleciona"></P>
</FORM>

</BODY>
</HTML>
```

Exemplo 6.3

No formulário acima teremos dois campos INPUT, com os atributos NAME `Nome` e `Idade`, respectivamente. No PHP essas variáveis serão acessadas como $_POST[‘Nome’] e $_POST[‘Idade’]. No exemplo abaixo, é exibido o conteúdo da variável $_POST de acordo com o formulário:

```
<?php
if (isset($_POST['action']) && $_POST['action'] == 'submitted') {
    echo '<pre>';
    print_r($_POST);
    echo '<a href="'. $_SERVER['PHP_SELF'] .'">Tente de novo</a>';

    echo '</pre>';
} else {
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
    Nome:  <input type="text" name="personal[name]" /><br />
    Email: <input type="text" name="personal[email]" /><br />
    Cerveja: <br />
    <select multiple name="beer[]">
        <option value="antartica">Antartica</option>
        <option value="brahma">Brahma</option>
        <option value="skol">Skol</option>
    </select><br />
    <input type="hidden" name="action" value="submitted" />
    <input type="submit" name="submit" value="Enviar dados!" />
</form>
<?php
}
?>
```

Exemplo 6.4

Veja o resultado:

```
Array
(
    [personal] => Array
        (
            [name] => pat
            [email] => pat
        )

    [beer] => Array
        (
            [0] => antartica
        )

    [action] => submitted
    [submit] => Enviar dados!
)
Tente de novo
```

Exemplo 6.5

Exercícios

1. Teste o exemplo 6.1 e 6.2. Depois de clicar no botão SELECIONA, o que significam na URL os valores "?" e "&"?

   _____

   _____

2. O que se referem as expressões `$_GET['Time']` e `$_GET['Nome']` do exemplo 6.2?

   _____

   _____

3. Testar o exemplo 6.4 e responder as questões abaixo:

   a. Para que serve a função `isset()` usada no exemplo 6.4?

   _____

   b. Explique o que acontece na linha `<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">` do exemplo 6.4?

   _____

   _____

Capítulo 7 - Strings

Vimos anteriormente que as strings são seqüências de caracteres que podem ser atribuídas a variáveis, manipuladas em funções, enviadas para uma página Web, etc. As strings são especificadas através de aspas (simples ‘’ ou duplas “”).

Vimos também que a diferença entre as aspas é em relação ao comportamento do PHP ao exibir uma cadeia de caracteres. No caso das aspas simples, os nomes das variáveis serão exibidos e não seus conteúdos. E caracteres especiais também poderão ser exibidos de acordo com a sequência de caracteres passada. Veja o resultado do exemplo abaixo:

```
<?php
echo 'isto é uma string comum <BR>';

echo 'Você pode incluir novas linhas em strings,
dessa maneira que estará
tudo bem <BR>';

// Imprime: Arnold disse uma vez: "I\'ll be back"
echo 'Arnold disse uma vez: "I\'ll be back" <BR>';

// Imprime: Você tem certeza em apagar C:\*.*?
echo 'Você tem certeza em apagar C:\\*.*? <BR>';

// Imprime: Você tem certeza em apagar C:\*.*?
echo 'Você tem certeza em apagar C:\*.*? <BR>';

// Imprime: Isto não será substituído: \n uma nova linha
echo 'Isto não será substituído: \n uma nova linha <BR>';

// Imprime: Variáveis $também não $expandem
echo 'Variáveis $também não $expandem';
?>
```

Exemplo 7.1

O resultado será:

```
isto é uma string comum
Você pode incluir novas linhas em strings, dessa maneira que estará tudo bem
Arnold once said: "I'll be back"
Você tem certeza em apagar C:\*.*?
Você tem certeza em apagar C:\*.*?
Isto não será substituído: \n uma nova linha
Variáveis $também não $expandem
```

Exemplo 7.2

Caracteres e Índices de Strings

No PHP é possível recuperar caracteres individuais de uma string incluindo o número do caracter, iniciando em 0 e incluindo entre chaves seguindo imediatamente uma variável alfanumérica. Exemplo:

```
<?php

$my_string = "Bom dia!";
for ($index=0; $index<8; $index++) {
   $string = $my_string{$index};
   print("$string$string"); }
```

```
?>
```

Exemplo 7.3

O resultado será:

```
BBoomm ddiiaa!!
```

Exemplo 7.4

Concatenação e Atribuição

Para concatenar strings usamos o ponto (.), como no exemplo abaixo:

```
<?php
   $str1 = "Bom";
   $str2=" Dia";
   echo $str1.$str2;
?>
```

Exemplo 7.5

Assim como em operações aritméticas, o PHP tem um operador abreviado que combina concatenação e atribuição:

```
<?php
   $str1 = "Bom";
   $str1.=" Dia";
   echo $str1;
?>
```

Exemplo 7.6

Sintaxe heredoc

O PHP oferece outra maneira de especificar uma string através da sintaxe *heredoc*. Essa maneira é útil para usar strings com textos grandes. O operador para identificar o início do trecho heredoc é <<< seguido de uma identificação. O texto será finalizado quando encontrar novamente a mesma identificação do início do trecho. Exemplo:

```
<?php
$str1 = <<<EOT
Essa é uma string
que contém várias linhas e não
preciso me preocupar com aspas... <BR>
<a href="http://localhost/exemplo.php">Exemplo de link!</a>
EOT;
echo $str1;
?>
```

Exemplo 7.7

O identificador de finalização EOT não deve ser recuado, caso contrário o PHP não o interpretará como final da string. Aqui usamos a expressão EOT como identificadores, mas é possível usar qualquer termo para esse fim, desde que sejam respeitadas as regras para nomeação de variáveis.

```
<?php
echo <<<FORM
<form action="" method="post" name="form1">
      Campo texto: <input type="text" name="texto"> <br>
      Campo de senha: <input type="password" name="senha"> <br>
```

```
      Campo de arquivo: <input type="file" name="arquivo"> <br>
      <input type="checkbox" name="opcao1" value="ok"> Check <br>
      <input type="radio" name="opcao2" value="ok"> Radio <br>
      <input type="submit"  value="Enviar">  <br>
      <input type="reset"  value="Limpar"> <br>
      <input type="button" name="Verificar" value="Botão"
onclick="verifica(form1)"> <br>
</form>
FORM;
?>
```

Exemplo 7.8

Funções de Limpeza de String

Essas funções são usadas para remover espaços em branco no final, no início, em ambos os lados, do único argumento de string.

Função	***Descrição***
`chop()`	Idêntico ao `rtrim()`
`ltrim()`	Retira espaços em branco do início da string
`rtrim()`	Retira espaços em branco do final da string
`trim()`	Retira espaços em branco do início e final da string

Além de espaços em branco, essas funções também removem seqüências de escape \n, \r, \t e \0 (caracteres de fim de linha, tabulações e caracteres nulos utilizados para encerrar uma string em programas em C).

Funções de Substituição de String

Veremos agora algumas funções de substituição e manipulação de string :

Função	***Descrição***	***Sintaxe***
`substr()`	Retorna parte de uma string	`substr("abc",1,2);`
`str_replace()`	Substitui todas as ocorrências da string de procura com a string de substituição	`str_replace("tarde","noite","Boa tarde!");`
`substr_replace()`	Substitui o texto dentro de uma parte de uma string	`substr_replace($var,'',4,-1)`
`strrev()`	Inverte uma string	`strrev("Bom dia!");`
`str_repeat()`	Repete uma string	`str_repeat("-=", 10);`

Exemplos:

```
<?php

// substr()
$str1 = substr("abcdef", 1);     // retorna "bcdef"
$str2 = substr("abcdef", 1, 3); // retorna "bcd"

print ("Primeira string: " . $str1 . "<BR>");
print ("Segunda string: " . $str2 . "<BR>");

//str_replace()
$string = str_replace("tarde","noite","Boa tarde!");
print ("A função retorna: " . $string . "<BR>");
```

```
//substr_replace()
$var = 'Bom dia!';

/* 4 é posição de início para substituição até o fim de $var */
print ( substr_replace($var, '', 4) . "<BR>");

/* 4 é posição de início para substituição e -1 indica
   a primeira permanecer a primeira posição após a substituição*/
print ( substr_replace($var, '', 4, -1) . "<BR>");

/* Retira 'Bom' de $var */
print ( substr_replace($var, '', 0, -4) . "<BR>" );


// strrev()
$var = 'Bom dia!';

// Exibe "!aid moB"
print ( strrev($var) . "<BR>" );

// str_repeat();
echo str_repeat("-=", 10); // Exibe "-=-=-=-=-=-=-=-=-=-=".

?>
```

Exemplo 7.10

Funções de Conversão de Letras Maiúsculas e Minúsculas

As funções abaixo alteram letras maiúsculas em minúsculas e vice-versa. Assim como as anteriores, retornam uma string com as modificações desejadas.

Função	***Descrição***	***Sintaxe***
`strtolower()`	Converte uma string para minúsculas	`strtolower($str);`
`strtoupper()`	Converte uma string para maiúsculas	`strtoupper($str);`
`ucfirst()`	Converte para maiúscula o primeiro caractere de uma string	`ucfirst($str);`
`ucwords()`	Converte para maiúsculas o primeiro caractere de cada palavra	`ucwords($str);`

Impressão e Saída

Até agora estudamos as construções básicas para impressão e saída sendo o `print` e o `echo`. Vimos que para imprimir o conteúdo de variáveis, basta acrescentá-las em uma string entre aspas duplas, fazendo a interpolação de seus valores.

O PHP possui outras funções de impressão e saída que são o `printf()` e `sprintf()`. São modeladas com base nas funções da linguagem C do mesmo nome. As duas funções recebem argumentos idênticos: uma string de formato especial e, em seguida, qualquer número de outros argumentos que serão interpolados nos lugares certos no formato de string para gerar o resultado.

A única diferença entre o `printf()` e `sprintf()` é que o `printf()` envia a string resultante diretamente para a saída, enquanto que `sprintf()` retira o resultado da string como o seu valor.

A parte complicada dessas funções é a string de formato. Todo caractere aparecerá literalmente no resultado, exceto o caractere % e os caracteres que imediatamente o seguem. O caractere % sinaliza o

início de uma "especificação de conversão", que indica como imprimir um dos argumentos que acompanham a string de formato.

Depois do %, existem 5 elementos que compõem a especificação de conversão, sendo alguns opcionais: preenchimento, alinhamento, largura mínima, precisão e tipo.

- Um especificador de *preenchimento* opcional que diz qual caractere será usado para preencher o resultado para o tamanho certo. Isto pode ser um espaço ou 0 (zero). O padrão é preencher com espaços. Um caractere alternativo de preenchimento pode ser especificado colocando uma aspa simples (') antes.

- Um especificador de alinhamento opcional que diz se o resultado deve ser alinhado a esquerda ou a direita. O padrão é alinhar a direita; um caractere (-) fará com que seja alinhado a esquerda.

- Um número opcional de *largura mínima*, um especificador de tamanho que diz quantos caracteres (mínimo) deve resultar desta conversão.

- Um especificador de precisão opcional, que é o ponto (.) seguido por um número, que diz quantos dígitos decimais devem ser exibidos para números de ponto flutuante. Esta opção não tem efeito para outros tipos que não sejam float. (Outra função útil para formatar números é number_format().)

- Um *especificador de tipo* que indica o tipo do valor a ser interpretado. Os tipos possíveis são:

    - *%* - Um caractere porcento. Não é requerido neenhum argumento.
    - **b** - O argumento é tratado com um inteiro, e mostrado como um binário.
    - **c** - O argumento é tratado como um inteiro, e mostrado como o caractere ASCII correspondente.
    - **d** - O argumento é tratado como um inteiro, e mostrado como um número decimal com sinal.
    - **u** - O argumento é tratado com um inteiro, e mostrado como um número decimal sem sinal.
    - f - O argumento é tratado como um float, e mostrado como um número de ponto flutuante.
    - **o** - O argumento é tratado com um inteiro, e mostrado como un número octal.
    - **s** - O argumento é tratado e mostrado como uma string.
    - **x** - O argumento é tratado como um inteiro, e mostrado como um número hexadecimal (com as letras minúsculas).
    - **X** - O argumento é tratado como um inteiro, e mostrado como um número hexadecimal (com as letras maiúsculas).

```
<pre>
<?php
$var = 3.14159;
printf("%f, %10f, %-010f, %2.2f\n", $var, $var, $var, $var);
?>
</pre>
```

Exemplo 7.11

A saída deverá ser:

```
3.141590,   3.141590, 3.14159000, 3.14
```

Funções de String

Na tabela abaixo listamos algumas funções para manipulação de strings.

Função	Descrição

addcslashes ()	Retorna uma string com barras invertidas na frente dos caracteres especificados. **Sintaxe:** addcslashes(string,characters)
addslashes ()	Retorna uma string com barras invertidas na frente de personagens pré-definidos
bin2hex ()	Converte uma seqüência de caracteres ASCII com valores hexadecimais
chop ()	Alias de rtrim ()
chr ()	Retorna um caractere de um valor ASCII especificado
chunk_split ()	Separa uma string em uma série de partes menores
convert_cyr_string ()	Converte uma string de um conjunto de caracteres cirílico para outro
convert_uudecode ()	Decodifica uma string uuencoded
convert_uuencode ()	Codifica uma string usando o algoritmo uuencode
count_chars ()	Retorna quantas vezes um caractere ASCII ocorre dentro de uma seqüência e retorna a informação
crc32 ()	Calcula um CRC de 32 bits para uma string
crypt ()	Uma forma de codificação de string (hash)
echo ()	Strings saídas
explode ()	Quebra uma string em um array
fprintf ()	Escreve uma string formatada para um fluxo de saída especificado
get_html_translation_table ()	Retorna a tabela de tradução usada por htmlspecialchars () e htmlentities ()
hebrev ()	Converte texto hebraico para texto visual
hebrevc ()	Converte texto em hebraico para um texto visual e novas linhas (\ n) em
html_entity_decode ()	Entidades converte HTML para caracteres
htmlentities ()	Converte caracteres para entidades HTML
htmlspecialchars_decode ()	Converte algumas entidades HTML para caracteres predefinidos
htmlspecialchars ()	Converte alguns caracteres pré-definidos para entidades HTML
implode ()	Retorna uma string a partir dos elementos de um array

`join ()`	Sinônimo de implode ()
`levenshtein ()`	Retorna a distância entre duas seqüências Levenshtein
`localeconv ()`	Locale retorna numéricos e informações de formatação monetária
`ltrim ()`	Tiras em branco do lado esquerdo de uma string
`md5 ()`	Calcula o hash MD5 de uma string
`md5_file ()`	Calcula o hash MD5 de um arquivo
`metaphone ()`	Calcula a chave metaphone de uma string
`money_format ()`	Retorna uma string formatada como uma string de moeda
`nl_langinfo ()`	Retorna informações locais específicos
`nl2br ()`	Insere uma linha HTML quebra na frente de cada nova linha em uma string
`number_format ()`	Formata um número com os milhares agrupados
`ord ()`	Retorna o valor ASCII do primeiro caractere de uma string
`parse_str ()`	Analisa uma seqüência de consulta em variáveis
`print ()`	Saídas de uma string
`printf ()`	Saídas de uma string formatada
`quoted_printable_decode ()`	Decodifica uma string quoted-printable
`quotemeta ()`	Cita personagens meta
`rtrim ()`	Tiras em branco do lado direito de uma string
`setlocale ()`	Conjuntos de informações de localidade
`sha1 ()`	Calcula o hash SHA-1 de uma string
`sha1_file ()`	Calcula o hash SHA-1 de um arquivo
`similar_text ()`	Calcula a similaridade entre duas strings
`soundex ()`	Calcula a chave soundex de uma string
`sprintf ()`	Escreve uma string formatada para uma variável
`sscanf ()`	Interpreta a entrada de uma string de acordo com um formato
`str_ireplace ()`	Substitui alguns caracteres em uma string (case-insensitive)
`str_pad ()`	Almofadas de uma string para um novo comprimento

`str_repeat ()`	Repete uma seqüência de um determinado número de vezes
`str_replace ()`	Substitui alguns caracteres em uma string (case-sensitive)
`str_rot13 ()`	Executa a codificação ROT13 em uma corda
`str_shuffle ()`	Aleatoriamente embaralha todos os caracteres em uma string
`str_split ()`	Separa uma string em um array
`str_word_count ()`	Contar o número de palavras em uma string
`strcasecmp ()`	Compara dois strings (case-insensitive)
`strchr ()`	Encontra a primeira ocorrência de uma string dentro de outra string (alias strstr ())
`strcmp ()`	Compara dois strings (case-sensitive)
`strcoll ()`	Locale comparação de seqüência com base
`strcspn ()`	Retorna o número de caracteres encontrados em uma string antes de qualquer parte de alguns caracteres especificados são encontrados
`strip_tags ()`	Tiras de tags HTML e PHP de uma string
`stripcslashes ()`	Unquotes de uma string com addcslashes ()
`stripslashes ()`	Unquotes de uma string com addslashes ()
`stripos ()`	Retorna a posição da primeira ocorrência de uma string dentro de outra string (case-insensitive)
`stristr ()`	Encontra a primeira ocorrência de uma string dentro de outra string (case-insensitive)
`strlen ()`	Retorna o comprimento de uma string
`strnatcasecmp ()`	Compara dois strings usando uma "ordem natural" algoritmo (maiúsculas e minúsculas)
`strnatcmp ()`	Compara dois strings usando uma "ordem natural" algoritmo (case-sensitive)
`strncasecmp ()`	Comparação de seqüência dos primeiros n caracteres (case-insensitive)
`strncmp ()`	Comparação de seqüência dos primeiros n caracteres (case-sensitive)
`strpbrk ()`	Busca em uma palavra para qualquer de um conjunto de caracteres
`strpos ()`	Retorna a posição da primeira ocorrência de uma string dentro de outra string (case-sensitive)

strrchr ()	Localiza a última ocorrência de uma string dentro de outra string
strrev ()	Inverte uma string
strripos ()	Encontra a posição da última ocorrência de uma string dentro de outra string (case-insensitive)
strrpos ()	Encontra a posição da última ocorrência de uma string dentro de outra string (case-sensitive)
strspn ()	Retorna o número de caracteres encontrados em uma seqüência que contém apenas caracteres de um especificado charlist
strstr ()	Encontra a primeira ocorrência de uma string dentro de outra string (case-sensitive)
strtok ()	Separa uma string em strings menores
strtolower ()	Converte uma string em letras minúsculas
strtoupper ()	Converte uma string em letras maiúsculas
strtr ()	Traduz certos caracteres em uma string
substr ()	Retorna uma parte de uma string
substr_compare ()	Compara dois strings a partir de uma posição inicial especificado (binary safe e, opcionalmente, case-sensitive)
substr_count ()	Conta o número de vezes que uma substring ocorre em uma string
substr_replace ()	Substitui uma parte de uma string com outra string
trim ()	Tiras em branco de ambos os lados de uma string
ucfirst ()	Converte o primeiro caractere de uma string para maiúsculas
ucwords ()	Converte o primeiro caractere de cada palavra em uma string para maiúsculas
vfprintf ()	Escreve uma string formatada para um fluxo de saída especificado
vprintf ()	Saídas de uma string formatada
vsprintf ()	Escreve uma string formatada para uma variável
wordwrap ()	Quebra uma string em um determinado número de caracteres

Fonte: http://www.w3schools.com/php/php_ref_string.asp

Alguns exemplos:

```
<?php
$string = "Bom dia!";
```

```
//Retorna a quantidade de caracteres da variável $string
print ("Esta string contém " . strlen($string) . " caracteres. <BR>");

//Retorna a posição do character "m", lembrando que as posições começam em
zero. Caso nãp encontre o caracter retorna FALSO e se for impresso, aparecerá
um caracter em branco
print ("A posição do character 'm' é " . strpos($string, 'm') . "<BR>");

//Retorna a substring "dia", lembrando que as posições começam em zero
print ("A função retorna: " . strstr($string, 'dia') . "<BR>");

?>
```

Exemplo 7.9

Exercícios

1. Crie uma página para solicitar o nome completo e exibir numa nova página uma saudação como abaixo:

   Olá <primeiro nome>,

   Ficamos honrados por sua visita em nosso site!

2. Alterar o exercício acima para exibir o nome em maiúsculo.

3. Alterar o exercício acima para exibir o nome invertido (primeiro sobrenome, depois o primeiro nome).

Capítulo 8 – Arrays

Neste capítulo estudaremos aspectos do funcionamento dos arrays e as funções predefinidas do PHP para manipulação de arrays.

No PHP, listamos abaixo as formas mais comuns de como os arrays são utilizados:

- As variáveis de ambiente do PHP estão na forma de arrays ($_POST, $_GET, etc.).
- A maioria das funções de banco de dados recupera as informações via arrays.
- É simples transportar conjuntos de argumentos de um formulário HTML de uma página para outra através de arrays.
- Trabalhar com arrays agiliza algumas operações (como classificar, ordenar, contar, etc.) quaisquer dados que sejam gerados pelo script.

Definição de Arrays no PHP

No PHP os arrays são **associativos**, isto é, os arrays armazenam valores de elemento em associação com valores chave, em vez de uma ordem linear de índice. Em outras linguagens de programação, é comum encontrarmos arrays vetoriais. Num array vetorial, todos os elementos contidos precisam ser do mesmo tipo e os índices valores inteiros seqüenciais.

No PHP, é possível atribuir valores arbitrários como elementos do array. Assim como as chaves que também podem assumir valores abitrários, incluindo chave na forma de string. Poderíamos ter atribuições sucessivas de array como estas:

```
$meu_array[1] = 1;
$meu_array ['orange'] = 2;
$meu_array [3] = 3;
```

Exemplo 8.1

No exemplo abaixo, quando armazenamos um elemento em um array, associado a uma chave, tudo o que precisamos recuperar mais tarde é o valor da chave.

```
$estado['São Paulo'] = 'Campinas';
```

Exemplo 8.2

Acima o elemento `'Campinas'` é armazenado na variável de array `$estado`, em associação com a chave de pesquisa `'São Paulo'`. Para recuperar o elemento armazenado na variável, é só utilizarmos a chave para a pesquisa:

```
cidade = $estado['São Paulo'];
```

Exemplo 8.3

Criando Arrays

Podemos criar arrays de três formas:

- Atribuindo um valor em um array
- Utilizando o comando (construtor) `array()`
- Chamando uma função que retorna um array como seu valor

Atribuição direta

A maneira mais simples de criar um array é através de uma atribuição, assim como fazemos nas variáveis:

```
$meu_array[1] = "Bom dia!";
```

Exemplo 8.4

Se `$meu_array` fosse uma variável comum (não array) antes da atribuição acima, depois de executar a atribuição a variável passaria a ser um array com um elemento. Se o array já existisse, seria criado um novo índice (inteiro 1) e um valor associado (a string "Bom dia"). Se o índice já existisse, o seu conteúdo seria sobrescrito.

Construção `array()`

Através do construtor `array()` podemos criar um novo array a partir da especificação de seus elementos e chaves (índices) associadas. Quando chamamos `array()` sem argumentos, é criado um novo array vazio. Também podemos usar `array()` passando uma lista de elementos separados por vírgulas, a qual será armazenada no array, na ordem especificada e serão atribuídas chaves de inteiro iniciando com zero. Exemplo:

```
$frutas = array('maçã','laranja´,´banana´,´pêra´);
```

Exemplo 8.5

A instrução acima faz com que o array `$frutas` receba quatro elementos de string `('maçã','laranja´,´banana´,´pera´)`, com os índices 0, 1, 2 e 3, respectivamente.

O comando acima produzirá o mesmo resultado das instruções abaixo:

```
$frutas[0] = 'maçã';
$frutas[1] = 'laranja';
$frutas[2] = 'banana';
$frutas[3] = 'pêra';
```

Exemplo 8.6

Se omitirmos os índices ainda teremos o mesmo resultado:

```
$frutas[] = 'maçã';
$frutas[] = 'laranja';
$frutas[] = 'banana';
$frutas[] = 'pêra';
```

Exemplo 8.6

Nos exemplos acima, os índices não são manipulados, portanto recebem o valor padrão que são inteiros iniciando em 0 (zero). Mas o construtor `array()` também nos permite especificar de que maneira

os índices devem ser. Para isso forneceremos pares de chaves e valores separados por vírgulas, onde a chave e o valor são separados pelo símbolo =>. Exemplo:

```
$frutas = array(0 => 'maçã', 1 => 'laranja',
                2 => 'banana', 3 => 'pêra');
```

Exemplo 8.7

O resultado acima será idêntico aos exemplos vistos anteriormente. Mas com essa sintaxe, podemos mudar o tipo dos índices. Exemplo:

```
$frutas = array('vermelho' => 'maçã','laranja' => 'laranja' ,
                'amarelo' => 'banana', 'verde' => 'pêra');
```

Exemplo 8.8

Funções que retornam arrays

A última maneira de criarmos um array é chamando uma função que retorna um array. Isso pode acontecer através de uma função do próprio PHP ou uma função criada pelo usuário.

Muitas funções que manipulam banco de dados retornam resultados em arrays. Existem também funções para manipulação de arrays onde um array será criado mediante uma operação de uma função. Por exemplo:

```
$meu_array = range(1,5);         // Cria array com elementos de 1 a 5
$meu_array = array(1,2,3,4,5);   // Idem ao comando acima
```

Exemplo 8.9

Recuperando Valores

Para recuperar valores dos arrays, podemos usar a maneira mais direta, que é através dos índices. Exemplo:

```
<?php

$frutas = array('maçã','laranja´,´banana´,´pêra´);

echo $frutas[2];       // Exibirá banana

$frutas = array('vermelho' => 'maçã',
                'laranja' => 'laranja' ,
                 'amarelo' => 'banana',
                 'verde' => 'pêra');

echo  $frutas['vermelho']; // Exibirá maçã

?>
```

Exemplo 8.10

No decorrer do capítulo veremos várias formas de recuperarmos os valores dos arrays sem utilizar as chaves. Como exemplo, o `list()` é utilizado para atribuir vários elementos do array a variáveis em sucessão. Exemplo:

```
<?php

$frutas = array('maçã', 'banana', 'pêra');
list($frutas_vermelhas,$frutas_amarelas) = $frutas;

echo $frutas_vermelhas;  // vai exibir maçã
echo $frutas_amarelas;   // vai exibir banana

?>
```

Exemplo 8.11

Será atribuído à variável `$frutas_vermelhas` a string `'maçã'` e à variável `$frutas_amarelas` a string `'banana'`. A string `'pêra'` não será atribuída, porque não fornecemos variáveis suficientes.

Ao compararmos `list()` e `array()` percebemos que eles se opõem pois `array()` agrupa elementos numa só estrutura e `list()` separa os elementos em variáveis independentes.

Arrays Multidimensionais

Vimos até agora arrays com apenas um nível de chaves. Na tabela abaixo, temos na primeira coluna os índices e na segunda os valores:

vermelho	maçã
laranja	laranja
amarelo	banana
verde	pêra

Porém o PHP pode suportar múltiplos arrays dimensionais, com número arbitrário de chaves. Assim como em arrays com uma dimensão, arrays multidimensionais podem ser criados a partir de uma atribuição como:

```
$multi_array[1][2][3][4][5] = "Bom dia!";
```

Exemplo 8.12

Esse é um array de cinco dimensões com chaves sucessivas que acontece, nesse caso, de ser cinco inteiros sucessivos. O conceito de arrays multidimensionais pode parecer um pouco confuso de se entender, mas se imaginarmos que um array multidimensional não passa de um array que armazena outros arrays fica mais fácil entendermos o que realmente acontece com os arrays multidimensionais. Exemplo:

```
$tipos = array('frut' =>
                 array ('vermelha' => 'maçã',
                        'laranja' => 'laranja',
                        'amarelo' => 'banana',
                        'verde' => 'pêra'),
                'flor' =>
                 array ('vermelha => 'rosa',
                        'amarelo' = > 'girassol',
                        'branco' => 'lírio'));
```

Exemplo 8.13

No exemplo acima, temos um array com dois elementos armazenados na associação com chaves. Cada um desses valores é um array próprio. Podemos referenciá-los assim:

```
$tipo_desejado = 'flor';
$cor_desejada = 'branco';

print ("A $tipo_desejado $cor_desejada é " . $tipos[$tipo_desejado][
$cor_desejada]);
```

Exemplo 8.14

Inspecionando arrays

Listamos algumas funções para descobrirmos informações sobre `arrays()`:

Função	Comportamento
`is_array()`	Aceita um único argumento de qualquer tipo e retorna um valor verdadeiro se o argumento for um array, e falso caso contrário.
`count()`	Aceita um array como um argumento e retorna o número de elementos não vazios no array. (Esse número será 1 para strings e números).
`sizeof()`	Idêntico ao `count()`.
`in_array()`	Aceita dois argumentos: um elemento (que poderia ser em um array) e um array (que poderia conter o elemento). Retorna true se o elemento estiver contido como um valor no array, false, caso contrário. (Não testa presença de chaves no array).
`isSet ($array[$key])`	Aceita uma forma array[chave] e retorna true se a aprtede chave for uma chave válida para o array.

Excluindo Elementos de Array

Para excluir elementos de um array seguimos o mesmo processo para excluir uma variável. Usamos a função `unset()`, como mostra o exemplo abaixo:

```
$meu_array[0] = "Bom dia";
$meu_array[1] = "Boa tarde";
$meu_array[2] = "Boa noite";

unset ($meu_array[1]);
```

Exemplo 8.15

No exemplo acima, o array `$meu_array` foi iniciado através de três atribuições. Foram criados três índices (0, 1 e 2) com seus respectivos valores (`"Bom dia", "Boa tarde" e "Boa noite"`). Ao executar a função unset(), o segundo índice deixa de existir. O array agora passa a ter apenas dois índices (0 e 2) e seus valores originais (`"Bom dia" e "Boa noite"`).

Devemos lembrar que usar a função unset() não é a mesma coisa de atribuir valor vazio. Nesse caso, o array continuaria com os três índices.

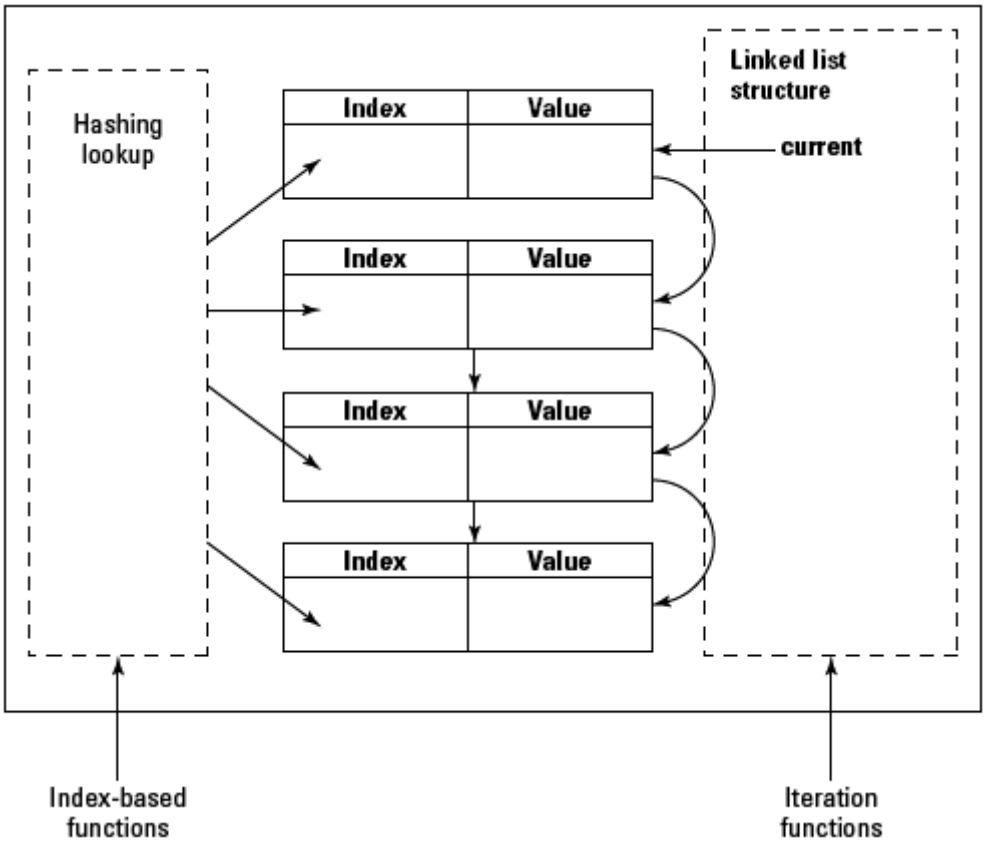
Iteração

Até agora vimos como criar arrays, como localizar conteúdos e como excluí-las. Veremos nessa seção como recuperar elementos do array em massa. Isso é possível percorrendo ou realizando loops por arrays, elemento por elemento ou chave por chave.

Em algumas situações, podemos nos deparar numa situação mais complicada quando um array pode apresentar índices de tipos variados. E um loop construído para percorrer um array com índices numéricos não funcionará se existir índices do tipo string.

No PHP, há um tipo de sistema de ponteiro oculto construído nos arrays. Cada par chave/valor armazenado aponta para o próximo par e um efeito colateral de adicionar o primeiro elemento a um array é que o ponteiro atual aponta para o primeiro elemento, onde permanecerá a menos que alterados por uma das funções de iteração.

O sistema de ponteiro de lista vinculada é uma maneira alternativa de inspecionar e manipular arrays, que existe lado a lado com o sistema que permite armazenamento e pesquisa baseada em chave. Abaixo temos uma figura que ilustra a estrutura interna de um array:



Funções de Iteração

Vamos construir um array de exemplo com a seguinte estrutura:

0	Caracas
Caracas	Venezuela
1	Paris
Paris	França
2	Tóquio
Tóquio	Japão

O código ficaria assim:

```
$cidade = array();
$cidade[0] = 'Caracas';
$cidade['Caracas'] = 'Venezuela';
$cidade[1] = 'Paris';
$cidade['Paris'] = 'França';
$cidade[2] = 'Tóquio';
$cidade['Tóquio'] = 'Japão';
```

Exemplo 8.16

No exemplo acima, criamos alguns índices numéricos e associamos nomes de cidades a eles. Também armazenamos nomes de países, indexados por nomes de cidades.

Se quisermos extrair os dados armazenados nesse array, podemos usar o seguinte exemplo:

```
function cidade_por_num($num, $array)
{
   if (isSet($array[$num]))
   {
      $nome_cidade = $array[$num];
      $nome_pais = $array[$nome_cidade];

      Print ("$nome_cidade está em $nome_pais <br>");
    }
}

cidade_por_num(0,$cidade);
cidade_por_num(1,$cidade);
cidade_por_num(2,$cidade);
```

Exemplo 8.17

Usando a criação do array acima, a saída esperada será:

```
Caracas está em Venezuela
Paris está em França
Tóquio está em Japão
```

Exemplo 8.18

Foreach

Quando conhecemos a estrutura do array fica fácil construir uma função para ler seus valores. Mas caso quiséssemos exibir todo o conteúdo de um array, poderíamos optar por outros recursos como é o caso do **`foreach`**. O `foreach` é uma herança da linguagem Perl e funciona apenas com arrays. Ele possui duas sintaxes:

```
foreach ($expressao_array as $valor) instrucoes
foreach ($expressao_array as $chave => $valor) instruções
```

Exemplo 8.19

No primeiro caso, é passado um array através de `$expressao_array`. Em cada 'loop', o valor do elemento corrente é atribuído a `$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 atribuído à variável $chave em cada iteração.

```
<?php

$cidade = array();
$cidade[0] = 'Caracas';
$cidade['Caracas'] = 'Venezuela';
$cidade[1] = 'Paris';
$cidade['Paris'] = 'França';
$cidade[2] = 'Tóquio';
$cidade['Tóquio'] = 'Japão';

function exibe_tudo($array_cidade)
{
   foreach ($array_cidade as $nome_valor) {
      print("$nome_valor <br>");
   }
}
exibe_tudo($cidade);

?>
```

Exemplo 8.19

Current() e Next()

A função current() retorna o elemento corrente em um array. Quando um array é criado, o valor apontado é sempre o primeiro. A função next() avança o ponteiro para o próximo elemento e retorna seu valor. Será retornado um valor falso caso o ponteiro já esteja no último elemento e é dado novamente o comando next() para avançar mais uma posição.

```
function exibe_array($array_cidade)
{
   print (current($array_cidade)."<br>");
   next($array_cidade);
   print (current($array_cidade)."<br>");
}
exibe_array($cidade);
```

Exemplo 8.20

Obs.: Se o array contêm elementos vazios (0 ou "", uma string vazia) então esta função retorna FALSE para esses elementos. Isso faz com que seja impossível determinar se você está realmente no final da lista de elementos de um array usando current(). Para percorrer devidamente um array que pode conter elementos vazios, use a função each().

Reset()

A função reset() faz o ponteiro interno de um array apontar para o seu primeiro elemento, independente da posição atual do ponteiro.

```
function exibe_array($array_cidade)
{
   print (current($array_cidade)."<br>"); // exibe Caracas (1o. elemento)
   next($array_cidade);
   print (current($array_cidade)."<br>"); // exibe Venezuela (2o. elemento)
   reset($array_cidade);                   // vai para 1a. posição no array
   print (current($array_cidade)."<br>"); // exibe Caracas (1o. elemento)
```

```
}
```

Exemplo 8.21

End() e Prev()

A função `prev()` retrocede o ponteiro interno de um array. A função `end()` faz o ponteiro interno de um array apontar para o seu último elemento.

```
function exibe_array($array_cidade)
{
   print (current($array_cidade)."<br>"); // exibe Caracas (1o. elemento)
   next($array_cidade);
   print (current($array_cidade)."<br>"); // exibe Venezuela (2o. elemento)
   reset($array_cidade);                       // vai para 1a. posição no array
   print (current($array_cidade)."<br>"); // exibe Caracas (1o. elemento)
   end($array_cidade);                    // vai para última posição
   print (current($array_cidade)."<br>"); // exibe Japão
   prev($array_cidade);               // antecede uma posição no array
   print (current($array_cidade)."<br>"); // exibe Tóquio
}
```

Exemplo 8.22

Key()

A função `key()` retorna a chave da posição corrente de um array.

```
function exibe_array($array_cidade)
{
print (key($array_cidade)."<br>"); // exibe 0 (chave do 1o. elemento)
next($array_cidade);
print (key($array_cidade)."<br>"); //exibe Caracas (chave do 2o. elemento)
reset($array_cidade);                 // vai para 1a. posição no array
print (key($array_cidade)."<br>"); // exibe 0 (chave 1o. elemento)
end($array_cidade);                    // vai para última posição
print (key($array_cidade)."<br>"); // exibe Tóquio
prev($array_cidade);                   // antecede uma posição no array
print (key($array_cidade)."<br>"); // exibe 2
}
```

Exemplo 8.23

Each()

A função `each()` retorna o par chave/valor corrente de array e avança o seu cursor. Esse par é retornado num array de quatro elementos, com as chaves 0, 1, key, e value. Os elementos 0 e key contêm o nome da chave do elemento do array, e 1 e value contêm o valor.

Se o cursor interno do array estiver apontando para além do final do array, `each()` retorna FALSE.

```
function exibe_array($array_cidade)
{
   $elemt = each($array_cidade);
   print_r ($elemt);
}
```

Exemplo 8.24

Funções de Transformação de Array

Veremos a seguir uma lista de funções de array destinadas a transformar arrays:

Função	Comportamento
array_keys()	Aceita um único argumento de array e retorna um novo array em que os novos valores são as chaves do array de entrada e as novas chaves são os inteiros incrementados a partir de zero.
array_values()	Aceita um único argumento de array e retorna um novo array em que os novos valores são os valores originais do array de entrada e as novas chaves são os inteiros incrementados a partir de zero.
array_count_values()	Aceita um único argumento de array e retorna um novo array em que as novas chaves são os valores do array antigo e os novos valores são uma contagem de quantas vezes esse valor original ocorreu no array de entrada.
array_flip()	Aceita um único argumento de array e inverte as relações entre chaves e valores.
array_reverse()	Aceita um único argumento de array e retorna um array com os elementos na ordem inversa.
shuffle()	Aceita um único argumento de array e mistura de forma aleatória os elementos de um array. Você deve usar essa função em conjunto com `srand()`.
array_merge()	Aceita dois argumentos de array, mescla-os e retorna um novo array contendo os elementos do primeiro array e sem seguida os elementos do segundo array.

Funções de Classificação de Array

Função	Comportamento
asort()	Aceita um único argumento de array. Classifica os pares chaves/valor por valor, mas mantém o mesmo mapeamento de chave/valor.
arsort()	Mesmo que `asort()`, mas classifica em ordem decrescente.
ksort()	Aceita um único argumento de array. Classifica os pares chaves/valor por chave, mas mantém o mesmo mapeamento de chave/valor.
krsort()	Mesmo que `ksort()`, mas classifica em ordem decrescente.
sort()	As chaves podem ser renumeradas para refletir o novo ordenamento dos valores.
rsort()	Mesmo que `sort()`, mas classifica em ordem decrescente
uasort()	Ordena um array utilizando uma função de comparação definida pelo usuário e mantendo as associações entre chaves e valores. As associações chave/valor não são mantidas.
uksort()	Ordena um array pelas chaves utilizando uma função de comparação definida pelo usuário.
usort()	Ordena um array pelos valores utilizando uma função de comparação definida pelo usuário. As associações chave/valor não são mantidas.

Exercícios

1. Execute o exemplo abaixo:

```
<?php
$idades['Pedro'] = "32";
```

```
$idades['Maria'] = "30";
$idades['João'] = "34";

echo "Pedro possui " . $ idades['Pedro'] . " anos de vida.";
?>
```

2. Crie um array com os estados brasileiros e os respectivos nomes dos estados. Exibir todo o array no navegador.

Capítulo 9 – Sessões

Nesse capítulo estudaremos a criação e manipulação de sessões para monitorar a navegação dos usuários pelo.

O que é uma sessão?

Sessão de navegação Web é um período de tempo durante o qual um usuário acessa páginas Web de um site. Para a sessão, é contado a desde o primeiro download da primeira até a última página que o usuário acessou.

Por que monitorar acessos ao meu site?

O protocolo ***HTTP***, por meio do qual os navegadores se comunicam com os servidores Web, não registra quaisquer informações que possam identificar os acessos às páginas do site, freqüência, etc. Ou seja, o servidor Web reage independentemente de cada solicitação individual que ele recebe e não tem nenhuma maneira de vincular solicitações entre si apesar de estar registrando em log tais solicitações. Se dois usuários distintos acessam um determinado site e os dois usuários pedem a página 1 e depois a página 2 do site, o protocolo HTTP não oferece nenhuma ajuda para descobrir que duas pessoas visualizaram dias páginas cada uma. O que se vê no log são quatro solicitações individuais para páginas, com várias informações anexadas a cada solicitação.

Além dessas informações não realizarem a identificação pessoal (com informações sobre o nome, endereço, correio eletrônico, etc.), elas não oferecem nenhuma confiabilidade para garantir se duas páginas foram acessadas pelo mesmo usuário.

Abaixo listamos algumas razões pelas quais devamos nos preocupar onde as sessões iniciam e onde terminam:

- Necessidade de personalizar as experiências de nossos usuários à medida que eles se movem pelo site, de uma maneira que dependa de quais ou quantas páginas eles já viram.
- Necessidade de exibir anúncios para o usuário, mas não exibir um anúncio dados mais de uma vez por sessão.
- Necessidade que a sessão acumule informações sobre as ações dos usuários enquanto eles navegam – como em um monitoramento de pontos acumulados em um jogo de aventura ou de guerra, ou em um shopping de um site de comércio eletrônico.
- Necessidade de monitorar como as pessoas navegam por nosso site em geral: quando elas visitam essa página interna, seria porque a marcaram ou chegaram a partir da página anterior?

Para todos esses propósitos, precisamos ser capazes de corresponder às solicitações de páginas com as sessões das quais elas fazem parte e, para alguns propósitos, seria ótimo armazenar algumas informações associadas à sessão que tem prosseguimento. As sessões do PHP resolvem estes dois problemas para nós.

Alternativas "domésticas"

Antes de examinarmos o tratamento de sessões do PHP, examinaremos algumas técnicas alternativas de como o problema pode ser tratado.

Endereço IP

Os servidores Web normalmente conhecem o nome de host (nome da máquina) ou o endereço IP do usuário que está solicitando uma página. Na maioria das configurações do PHP, essas informações aparecem como variáveis **$REMOTE_HOST** e **$REMOTE_ADDR**, respectivamente.

Se verificarmos nos logs do servidor Web que duas solicitações em sucessão do mesmo endereço IP, seu código pode concluir seguramente que a mesma pessoa seguiu um link ou formulário de uma de suas páginas do site para outra.

A informação baseada nos endereços IPs não é segura, pois o endereço IP registrado no log pode não pertencer ao computador que o usuário está usando para acessar a Internet, mas pertencer a um servidor de **Proxy**, por exemplo. O navegador do usuário solicita a URL do servidor de proxy, que por sua vez solicita a página de seu servidor Web e então encaminha essa página para o usuário.

Variáveis Ocultas

Cada solicitação de http é tratada independentemente, mas toda vez que seu usuário se move de página a página dentro de seu site, isso normalmente ocorre via um link ou pelo envio de um formulário. Se a primeira página que um usuário visita pode de alguma maneira gerar um rótulo único para essa visita, então cada "passagem" subseqüente de uma página para outra pode passar esse identificador único junto.

Por exemplo, uma parte hipotética de código possa ser incluído na parte superior de cada página em site:

```
If (!IsSet($minha_s_id))
   $minha_s_id = gerar_session_id(); //função hipotética!
```

Exemplo 9.1

No código acima, é verificado se a variável `$minha_s_id` é vinculada. Se for, assumimos que ela foi passada e estamos no meio de uma sessão. Se não for, assumimos que somos a primeira página de uma nova sessão e chamamos a função hipotética `gerar_session_id()` para criar um identificador único.

Depois de incluirmos o código anterior, assumimos que temos um identificador único para a sessão e nossa única responsabilidade é passá-lo em qualquer página que vinculamos ou enviamos. Cada link de nossa página deve incluir `$minha_s_id` como um argumento GET, como em:

```
<A HREF="next.php?minha_s_id=<?php echo $minha_s_id; ?>"> Próximo </A>
```

Exemplo 9.2

E cada formulário de envio deve ter um argumento POST oculto embutido nele, como este:

```
<FORM ACTION=next.php METHOD=POST>
  Corpo do formulário...
  <INPUT TYPE=HIDDEN NAME=minha_s_id VALUE="<?php echo $minha_s_id;?>" >
</FORM>
```

Exemplo 9.3

A técnica acima oferece uma ótima maneira de manter sessões diferentes, desde que sejam gerados identificadores únicos.

Cookies

Outra maneira de monitorar sessão é utilizar um identificador único de sessão como na seção anterior, mas fazer a "passagem" configurando ou verificando um cookie.

Um cookie é um tipo especial de arquivo, localizado no sistema de arquivos do computador do usuário, que os servidores Web podem ler e gravar. Em vez de verificar uma variável GET/POST passada, e atribuir um novo identificador se nenhum for localizado, seu script verifica se existe um arquivo de cookie previamente gravado e armazena um novo identificador em um novo arquivo de cookie se nenhum for localizado. Esse método tem alguns benefícios sobre utilizar variáveis ocultas: o mecanismo trabalha nos bastidores (em geral não mostra nenhum traço de sua atividade na janela de navegador) e o código que verifica ou configura o cookie pode estar centralizado (em vez de afetar cada formulário e link).

A desvantagem é que navegadores antigos não suportam cookies e os mais recentes permitem aos usuários negar privilégios de configuração de cookie para servidores Web. Portanto, embora os cookies ofereçam uma solução relativamente fácil, não podemos assumir que eles sempre estejam disponíveis.

Estudaremos mais sobre os cookies nos capítulos seguintes.

Como as sessões trabalham no PHP

Bom suporte de sessão cuida de dois pontos:

- Monitoramento de sessão, isto é, detectar se duas invocações separadas de script são, de fato, parte da mesma sessão de usuário.
- Armazenamento de informações associadas a uma sessão.

O monitoramento de sessão do PHP trabalha com uma combinação do método de variáveis ocultas e como método de cookie descrito acima. Por causa das vantagens dos cookies, o PHP os utilizará quando o navegador do usuário suportá-los e, caso contrário, recorreremos ao ocultamento do ID de sessão em um argumento GET e POST. Felizmente, porém, as próprias funções de sessão operam em um nível mais abstrato e cuidam de verificar suporte de cookie por si próprios.

Obs.: Se quiser que o PHP trate transparentemente a passagem de variáveis de sessão quando os cookies não estiverem disponíveis, você precisa configurar o PHP com ambas as opções `--enable-trans-sid` e `--enable-track-vars`. Se o PHP não tratar disso, você deve organizar para passar um argumento GET ou POST, na forma de `nome_da_sessão=id_da_sessão`, para todos os seus links e formulários. Quando uma sessão está ativa, o PHP fornece uma constante especial, `SID`, que contém o par argumento/valor correto. A seguir um exemplo de inclusão dessa constante em um link:

```
<A HREF="proxima_pagina.php"?<?php echo (SID);?>"> Próxima página </A>
```

Exemplo 9.4

Identificando a Sessão

Para informar ao PHP que uma sessão está sendo criada, para que ele possa se conectar a ela e recuperar quaisquer informações associadas, usamos a função `session_start()`.

O efeito de `session_start()` depende do PHP poder ou não localizar um identificador de sessão prévia, como fornecido tanto pelos argumentos de HTTP como por um cookie. Se um identificador for localizado, os valores de quaisquer variáveis de sessão previamente registradas serão recuperados. Se um

identificador não for localizado, o PHP assume que estamos na primeira página de uma nova sessão e gera um novo ID de sessão.

Propagando Variáveis de Sessão

Depois de chamar a função `session_start()`, de preferência faça isso no início do script, utilize o array superglobal `$_SESSION` como variável para armazenar qualquer coisa que você queira recuperar novamente de uma página anterior na mesma sessão. Assuma que quaisquer outras variáveis serão abandonadas quando você sair dessa página e que tudo nessa "valise" estará aí quando você chegar à próxima página.

Sendo assim, o código de sessão para propagar uma única variável numérica é tão simples quanto isso:

```
<?php

session_start();

$tmp_num = 45;
$num1 = 19;
$num2 = 33;

$_SESSION['num'] = $num1;

?>
```

Exemplo 9.5

O código receptor será como segue o exemplo:

```
<?php

session_start();

$var_pagina_anterior = $_SESSION['num'];

[..]

?>
```

Exemplo 9.6

Dessa forma fazemos a passagem de informações entre páginas através de sessões. A atribuição no array superglobal `$_SESSION` cria implicitamente qualquer registro necessário para que o novo valor seja transportado adiante para a próxima página.

Onde os dados são realmente armazenados?

Há duas coisas sobre as quais o mecanismo de sessão deve apoiar-se: o próprio ID de sessão e quaisquer vinculações de variáveis vinculadas.

Como vimos, o ID de sessão é tanto armazenado como um cookie na máquina do usuário, como incorporado nos argumentos GET/POST enviados com solicitações de página. No último caso, não há realmente armazenamento acontecendo. O ID é enviado como parte de uma solicitação e é retornado dentro do código de HTML para links e formulários, que podem gerar a próxima solicitação. O navegador e o servidor passam essas informações vitais de um lado para outro, e a sessão é efetivamente encerrada se qualquer lado derrubá-la.

Por padrão, os conteúdos de variáveis de sessão são armazenados em arquivos especiais no servidor, um arquivo por ID de sessão.

É possível configurar o PHP para armazenar o conteúdo de variáveis de sessão em um banco de dados do lado servidor, em vez de em arquivos. Veremos isso mais adiante.

Exemplo

A seguir mostraremos um script curto (tirado do livro PHP, a Bíblia) que tem um propósito duplo. O primeiro propósito é fornecer um exemplo de um script completo que utiliza com sucesso as funções de sessão; o segundo é fornecer um script de teste que podemos utilizar para certificar-nos de que temos suporte a sessão e que esse suporte está sendo feito o que esperamos.

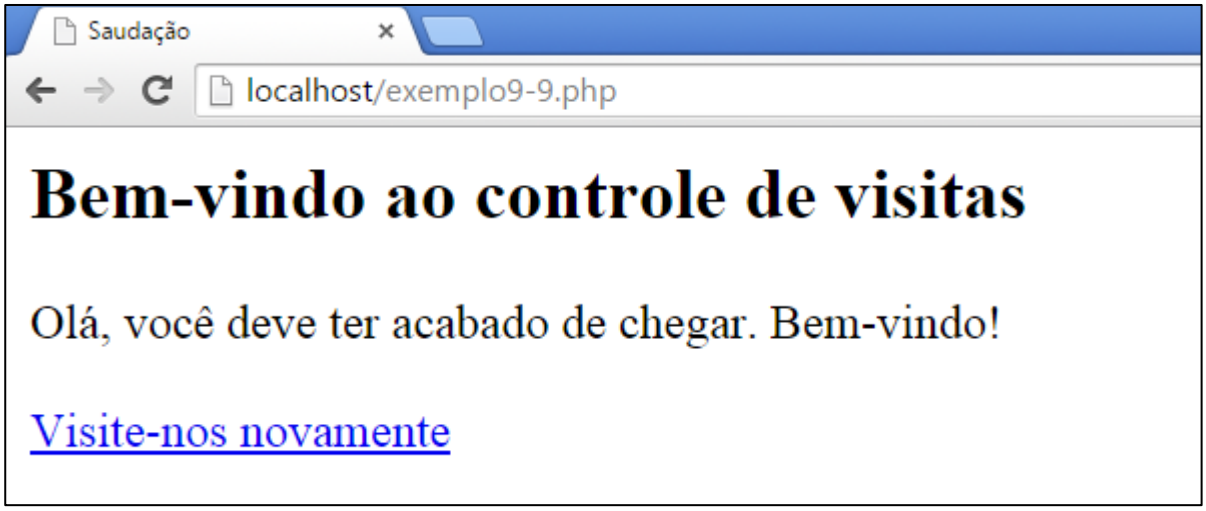
São realizadas as seguintes tarefas:

- Iniciar uma sessão (ou selecionar uma existente) utilizando `session_start()`.
- Verificar a existência de uma entrada pré-existente em `$_SESSION`. Se não estiver presente, assumimos que a sessão é nova.
- Incrementar um contador que monitora o número de vezes que o usuário visitou essa página.
- Armazenar o contador que monitora o número de vezes que o usuário visitou a página.
- Fornecer um link de volta à própria página, incorporando o ID de sessão como um argumento se ele for localizado.

```
<?php
session_start();
?>
<HTML>
    <HEAD>
        <TITLE>Saudação</TITLE>
        <META charset="utf-8">
    </HEAD>
    <BODY>
        <H2>Bem-vindo ao controle de visitas </H2>
        <?php

        if (!IsSet($_SESSION['visit_count'])) {
            print("Olá, você deve ter acabado de chegar. Bem-vindo!<BR>");
            $_SESSION['visit_count'] = 1;
        } else {
            $_SESSION['visit_count'] += 1;
            print("De volta à página? Isto acontece " . $_SESSION['visit_count'] .
                    " vez(es) agora (sem contar outro visitante).<BR>");

        }
        $self_url = $_SERVER['PHP_SELF'];
        $session_id = SID;
        if (IsSet($session_id) &&
                $session_id) {
            $href = "$self_url?$session_id";
        } else {
            $href = $self_url;
        }
        print("<BR><A HREF=\"$href\">Visite-nos novamente</A> ");
        ?>


    </BODY>
</HTML>
```

Exemplo 9.9

Ao rodar o script, devemos ver algo como na tela abaixo:



Ao clicar no link, veremos a tela abaixo. Repare que há um contador e que a cada carregamento da página, o contador é incrementado:

A segunda parte da listagem, é sobre como construir um autolink que propagará as informações de sessão mesmo sem suporte de cookie. Podemos testá-lo desativando os cookies no navegador. Normalmente, isto é uma opção avançada ou de segurança nas preferências ou nas opções do navegador.

A incorporação do ID de sessão no URL é exatamente o que deve ser desnecessário se o PHP foi compilado com `--enable-trans-sid`. Neste caso, podemos adicionar outro autolink para essa página, sem incorporar qualquer coisa extra no URL. O PHP deve cuidar de fazê-lo.

Funções de sessão

Abaixo temos uma lista das principais funções relacionadas às sessões.

Função	Descrição
`session_start()`	Não aceita argumento e faz com que o PHP note um ID de sessão que foi passado para ele (via cookie ou GET/POST) ou crie um novo ID de sessão se nenhum for encontrado. Se um ID de sessão antigo é localizado, o PHP recupera as atribuições de todas as variáveis que foram registradas e disponibiliza essas variáveis atribuídas como variáveis globais regulares.

`session_register()`	Recebe uma string como argumento e registra a variável identificada pela string (a variável como parâmetro não deve conter o $). O efeito de registrar uma variável é que atribuições subseqüentes para essa vairável serão preservados para sessões futuras. (Depois que um script termina, as variáveis registradas e seus valores são serializados e propagados de tal maneira que chamadas posteriores a `session_start()` possam recriar as vinculações). Se `session_start()` ainda não tiver sido chamada, `session_register` a fará implicitamente.
`session_unregister()`	Recebe um nome de variável de string como argumento e "desregistra" a variável correspondente da sessão.
`session_id_registered()`	Recebe uma string de nome de variável e testa se uma variável com um nome de variável dado está registrada na sessão atual, retornando TRUE se estiver e FALSE caso contrário.
`session_destroy()`	Essa função livra todas as informações de variáveis de sessão que foram armazenadas. Não zera nenhuma variável no script atual.
`session_name()`	Quando chamada sem argumentos, retorna a string do nome da sessão atual. Isso é normalmente 'PHPSESSID' por padrão. Quando chamada com um argumento de string, `session_name()` configura o nome de sessão atual para essa string. Esse nome é utilizado como uma chave para localizar o ID de sessão em cookies e argumentos GET/POST – para quando os valores foram serializados e armazenados.
`session_module_name()`	Quando chamada sem argumentos, retorna a string do nome do módulo que é responsável pelo tratamento de dados de sessão. Esse nome atualmente assume o padrão 'files', o que significa que vinvulações de sessão são serializadas e depois gravadas em arquivos no diretório identificado pela função `session_save_path()`. Quando chamada com um argumento de string, altera o nome de módulo.
`session_save_path()`	Retorna (ou configura, se receber um argumento) o nome de caminho do diretório em que os arquivos de vinculação de variável de sessão serão gravados (que em geral assume o padrão /tmp em sistemas Unix).
`session_id()`	Não recebe argumentos e retorna uma string que é a única chave correspondente a uma sessão particular. Se dado um argumento de string, será configurado o ID de sessão para essa string.
`session_encode()`	Retorna uma codificação de string do estado da sessão atual, adequado para utilização por `string_decode`(). Isso pode ser utilizado para salvar uma sessão para renascimento em algum momento mais tarde, como gravar a string codificada em um arquivo ou banco de dados.
`session_decode()`	Recebe uma codificação de string como produzida por `session_encode()` e restabelece o estado de sessão, transformando vinculações de página como `session_start()` faz.

Configurações

As variáveis abaixo são configuráveis no arquivo **php.ini** e visualizáveis chamando `phpinfo()`.

Variável php.ini	Valor típico	Descrição
`session.save_path`	/tmp (no Unix)	Nome do caminho para a pasta do lado do servidor onde os arquivos de vinculação serão gravados.
`session.auto_start`	0	Quando 1, as sessões inicializarão automaticamente toda vez que um script carregar. Quando 0, nenhum código de sessão será código

		a menos que haja uma chamada explícita para `session_start()` ou `session_register()`.
`session.save_handler`	'files'	A string que determina o método subjacente para salvar informações de variáveis de sessão.
`session.cookie_lifetime`	0	Especifica quanto tempo os cookies de sessão levam para expirar e consequentemente o tempo de vida de uma sessão. O padrão 0 significa que as sessões duram até o navegador ser fechado.
`session.use_cookies`	1	Se 1, o mecanismo de sessão tentará propagar o ID de sessão configurando/verificando um cookie. Se o valor for 0 não ocorrerá nenhuma tentativa para utilizar cookies.

Exercícios

1. Execute o exemplo 9.9.

Capítulo 10 – Manipulação de Arquivos

O PHP possui suporte a manipulação de arquivos. Veremos adiante algumas funções para manipulação de arquivos.

Abrindo Arquivos

Usamos a função `fopen()` para abrir arquivos. Essa função aceita dois argumentos: nome do arquivo e o modo que será aberto. No exemplo abaixo estamos abrindo o arquivo `welcome.txt` no modo apenas de leitura dos dados do arquivo.

```
<?php

$file=fopen("welcome.txt","r");

?>
```

Exemplo 10.1

Na tabela abaixo listamos as demais opções de abertura de arquivo:

Modos	Descrição
r	Somente para leitura. Começa no início do arquivo
r+	Read / Write. Começa no início do arquivo
w	Escrever apenas. Abre e limpa o conteúdo do arquivo, ou cria um novo arquivo se ele não existe
w+	Read / Write. Abre e limpa o conteúdo do arquivo, ou cria um novo arquivo se ele não existe
a	Acréscimo. Abre e escreve para o fim do arquivo ou cria um novo arquivo se ele não existe
a+	Leitura / Append. Preserva o conteúdo do arquivo por escrito para o final do arquivo
x	Escrever apenas. Cria um novo arquivo. Retorna FALSE e um erro se o arquivo já existe
x+	Read / Write. Cria um novo arquivo. Retorna FALSE e um erro se o arquivo já existe

ⓘ Se a função fopen () é incapaz de abrir o arquivo especificado, ele retorna 0 (false).

```
<?php

$file=fopen("welcome.txt","r") or exit("Não é possível abrir o arquivo!");

?>
```

Exemplo 10.2

Fechando Arquivos

Usamos a função `fclose()` para fechar arquivos abertos.

```
<?php

$file = fopen("teste.txt","r");

//comandos ...

fclose($file);
```

```
?>
```

Exemplo 10.3

Verificando Final de Arquivo

O `feof()` função verifica se o "fim-de-arquivo" (EOF) tenha sido atingido. O `feof()` função é útil para loop através de dados de comprimento desconhecido.

```
<?php

$file = fopen("teste.txt","r");

if (feof($file)) echo "End of file";

fclose($file);

?>
```

Exemplo 10.4

Mais Funções de Arquivos

A função `fgets()` é usado para ler uma única linha de um arquivo.

ⓘ Depois de uma chamada para esta função o ponteiro do arquivo mudou-se para a próxima linha.

```
<?php

$arq = fopen("welcome.txt", "r") or exit("Não é possível abrir o arquivo!");

while(!feof($arq))

  {

  echo fgets($arq). "<br />";

  }

fclose($arq);

?>
```

Exemplo 10.5

A função `fwrite()` grava dados num arquivo.

A função vai parar no final do arquivo ou quando atinge o comprimento especificado, o que ocorrer primeiro.

Esta função retorna o número de bytes escritos, ou FALSE em falhas.

```
<?php

$file = fopen("test.txt","w");
```

```
echo fwrite($file,"Hello World. Testing!");

fclose($file);

?>
```

Exemplo 10.6

Exercícios

1. Execute o exemplo 10.5. Explique o que o exemplo faz.

   _____

2. Execute o exemplo 10.6. Explique o que o exemplo faz.

   _____

Capítulo 11 – Banco de Dados

Nesse capítulo estudaremos mecanismos da linguagem para conexão e manipulação de banco de dados. Listaremos comandos e exemplos tanto no MySQL como no Microsoft SQL Server.

Como vimos anteriormente, o PHP suporta várias bases de dados. O PHP possui suporte através de camadas de abstração e de extensões específicas de diversas bases de dados. Algumas bases aceitam acessos tanto pela camada de abstração como por extensões. A seguir temos uma lista das plataformas suportadas pelo PHP.

Camadas de Aplicação:

- DBA
- dbx
- ODBC
- PDO

Extensões específicas:

- CUBRID
- dBase
- DB++
- FrontBase
- filePro
- Firebird/InterBase
- Informix
- IBM DB2
- Ingres
- MaxDB
- Mongo
- mSQL
- Mssql
- MySQL
- Mysqli
- Mysqlnd
- mysqlnd_ ms
- mysqlnd_ qc
- OCI8
- Ovrimos SQL
- Paradox
- PostgreSQL
- SQLite
- SQLite3
- Sybase
- tokyo_ tyrant

No site do PHP encontramos maiores detalhes sobre a instalação de drivers, configuração e funções relacionadas a cada uma das plataformas (http://br2.php.net/manual/pt_BR/refs.database.php).

Abordaremos em aula o acesso ao Microsoft SQL Server, por ser a plataforma de estudo de outras disciplinas do curso. Nesta apostila incluímos o Anexo III com informações de acesso ao banco de dados através da extensão do MySQL.

Conectando ao Banco de Dados

Antes de qualquer consulta ao banco de dados, é necessário abrir uma conexão. Nesse procedimento passaremos informações do servidor e autenticação.

Para o SQL Server, usamos a seguinte função:

```
sqlsrv_connect( string $serverName [, array $connectionInfo])
```

Onde:

- serverName: uma cadeia de caracteres que especifica o nome do servidor com o qual uma conexão está sendo estabelecida. É possível incluir um nome de instância (por exemplo, "meuServidor\nomedaInstância") ou um número de porta (por exemplo, "meuServidor, 1521") como parte dessa cadeia de caracteres.
- connectionInfo: uma array associativa que contém atributos de conexão (por exemplo, array("Database" => "Compras")). A tabela a seguir descreve as opções dessa opção.

Chave	Valor	Descrição	Padrão
APP	Cadeia de caracteres	O nome do aplicativo usado no rastreamento.	Nenhum valor definido.
CharacterSet	Cadeia de caracteres	Especifica o conjunto de caracteres usado para enviar dados ao servidor. Essa chave é nova no Driver do SQL Server para PHP versão 1.1.	SQLSRV_ENC_CHAR
ConnectionPooling	1 ou **true** para o pool de conexões ativado. 0 ou **false** para o pool de conexões desativado.	Especifica se a conexão é atribuída a partir de um pool de conexões (1 ou **true**) ou não (0 ou **false**).	**true** (1)
Database	Cadeia de caracteres	Especifica que o nome do banco de dados em uso para a conexão estabelecida1.	O banco de dados padrão para o logon que está sendo utilizado.
Encrypt	1 ou **true** para criptografia ativada. 0 ou **false** para criptografia desativada.	Especifica se a comunicação com o SQL Server é criptografada (1 ou **true**) ou não (0 ou **false**).	**false** (0)
Failover_Partner	Cadeia de caracteres	Especifica o servidor e a instância do espelho do banco de dados (se habilitado e configurado) a ser usado quando o servidor primário está indisponível.	Nenhum valor definido.
LoginTimeout	Inteiro	Especifica o número de segundos a aguardar antes de a tentativa de conexão falhar.	Nenhum tempo limite.
MultipleActiveResultSets	1 ou **true** para usar vários conjuntos de resultados ativos. 0 ou **false** para desabilitar vários conjuntos de resultados ativos.	Permite a você desabilitar ou habilitar explicitamente o suporte a MARS (vários conjuntos de resultados ativos). Essa chave é nova no Driver do SQL Server para PHP versão 1.1.	**false** (0)
PWD	Cadeia de caracteres	Especifica a senha associada à identificação do usuário a ser	Nenhum valor definido.

		usada ao se conectar com a Autenticação do SQL Server3.	
QuotedId	1 ou **true** para usar regras SQL-92. 0 ou **false** para usar regras herdadas.	Especifica se devem ser usadas regras SQL-92 para identificadores entre aspas (1 ou **true**) ou regras Transact-SQL herdadas (0 ou **false**).	**true** (1)
ReturnDatesAsStrings	1 ou **true** para recuperar tipos de data e hora como cadeias de caracteres. 0 ou **false** para recuperar tipos de data e hora como tipos PHP **DateTime**.	É possível recuperar tipos de data e hora (datetime, date, time, datetime2 e datetimeoffset) como cadeias de caracteres ou tipos PHP.	**false**
TraceFile	Cadeia de caracteres	O caminho do arquivo usado para dados de rastreamento.	Nenhum valor definido.
TraceOn	1 ou **true** para habilitar o rastreamento. 0 ou **false** para desabilitar o rastreamento.	Especifica se o rastreamento ODBC está habilitado (1 ou **true**) ou desabilitado (0 ou **false**) para a conexão que está sendo estabelecida.	**false** (0)
TransactionIsolation	SQLSRV_TXN_READ_UNCOMMITTED, SQLSRV_TXN_READ_COMMITTED, SQLSRV_TXN_REPEATABLE_READ, SQLSRV_TXN_SNAPSHOT, SQLSRV_TXN_SERIALIZABLE	Especifica o nível de isolamento da transação.	SQLSRV_TXN_READ_COMMITTED
TrustServerCertificate	1 ou **true** para confiar no certificado. 0 ou **false** para não confiar no certificado.	Especifica se o cliente deve confiar (1 ou **true**) ou rejeitar (0 ou **false**) um certificado de servidor auto-assinado.	**false** (0)
UID	Cadeia de caracteres	Especifica a identificação do usuário a ser usada ao se conectar com a Autenticação do SQL Server3.	Nenhum valor definido.
WSID	Cadeia de caracteres	O nome do computador para o rastreamento.	Nenhum valor definido.

Abaixo temos um exemplo para abertura de conexão com o banco de dados:

```
<?php

$serverName = "regulus";

$uid = "admin";

$pwd = "senha";

$database="vendas";

$connectionInfo = array( "Database"=>$database, "PWD"=>$pwd, "UID"=>$uid);
```

```
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )

{

     echo "Conexão estabelecida com sucesso.\n";

}

else

{

     echo "Falha na conexão com o banco de dados.\n";

}

?>
```

Exemplo 10.1

Obs.: Podemos encontrar as funções acima com um @ prefixado, como `@mysql_select_db($database)`. Esse símbolo denota o modo silencioso, o que significa que a função não retornará nenhuma mensagem na falha, como medida de segurança.

Será preciso selecionar um banco de dados toda vez que fizermos uma conexão, o que significa uma vez por página ou todas as vezes que alterar o bando de dados. Caso contrário, aparecerá um erro: `Database not selected`.

Fazendo Consultas

Através do Driver do SQL Server, podemos submeter consultas SQL de duas formas:

- **sqlsrv_query():** é método padrão, recomendado para consultas únicas. Essa função fornece um método simplificado para executar uma consulta com uma quantidade de código mínima. A função sqlsrv_query realiza a preparação e a execução da instrução, e pode ser usada para executar consultas com parâmetros.
- **sqlsrv_prepare()** e **sqlsrv_execute():** essa combinação de funções separa a preparação da instrução e a execução da instrução em duas chamadas de função e pode ser usadas para executar consultas com parâmetros. Essa função é ideal para executar uma instrução várias vezes com valores de parâmetros diferentes para cada execução.

Como executar uma única consulta

No exemplo abaixo vemos que a função **sqlsrv_query()** fornece um método simplificado para executar uma consulta com codificação mínima. Essa função realiza a preparação e a execução da instrução, e pode ser usada para executar consultas com parâmetros.

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
```

```
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o banco de dados.\n";

}


/* Configuração da query SQL */
$sql = "INSERT INTO Aluno
         (RA, Nome, Curso)
         VALUES
         ('11002', 'Ana Júlia', '1')";


/* Preparação e execução da query. */
$stmt = sqlsrv_query( $conexao, $sql);

if( $stmt )
{
     echo "Inserção realizada com sucesso.\n";
}
else
{
     echo "Falha na inserção.\n";
     die( print_r( sqlsrv_errors(), true));
}

/* Liberar statement e fechar conexão */
sqlsrv_free_stmt( $stmt);
sqlsrv_close( $conexao);

?>
```

Exemplo 10.2

No exemplo abaixo podemos notar ainda outra variação do uso da função `sqlsrv_query()` com o acréscimo da variável `$params`.

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o banco de dados.\n";

}
```

```
/* Configuração da query SQL */
$sql = "INSERT INTO Aluno
        (RA, Nome, Curso, Periodo)
        VALUES
        (?, ?, ?)";

/* Configuração dos parâmetros. */
$params = array("10003", "Maria", "2", "M" );


/* Preparação e execução da query. */
$stmt = sqlsrv_query( $conexao, $sql, $params);

if( $stmt )
{
     echo "Inserção realizada com sucesso.\n";
}
else
{
     echo "Falha na inserção.\n";
     die( print_r( sqlsrv_errors(), true));
}

/* Liberar statement e fechar conexão */
sqlsrv_free_stmt( $stmt);
sqlsrv_close( $conexao);

?>
```

Exemplo 10.3

Como executar uma consulta várias vezes

Usando a combinação de `sqlsrv_prepare` e `sqlsrv_execute`, uma instrução é compilada uma vez no servidor e pode então ser executada várias vezes com valores de parâmetros diferentes.

Ao preparar uma instrução que usa variáveis como parâmetros, as variáveis são associadas à instrução. Isso significa que, se você atualizar os valores das variáveis, da próxima vez que executar a instrução, ela será executada com valores de parâmetros atualizados.

O exemplo abaixo faz uso desse recurso diante da necessidade de alterar todos os registros da tabela Aluno diante de uma determinada situação. Vamos considerar a seguinte tabela no bando de dados, onde temos os cursos 1 e 2 oferecidos nos períodos manhã e tarde:

RA	Nome	Curso	Período
10005	Ana Beatriz	1	M
10003	João	1	M
10004	Pedro	2	T
10007	Luiza	1	M
10008	Carlos	2	M
10002	Paulo	2	M
10010	André	1	M

Vamos imaginar que o curso 1, passará a ser oferecido apenas no período da manhã e o curso 2 passará a ser oferecido apenas no período da tarde. Nesse caso, necessitaremos realizar uma consulta (UPDATE na tabela) para cada registro cujo campo CURSO seja igual a "1" o campo PERÍODO deverá ser

"M" e para cada registro cujo campo CURSO seja igual a "2" o campo PERÍODO deverá ser "T". Para isso teremos o seguinte exemplo:

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o banco de dados.\n";

}


/* Configuração da query SQL */
$tsql = "UPDATE Aluno SET Periodo = ?
        WHERE Curso = ?";

/* Ajustes dos parâmetros. Variáveis $periodo
e $curso são adicionadas à sentença, $stmt1. */
$periodo = "'0'";
$curso = "'0'";
$stmt1 = sqlsrv_prepare( $conexao, $tsql, array( &$periodo, &$curso));
if( $stmt1 )
{
     echo "Sentença 1 preparada.\n";
}
else
{
     echo "Erro na preparação do Statement.\n";
     die( print_r( sqlsrv_errors(), true));
}

/* Configuração das informações de Curso e Período. Este array
mapeará maps the order ID to order quantity in key=>value pairs. */
$valores = array( "M"=>"1", "T"=>"2");


/* Executar a sentença para cada período. */
foreach( $valores as $periodo => $curso)
{
     // Os valores das variaáveis $periodo e $curso, serão atualizados
     // e executados em cada execução da sentença.

     if( sqlsrv_execute( $stmt1) === false )
     {
          echo "Erro na execução da sentença;.\n";
          die( print_r( sqlsrv_errors(), true));
     }
}
echo "Registros atualizados.\n";
```

```
/* Recursos de $stmt1 liberados.  Esto permitirá que as variáveis $periodo e
$curso sejam alocados a diferentes sentenças.*/
sqlsrv_free_stmt( $stmt1);
sqlsrv_close( $conn);

?>
```

Exemplo 10.4

Recuperando Dados

É possível recuperar uma linha de dados como uma matriz, como um objeto ou como um único campo em uma linha de dados. Os tópicos nesta seção examinam as diversas maneiras como os dados podem ser recuperados.

Dependendo da aplicação, podemos escolher como os dados serão recuperados, de acordo com as seguintes possibilidades:

Como recuperar dados como uma matriz	Demonstra como recuperar uma linha de dados como uma matriz (como uma matriz indexada numericamente e como uma matriz associativa) e como processar os dados retornados.
Como recuperar dados como um objeto	Demonstra como recuperar uma linha de dados como um objeto PHP e processar os dados retornados.
Como recuperar um único campo	Demonstra como recuperar um único campo de uma linha do conjunto de resultados e processar os dados retornados.
Como trabalhar com vários conjuntos de resultados	Demonstra como usar mover-se de um conjunto de resultados de uma consulta em lotes para o próximo.
Como detectar conjuntos de resultados vazios	Demonstra como determinar se um conjunto de resultados está vazio.
Recuperando dados como um fluxo	Fornece uma visão geral de como transmitir dados do servidor e fornece links para casos de uso específicos.
Como recuperar o tipo de data e hora como cadeias de caracteres	Descreve como recuperar tipos de data e hora como cadeias de caracteres.

Fonte: http://msdn.microsoft.com/pt-br/library/cc296160%28v=SQL.90%29.aspx

Como recuperar dados como uma matriz

A função `sqlsrv_fetch_array()` para recuperar uma linha de dados como uma matriz. Podemos recuperar de duas formas:

- para recuperar uma linha de dados como uma matriz indexada numericamente.
- para recuperar uma linha de dados como uma matriz associativa.

No exemplo a seguir, cada linha de um conjunto de resultados é recuperada como uma matriz indexada numericamente (`$row[0]`).

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o banco de dados.\n";

}


/* Configuração da query SQL */
$stmt = "SELECT * FROM Aluno";


/* Preparação e execução da query. */
$stmt = sqlsrv_query( $conexao, $stmt, $params);

if( $stmt )
{
     echo "Sentença exucutada com sucesso.\n <BR>";
}
else
{
     echo "Falha na execução da setença.\n";
     die( print_r( sqlsrv_errors(), true));
}

/* Iterate through the result set printing a row of data upon each
iteration.*/
while( $row = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_NUMERIC))
{
     echo "RA: ".$row[0]."\n";
     echo "Nome: ".$row[1]."\n";
     echo "Curso: ".$row[2]."\n";
     echo "Período: ".$row[3]."\n<BR>";
}

/* Liberar sentença e fechar conexão */
sqlsrv_free_stmt( $stmt);
sqlsrv_close( $conexao);

?>
```

Exemplo 10.5

Como recuperar dados como um objeto

Podemos recuperar dados do banco através da utilização de objetos. Através da função `sqlsrv_fetch_object()` , é possível armazenar os dados consultados em objetos. É possível também,

criar classes com propriedades (que poderiam ser os campos da tabela de uma base de dados) e métodos que manipulam esses dados para serem trabalhos no programa PHP. Maiores detalhes dessa função, podemos encontrar em http://msdn.microsoft.com/pt-br/library/cc626308%28v=SQL.90%29.aspx.

Abaixo temos um exemplo onde é realizada uma consulta no banco. O retorno dessa consulta poderá ter uma ou várias linhas (registros). Como a função `sqlsrv_fetch_object()` foi colocada no while, cada loop chamará a função que retornará uma linha da consulta. A diferença para a função que vimos anteriormente é que os dados retornados serão manipulados através do conceito de objetos (`$obj->Nome`).

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o banco de dados.\n";

}


/* Configurar e executar a query. */
$tsql = "SELECT Nome
         FROM Aluno";
$stmt = sqlsrv_query( $conexao, $tsql);

if( $stmt === false )
{
     echo "Erro na preparação/execução da query<BR>\n";
     die( print_r( sqlsrv_errors(), true));
}

/* Receber cada linha como objeto do PHP*/
while( $obj = sqlsrv_fetch_object( $stmt))
{
      echo $obj->Nome."<BR> \n";
}

/* Liberar recursos da sentença e conexão. */
sqlsrv_free_stmt( $stmt);
sqlsrv_close( $conexao);
?>
```

Exemplo 10.6

Como recuperar um único campo

Através da função `sqlsrv_fetch_field()` podemos recuperar um único campo de um registrp de uma tabela do banco de dados. Recupera dados do campo especificado da linha atual. Os dados do campo devem ser acessados na ordem. Por exemplo, não será possível acessar os dados do primeiro campo depois

que os dados do segundo campo tiverem sido acessados. Maiores detalhes dessa função, podemos encontrar em http://msdn.microsoft.com/pt-br/library/cc296207%28v=SQL.90%29.aspx.

Abaixo temos um exemplo onde é realizada uma consulta no banco. O retorno dessa consulta poderá ter uma ou várias linhas (registros). Como a função `sqlsrv_fetch_object()` foi colocada no while, cada loop chamará a função que retornará uma linha da consulta. A diferença para a função que vimos anteriormente é que os dados retornados serão manipulados através do conceito de objetos (`$obj->Nome`).

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o servidor de banco de dados.\n";

}

/* Configurar e executar a query. */
$tsql = "SELECT * FROM Aluno";
$stmt = sqlsrv_query( $conexao, $tsql);

if( $stmt === false )
{
     echo "Erro na preparação/execução da query<BR>\n";
     die( print_r( sqlsrv_errors(), true));
}


/* Disponibiliza o resultado de cada linha */
while( !sqlsrv_fetch( $stmt ) === false )
{

    /* Nota: Os campos devem ser acessados em ordem. */
    $Nome = sqlsrv_get_field( $stmt, 1);
    echo $Nome."<BR>";

}


/* Liberar recursos da sentença e conexão. */
sqlsrv_free_stmt( $stmt);
sqlsrv_close( $conexao);
?>
```

Exemplo 10.7

Como trabalhar com vários conjuntos de resultados

A função `sqlsrv_next_result()` ativa o próximo resultado (conjunto de resultados, contagem de linhas ou parâmetro de saída) da instrução especificada.. Maiores detalhes dessa função, podemos encontrar em http://msdn.microsoft.com/pt-br/library/cc296167%28v=SQL.90%29.aspx.

Abaixo temos um exemplo onde é realizada uma consulta no banco. O retorno dessa consulta poderá ter uma ou várias linhas (registros). Como a função `sqlsrv_next_result()` permitirá que se avance as linhas retornadas de uma consulta através de uma solução diferente do vimos até aqui.

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o servidor de banco de dados.\n";

}

/* Configurar e executar a query. */
$tsql = "--Query 1
             SELECT * FROM Aluno;

             --Query 2
         INSERT INTO Aluno (RA,
                            Nome,
                            Curso,
                            Periodo)
         VALUES ('10010','Bruno', '1', 'M' ); ";


$stmt = sqlsrv_query( $conexao, $tsql);

if( $stmt === false )
{
     echo "Erro na preparação/execução da query<BR>\n";
     die( print_r( sqlsrv_errors(), true));
}


/* Resultado da query 1 */
echo "Resultado da query 1:<BR>\n";
while( $linha = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_NUMERIC ))
{
     print_r($linha);
}


/* Move para o próximo resultado da query. */
sqlsrv_next_result($stmt);
```

```
/* Exibe o resultado da segunda query (INSERT). */
echo "Resultado da query 2: <BR>\n";
echo "Linhas acrescentadas: ".sqlsrv_rows_affected($stmt)."\n";



/* Liberar recursos da sentença e conexão. */
sqlsrv_free_stmt( $stmt);
sqlsrv_close( $conexao);
?>
```

Exemplo 10.8

Como detectar conjuntos de resultados vazios

A prática recomendada para determinar se o conjunto de resultados de uma instrução está vazio é chamar sqlsrv_fetch, sqlsrv_fetch_array ou sqlsrv_fetch_object na instrução e examinar o valor retornado. Se o conjunto de resultados ativo não contiver resultados, a chamada da função retornará **null**. Observe que, se você chamar uma dessas funções em uma instrução que não retorna um conjunto de resultados (e não uma instrução que retorna um conjunto de resultados vazio), ela retornará **false**.

Os exemplos acima mostram como os devidos tratamentos evitam que erros ocorram caso os resultados das consultas sejam nulos.

Recuperando dados como um fluxo

O PHP tira proveito dos fluxos para recuperar grandes quantidades de dados. As etapas a seguir resumem como recuperar dados como um fluxo:

1. Prepare e execute uma consulta Transact-SQL com sqlsrv_query ou com a combinação de `sqlsrv_prepare/sqlsrv_execute`.
2. Use `sqlsrv_fetch` para mover-se para a próxima linha no conjunto de resultados.
3. Use `sqlsrv_get_field` para recuperar um campo da linha. Especifique que os dados devem ser recuperados como um fluxo usando SQLSRV_PHPTYPE_STREAM(<encoding>) como o terceiro parâmetro na chamada da função. Esta tabela lista as constantes usadas para especificar as codificações e suas descrições:

Constante SQLSRV	Descrição
SQLSRV_ENC_BINARY	Os dados são retornados como um fluxo de bytes brutos do servidor sem executar a codificação ou a conversão.
SQLSRV_ENC_CHAR	Os dados são retornados em caracteres de 8 bits, como especificado na página de código da localidade do Windows definida no sistema. Todos os caracteres multibyte ou os caracteres que não estão mapeados nessa página de código são substituídos por um caractere de ponto de interrogação (?) de um byte.

Como recuperar o tipo de data e hora como cadeias de caracteres

É possível recuperar tipos de data e hora (`datetime`, `date`, `time`, `datetime2` e `datetimeoffset`) como cadeias de caracteres, especificando uma opção na cadeia de conexão.

Use a seguinte opção de conexão:

```
'ReturnDatesAsStrings'=>true
```

O padrão é false, que significa que os tipos `datetime`, `Date`, `Time`, `DateTime2` e `DateTimeOffset` serão retornados como tipos PHP **Datetime**. Exemplo:

```
<?php

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd,
"ReturnDatesAsStrings"=>true);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao )
{
     echo "Conexão estabelecida com sucesso.\n";
}
else
{
     echo "Falha na conexão com o servidor de banco de dados.\n";

}

$tsql = "SELECT Data FROM Aluno";

$stmt = sqlsrv_query( $conexao, $tsql);

if ( $stmt === false ) {
   echo "Erro na preparação/execução da sentença.\n";
   die( print_r( sqlsrv_errors(), true));
}

sqlsrv_fetch( $stmt );

// Obtem a data como string
$data = sqlsrv_get_field( $stmt, 0, SQLSRV_PHPTYPE_STRING("UTF-8"));

if( $data === false ) {
   die( print_r( sqlsrv_errors(), true ));
}

echo $data;

sqlsrv_close( $conexao);
?>
```

Exemplo 10.9

Atualizando Dados

Veremos nesta seção como atualizar dados em um banco de dados examinando os casos de uso comuns.

As etapas para usar o Driver do SQL Server para PHP para atualizar dados em um banco de dados podem ser resumidas da seguinte maneira:

1. Defina uma consulta Transact-SQL que executa uma operação de inserção, atualização ou exclusão.
2. Atualize valores de parâmetros para consultas com parâmetros.
3. Use `sqlsrv_query` ou `sqlsrv_prepare/sqlsrv_execute` para executar consultas Transact-SQL com os valores de parâmetros atualizados (se aplicável).

Tratando Erros e Avisos

Por padrão, o Driver do SQL Server para PHP trata avisos como erros; uma chamada de uma função sqlsrv que gera um erro ou um aviso retornará false. Este tópico demonstra como desativar esse comportamento padrão e como tratar avisos separadamente de erros.

No exemplo abaixo temos as funções `DisplayErrors() e DisplayWarnings()` que tratam e exibem erros e *warnings* separadamente. Nas fases de desenvolvimento e testes ajudam a identificar problemas para as correções necessárias.

```
<?php

/* Desabilita a configuração padrão de exibição de erros.
Se a configuração falhar, exibe a mensagem de erro e finaliza o script.*/
if( sqlsrv_configure("WarningsReturnAsErrors", 0) === false)
{
     DisplayErrors();
     exit;
}

$serverName = "regulus";
$uid = "admin";
$pwd = "senha";
$database="teste";
$connectionInfo = array( "UID"=>$uid,"Database"=>$database, "PWD"=>$pwd);
$conexao = sqlsrv_connect($serverName, $connectionInfo);

if( $conexao === false )
{
     DisplayErrors();
     die;
}
/* Display any warnings. */
DisplayWarnings();

/* ------------- Error Handling Functions --------------*/
function DisplayErrors()
{
     $errors = sqlsrv_errors(SQLSRV_ERR_ERRORS);
     foreach( $errors as $error )
     {
```

```
            echo "Error: ".$error['message']."\n";
      }
}

function DisplayWarnings()
{
      $warnings = sqlsrv_errors(SQLSRV_ERR_WARNINGS);
      if(!is_null($warnings))
      {
            foreach( $warnings as $warning )
            {
                  echo "Warning: ".$warning['message']."\n";
            }
      }
}

 ?>
```

Exemplo 10.10

Considerações Sobre Segurança

As questões sobre segurança tratadas têm como base o site sobre o Driver PHP da Microsoft no link http://msdn.microsoft.com/pt-br/library/cc296190%28v=SQL.90%29.aspx.

Conectar-se usando a Autenticação do Windows

A Autenticação do Windows deve ser usada para se conectar ao SQL Server sempre que possível, pelos seguintes motivos:

* Nenhuma credencial passa pela rede durante a autenticação. Os nomes de usuário e as senhas não são incorporados na cadeia de conexão do banco de dados. Isso significa que os invasores ou usuários mal-intencionados não podem obter as credenciais por meio do monitoramento da rede ou da exibição das cadeias de conexão nos arquivos de configuração.

* Os usuários estão sujeitos ao gerenciamento centralizado de contas. São impostas diretivas de segurança, como períodos de validade da senha, comprimentos mínimos de senha e bloqueio de contas depois de várias solicitações de logon inválidas.

Ao se conectar usando a Autenticação do Windows, é recomendável configurar seu ambiente de forma que o SQL Server possa usar o protocolo de autenticação Kerberos. Para obter mais informações, consulte a página Como verificar se você está usando a autenticação Kerberos ao criar uma conexão remota com uma instância do SQL Server 2005 ou Kerberos Authentication and SQL Server.

Usar conexões criptografadas ao transferir dados confidenciais

Conexões criptografadas deverão ser usadas sempre que dados confidenciais forem enviados ou recuperados do SQL Server. Para obter informações sobre como habilitar as conexões criptografadas, consulte a página Como habilitar conexões criptografadas ao Mecanismo de Banco de Dados (SQL Server Configuration Manager). Para estabelecer uma conexão segura com o Driver do SQL Server para PHP, use o atributo de conexão Encrypt ao se conectar ao servidor. Para obter mais informações sobre os atributos de conexão, consulte sqlsrv_connect.

Usar consultas com parâmetros

Use consultas com parâmetros para reduzir o risco de ataques de injeção de SQL. Para obter exemplos de como executar consultas com parâmetros, consulte Como executar consultas com parâmetros, Como executar uma única consulta e Como executar uma consulta várias vezes.

Para obter mais informações sobre ataques de injeção de SQL e as considerações de segurança relacionadas, consulte a página SQL Injection.

Não aceitar informações sobre cadeia de conexão ou servidor de usuários finais

Escreva os aplicativos de forma que os usuários finais não possam enviar informações da cadeia de conexão ou do servidor para o aplicativo. A manutenção de um controle rígido sobre essas informações reduz a área de superfície para atividades mal-intencionadas.

Ative WarningsAsErrors durante o desenvolvimento do aplicativo

Desenvolva aplicativos com a configuração WarningsAsErrors definida como true para que os avisos emitidos pelo driver sejam tratados como erros. Isso permitirá que você resolva avisos antes de implantar seu aplicativo. Para obter mais informações, consulte Tratando erros e avisos.

Logs seguros para aplicativo implantado

Para aplicativos implantados, verifique se os logs são gravados em um local seguro ou se o recurso de registro está desativado. Isso o ajuda a se proteger contra a possibilidade de usuários finais acessarem informações gravadas nos arquivos do log. Para obter mais informações, consulte Registrando atividades.

Maiores detalhes sobre as considerações acima podem ser consultadas nos sites listados na Seção Bibliografia.

Exercícios

1. Crie a tabela abaixo no servidor de banco de dados:

```
CREATE TABLE Aluno (
   RA VARCHAR( 5 ) ,
   Nome VARCHAR( 20 ) ,
   Curso VARCHAR( 1 ) ,
   Periodo VARCHAR( 1 ) )
```

Teste o exemplo 10.3 acrescentando os seguintes dados:

RA	Nome	Curso	Período
10005	Ana Beatriz	1	M
10003	João	1	M
10004	Pedro	2	T
10007	Luiza	T	M
10008	Carlos	1	M
10002	Paulo	2	T
10010	André	2	T

2. Crie uma tabela que tenha um campo do tipo data e testar o exemplo 10.9.

Anexo I – Lista de Servidores Web

Fonte: Wikipédia, a enciclopédia livre.

- Apache HTTP Server
- BadBlue
- Boa
- Caudium, uma derivação do Roxen
- Covalent Enterprise Ready Server, baseado no Apache HTTP Server
- Fnord
- IBM HTTP Server (baseado no Apache HTTP Server), antigo Domino Go Webserver
- Internet Information Services (IIS) da Microsoft, incluso no Windows XP
- Light HTTP Server (lighttpd)
- NaviServer
- Nginx
- Oracle HTTP Server, baseado no Apache HTTP Server
- Roxen
- Sun Java System Web Server da Sun Microsystems, antigo Sun ONE Web Server, iPlanet Web Server, and Netscape Enterprise Server.
- thttpd da ACME Laboratories
- Zeus Web Server
- AOLWebServer

Anexo II – Lista de Linguagens de Script

Origem: Wikipédia, a enciclopédia livre.

Abaixo, segue-se algumas linguagens de programação que são tipicamente **usadas como** script (não sendo, necessariamente, apenas de script):

- ActionScript
- BASIC
- C e derivados
- Euphoria
- JavaScript
- Lua
- PHP
- Python
- Ruby
- Tcl
- VBScript

Anexo III – Banco de Dados com MySQL

Conectando ao Banco de Dados

O comando básico para iniciar uma conexão tanto no MySQL é o seguinte:

```
mysql_connect($hostname, $user, $password);
```

Onde temos:

- `$hostname`: variável com nome do servidor de Banco de Dados
- `$user`: variável com *username* usado para conectar ao servidor de Banco de Dados
- `$password`: variável com senha usada para conectar ao servidor de Banco de Dados

Podemos também inserir diretamente *strings* literais, sendo o argumento de senha opcional:

```
mysql_connect('localhost', 'root', 'teste123');
```

Depois disso, devemos selecionar o banco de dados que iremos trabalhar:

```
mysql_select_db($database);
```

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$db_selected = mysql_select_db(agenda, $link);
?>
```

Exemplo 1.1 – Anexo III

Fazendo Consultas

Podemos fazer uma simples consulta através do comando abaixo:

```
mysql_query("SELECT Nome FROM Clientes WHERE ID<10");
```

Em caso de erro na tentativa de executar o comando acima, se colocarmos a *string* em uma variável e depois inseri-la na função, teremos mais chances de identificar onde ocorreu o problema. Exemplo:

```
<?php

$conec=mysql_connect("localhost", "root");
mysql_select_db("Agenda");

$declar="SELECT Nome FROM tblAlunos WHERE RA='01234'";
$result=mysql_query($declar);  // No MySQL

?>
```

Exemplo 1.2 – Anexo III

Se a consulta foi um INSERT, UPDATE, DELETE, CREATE TABLE ou DROP TABLE e retornou TRUE, é possível utilizar a função `mysql_affected_rows()` para saber quantas linhas foram alteradas pela consulta. Isso porque tais consultas alteram o banco mas não retornam dados para verificarmos e concluirmos que a operação foi bem sucedida. Essa função opcionalmente recebe um identificador de link. Exemplo:

```
$affected_rows = mysql_affected_rows();
```

Exemplo 1.3 – Anexo III

Se a consulta foi uma instrução SELECT, o inteiro retornado terá um significado ligeiramente diferente: em vez de TRUE ou FALSE, ele retorna um inteiro chamado de *identificador de resultado*. No caso do SELECT utilizamos a função `mysql_num_rows()` para descobrir quantas linhas foram retornadas por uma SELECT bem-sucedida.

Buscando Dados

No PHP é preciso acrescentar instruções para buscar (*fetch*) os dados. Seria lógico supor que o resultado de uma consulta fosse os dados desejados, mas isso não é o correto. Como vimos anteriormente, o resultado de uma consulta do PHP é um **inteiro** que representa o **sucesso** ou **falha** ou a identidade da consulta.

Na realidade, a função `mysql_query()` recupera os dados do banco de dados e envia um recibo de volta para o PHP informando o status da operação. A essa altura, os dados existem em um "purgatório" que não é imediatamente acessível nem no MySQL nem no PHP. Para resgatarmos esses dados, precisamos usar uma das funções `mysql_fetch` para tornar os dados completamente disponíveis para o PHP. Abaixo listamos algumas funções de busca:

- `mysql_fetch_row`: retorna a linha como um *array* enumerado.
- `mysql_fetch_object`: retorna a linha como um objeto.
- `mysql_fetch_array`: retorna a linha como um *array* associativo.

As diferenças entre as três principais funções de busca são pequenas. Uma das mais comuns é a `mysql_fetch_row`, que pode ser utilizada de acordo com o exemplo abaixo:

```
$declar = "SELECT ID, LastName, FirstName FROM users WHERE Status = 1";
$result = mysql_query($declar);
While ($linha = mysql_fetch_row($result)) {        // no MySQL
   Print("$linha[0] $linha[1] $linha[2]<BR>");
}
```

Exemplo 1.4 – Anexo III

A função `mysql_fetch_object` realiza quase a mesma tarefa, exceto pelo fato de a linha ser retornada como um objeto em vez de um *array*. Exemplo:

```
$declar = "SELECT ID, LastName, FirstName FROM users WHERE Status = 1";
$result = mysql_query($declar);
While ($linha = mysql_fetch_object($result)) {        // no MySQL
   Print("$linha->ID, $linha->LastName, $linha->FirstName <BR>");
}
```

Exemplo 1.5 – Anexo III

A função de busca `mysql_fetch_array`, oferece a escolha de resultados como um *array* associativo ou um enumerado, ou ambos, que é o padrão. Isso significa que podemos referenciar as saídas pelo nome de campo de banco de dados em vez de número:

```
$declar = "SELECT ID, LastName, FirstName FROM users WHERE Status = 1";
$result = mysql_query($declar);
While ($linha = mysql_fetch_array($result)) {        // no MySQL
   Print("$linha['ID'] $linha['LastName'] $linha['FirstName']<BR>");
}
```

Exemplo 1.6 – Anexo III

Diversas Conexões

Normalmente não é necessário requerer diversas conexões por página do PHP. Mesmo porque isso é uma das partes mais caras e que mais consomem tempo na maioria das consultas de banco de dados.

Não há nenhuma maneira fácil de manter sua conexão aberta de página para página, uma vez que o PHP e o MySQL nunca saberiam com certeza quando fechá-la depois que os visitantes se extraviassem. Portanto, a conexão é fechada no fim de cada script a menos que se utilize de conexões persistentes.

A razão principal por que utilizar conexões diferentes é no caso de se estar consultando dois ou mais bancos de dados completamente separados. A situação mais comum é num ambiente de bases replicadas.

Para utilizar múltiplas conexões, simplesmente abrimos conexões com cada banco de dados conforme necessário. O PHP utiliza identificadores de resultado para reter os resultados corretos. No exemplo abaixo temos conexões em três bancos de dados diferentes:

```
<?php

$link1 = mysql_connect('host1','admin1','senha');
mysql_select_db('UserDB',$link1);
$declar1 = "SELECT ID FROM tblUserTable WHERE username='$username'";
$result1 = mysql_query($declar1,$link1);
$array1 = mysql_fetch_array($result1);
$usercount = mysql_num_rows($result1);
mysql_close($link1);

$hoje = "2012-01-01";
$link2 = mysql_connect('host2','admin2','senha');
```

```
mysql_select_db('Inventario',$link2);
$declar2 = "SELECT sku FROM widgets WHERE ship_date='$today'";
$result2 = mysql_query($declar2,$link2);
$array2 = mysql_fetch_array($result2);
$widgetcount = mysql_num_rows($result2);
mysql_close($link2);

if ($usercount > 0 && $widgetcount > 0) {
   $link3 = mysql_connect('host3','admin3','senha');
   mysql_select_db('salesdb',$link3);
   $declar3 = "INSERT INTO saleslog (ID, date, userID, sku) VALUES
(NULL, '$today', 'array1[0], 'array2[0]')";
   $result3 = mysql_query($declar3,$link3);
   $insertID = mysql_insert_id($link3);
   mysql_close($link3);
   if ($insertID >= 1) {
      print ("Inserção realizada com sucesso");
   }
   else {
      print ("Erro na inserção");
   }
}
else {
   print ("Nenhuma informação");
}

?>
```

Exemplo 1.6 – Anexo III

Verificação de Erro

A principal função de verificação de erro usada nas operações de banco de dados é `exit()`. Essa função retorna uma mensagem de erro e encerra o *script*. É bastante útil no caso de falhas onde é possível personalizar as mensagens de erro. Isso nos traz duas vantagens:

- identificar onde exatamente ocorreu o erro de acordo com a mensagem exibida (que nós escolhemos na função `exit()`)
- evitar que informações do banco de dados e do próprio programa sejam exibidas no navegador para o usuário se ocorrer alguma falha

No exemplo abaixo vemos que a função será terminada e caso ocorra algum erro será exibido uma *string*:

```
mysql_query ("SELECT * FROM alunos WHERE RA = '$RA'")
    or exit("Favor checar a delaração SQL e tentar novamente.");
```

Exemplo 1.7 – Anexo III

Dependendo do banco de dados utilizado, as mensagens de erro são um tanto quanto difíceis de interpretar onde exatamente estaria o erro.

Exibindo Consultas em Tabelas

As tabelas de HTML são construídas com linhas. A construção <TR></TR> e as colunas não têm existência independente, sendo que cada linha tem alguma variedade de itens de dados de tabela. Inversamente, os campos ou colunas nas tabelas de banco de dados são a entidade mais primária. Definir uma tabela significa definir os campos e então podemos adicionar quantas linhas quisermos. Nessa seção, focalizaremos a impressão de tabelas e consultas de tal maneira que cada campo de banco de dados imprima na sai própria coluna de HTML, simplesmente porque há normalmente mais linhas de banco de dados que campos de banco de dados.

O caso mais simples de exibição é onde a estrutura de uma tabela de banco de dados ou consulta realmente corresponde à estrutura da tabela de HTML que queremos exibir. A entidade banco de dados tem m colunas e n linhas e gostaríamos de exibir uma grade retangular m por n na janela de navegador do usuário, com todas as células preenchidas apropriadamente.

Exemplo: exibindo uma única tabela

Vamos escrever um tradutor simples que consulta o banco quanto ao conteúdo de uma única tabela e exibe os resultados na tela. Eis a estrutura top-down de como o código fará o trabalho:

1. Estabelece uma conexão de banco de dados.
2. Constrói uma consulta para enviar para o banco de dados.
3. Envia a consulta e espera o identificador de resultado que é retornado.
4. Utilizando o identificador de resultado, descobre quantas colunas (campos) estão em cada linha.
5. Inicia uma tabela em HTML.
6. Faz um loop pelas linhas de resultado do banco de dados, imprimindo um par <TR></TR> para fazer uma linha de tabela de HTML correspondente.
7. Em cada linha, recupera os campos sucessivos e os exibe empacotados em um par <TD></TD>.
8. Fecha a tabela de HTML.
9. Fecha a conexão de banco de dados.

Abaixo temos uma implementação dos passos descritos acima.

```
<?php
include("/home/phpbook/phpbook-vars.inc.php");
$global_dbh = mysql_connect($hostname, $username, $password);
mysql_select_db($db, $global_dbh);

function display_db_table($tablename, $connection)
{
  $query_string = "select * from $tablename";
  $result_id = mysql_query($query_string, $connection);
  $column_count = mysql_num_fields($result_id);

  print("<TABLE BORDER=1>\n");
  while ($row = mysql_fetch_row($result_id))
```

```
    {
      print("<TR ALIGN=LEFT VALIGN=TOP>");
      for ($column_num = 0;
           $column_num < $column_count;
           $column_num++)
        print("<TD>$row[$column_num]</TD>\n");
      print("</TR>\n");
    }
    print("</TABLE>\n");
}
?>

<HTML>
<HEAD>
<TITLE>Cities and countries</TITLE>
</HEAD>
<BODY>

<TABLE><TR><TD>
<?php display_db_table("country", $global_dbh); ?>
</TD><TD>
<?php display_db_table("city", $global_dbh); ?>
</TD></TR></TABLE></BODY></HTML>
```

Exemplo 1.8 – Anexo III

Algumas coisas a considerar sobre esse script:

- Embora o script refira-se a tabelas específicas de banco de dados, a própria função `display_tb_table()` é geral. Poderíamos colocar a definição de função em um arquivo include e depois utilizá-la em qualquer lugar do site.
- A primeira coisa que o script faz é se carregar com um arquivo include contendo atribuições de variáveis para o nome de banco de dados, o nome de usuário do banco de dados e a senha. Então, utilize essas variáveis para conectar so MySQL e depois escolher o banco de dados desejado.
- Na própria função, escolhemos utilizar um loop `while` para linhas e um `loop` for para imprimir os itens individuais. Poderíamos facilmente ter utilizado um loop for limitado para ambos e recuperado o número de linhas com `mysql_num_rows()`.

O script da listagem acima exibe o conteúdo da seguinte estrutura:

Country:
ID int (auto-incremented primary key)
Continent varchar(50)
Countryname varchar(50)

City:
ID int (auto-increment primary key)
CountryID int
cityName varchar(50)

O exemplo do script possui algumas limitações: funciona somente com uma tabela única, não faz verificação de erro e é muito básica em sua apresentação. Encaminharemos a solução desses problemas.

Exibindo Cabeçalhos de Coluna

Na primeira versão do script, não é exibido nenhum rótulo de que são os campos. É convencional em HTML utilizar as tags <TH> para coluna e/ou cabeçalhos de linha. Para utilizarmos os nomes dos campos nas tabelas dos bancos de dados podemos usar a função `mysql_field_name()`.

Verificação de Erro

Acrescentaremos chamadas á função die() para realizar as verificações de erro no script.

Questões Cosméticas

Podemos melhorar a aparência da página, oferecendo a opção, por exemplo, de exibir ou não as bordas da tabela. Para isso basta permitir a passagem de uma string de argumentos que será encaixada na definição de tabela HTML.

Exibindo Consultas Arbitrárias

Por fim, seria ótimo poder explorar nosso banco de dados relacional e exibir o resultado de consultas complexas em vez de somente tabelas únicas. Realmente, nossa página de tabela única tem uma consulta arbitrária embutida nela. Isso só acontece se ele for manualmente codificado como *select * from table*. Então vamos transformar a página de exibição de tabela simples para exibição de consulta e recriar a página de exibição de tabela como um empacotador simples em torno da exibição da consulta.

Abaixo temos um exemplo com as alterações descritas acima:

```
<?php
include("/home/phpbook/phpbook-vars.inc.php");
$global_dbh = mysql_connect($hostname, $username, $password)
                or die("Could not connect to database");

mysql_select_db($db, $global_dbh)
  or die("Could not select database");

function display_db_query($query_string, $connection,
                          $header_bool, $table_params)
{

  // perform the database query
  $result_id = mysql_query($query_string, $connection)
               or die("display_db_query:" . mysql_error());

  // find out the number of columns in result
  $column_count = mysql_num_fields($result_id)
                  or die("display_db_query:" . mysql_error());

  // TABLE form includes optional HTML arguments passed
  //  into function
  print("<TABLE $table_params >\n");

  // optionally print a bold header at top of table
  if ($header_bool)
```

```
  {
    print("<TR>");
    for ($column_num = 0;
          $column_num < $column_count;
          $column_num++)
    {
      $field_name =
         mysql_field_name($result_id, $column_num);
      print("<TH>$field_name</TH>");
     }
     print("</TR>\n");
  }
  // print the body of the table
  while ($row = mysql_fetch_row($result_id))
  {
    print("<TR ALIGN=LEFT VALIGN=TOP>");
    for ($column_num = 0;
         $column_num < $column_count;
         $column_num++)
      {
        print("<TD>$row[$column_num]</TD>\n");
      }
    print("</TR>\n");
  }
  print("</TABLE>\n");   }

function display_db_table($tablename, $connection,
                          $header_bool, $table_params)
{
  $query_string = "select * from $tablename";
  display_db_query($query_string, $connection,
                   $header_bool, $table_params);
}
?>

<HTML><HEAD><TITLE>Countries and cities</TITLE></HEAD>
<BODY>
<TABLE><TR><TD>
<?php display_db_table("country", $global_dbh,
                       TRUE, "BORDER=2"); ?>
</TD><TD>
<?php display_db_table("city", $global_dbh,
                       TRUE, "BORDER=2"); ?>
</TD></TR></TABLE></BODY></HTML>
```

Exemplo 1.8 – Anexo III

Construindo Formulários a partir de Consultas

Listamos alguns pontos para lembrar do que é preciso fazer para que bons formulários sejam tratados pelo PHP e um banco de dados:

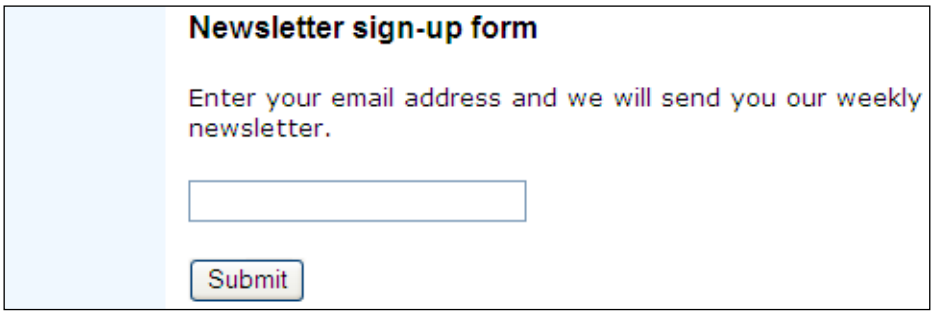
- Sempre utilize um atributo NAME para cada elemento de entrada de dados (INPUT, SELECT, TEXTAREA, etc.). Esses atributos NAME se tornarão nomes de variável no PHP.

- Um formulário NAME não precisa ser o mesmo que o nome de campo de banco de dados correspondente, mas frequentemente essa é uma boa ideia.
- É possível especificar um VALUE em vez de permitir que o PHP envie o valor padrão. Considere substituir um valor numérico por um valor de texto se possível, porque o banco de dados é muito mais lento nas *strings* correspondentes do que os inteiros.
- O VALUE pode ser configurado como os dados que você deseja exibir no formulário.
- Lembre-se de que ao passar variáveis ocultas de um formulário para outro (ou para uma página) utilizando os elementos HIDDEN de entrada de dados. Essa prática pode ter implicações negativas de segurança, mas não é frequentemente pior do que armazenar os dados em um cookie.

Envio de Formulário Básico para um Banco de Dados

Enviar dados para um banco de dados via formulário de HTML é simples e direto se o formulário e o *handler* de formulário forem duas páginas separadas.

Abaixo temos um exemplo:




```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
BODY, P      {color: black; font-family: verdana; font-size: 10 pt}
H1        {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>

<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=17%>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Newsletter sign-up form</H1>
<P>Enter your email address and we will send you our weekly
newsletter.</P>
<FORM METHOD="post" ACTION="formhandler.php">
<INPUT TYPE="text" SIZE=25 NAME="email">
<BR><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Submit">
</FORM>
</TD>
</TR>
```

```
</TABLE>

</BODY>
</HTML>
```

Exemplo 1.9 – Anexo III

Inserimos os dados no banco de dados e reconhecemos o recibo no *handler* de formulário na listagem abaixo:

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
BODY, P      {color: black; font-family: verdana; font-size: 10 pt}
H1        {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>

<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=17%>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Newsletter sign-up form</H1>
<?php

if (!$_POST['email'] || $_POST['email'] == "" || strlen($_POST['email']) > 30)
{
  echo '<P>There is a problem.  Did you enter an email address?</P>';
} else {
  // Open connection to the database
  mysql_connect("localhost", "phpuser", "sesame") or die("Failure to
communicate with database");
  mysql_select_db("test");

  // Insert email address
  $as_email = addslashes($_POST['email']);
  $tr_email = trim($as_email);
  $query = "INSERT INTO mailinglist (ID, Email, Source)
            VALUES(NULL,'$tr_email', 'www.example.com/newsletter_signup.html')
           ";
  $result = mysql_query($query);
  if (mysql_affected_rows() == 1) {
    echo '<P>Your information has been recorded.</P>';
  } else {
    error_log(mysql_error());
    echo '<P>Something went wrong with your signup attempt.</P>';
  }
}
?>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Exemplo 1.10 – Anexo III

Auto-envio

O auto-envio refere-se ao processo de combinar um ou mais formulários em um único script, utilizando HTML FORM padrão para enviar dados para o script uma ou mais vezes. O auto-envio é realizado pelo mais simples dos meios: especificando o mesmo nome de script que o alvo ACTION no elemento FORM assim:

```
<FORM METHOD=”POST” ACTION=”meu.php”>
```

Ou utilizando um recurso predefinido único no PHP:

```
<FORM METHOD=”POST” ACTION=”<?php echo $_SERVER[‘PHP_SELF’]; ?>”>
```

No exemplo abaixo, unimos a lógica e o HTML no mesmo arquivo. O botão SUBMIT será usado para saber quando a consulta ao banco de dados deverá ser realizada. Isso porque a lógica aparece antes do HTML.

```
<?php

if ($_POST['submit'] == 'Submit') {
  if     (!$_POST['email']     ||     $_POST['email']     ==     ""     ||
strlen($_POST['email'] > 30)) {
    $message  =  '<P>There  is  a  problem.    Did  you  enter  an  email
address?</P>';
  } else {
    // Open connection to the database
    mysql_connect("localhost", "phpuser", "sesame") or die("Failure to
communicate with database");
    mysql_select_db("test");

    // Insert email address
    $as_email = addslashes($_POST['email']);
    $tr_email = trim($as_email);
    $query = "INSERT INTO mailinglist (ID, Email, Source)
              VALUES(NULL,                                    '$tr_email',
'www.example.com/newsletter_signup.html')
             ";
    $result = mysql_query($query);
    if (mysql_affected_rows() == 1) {
      $message = '<P>Your information has been recorded.</P>';

      $noform_var = 1;
    } else {
      error_log(mysql_error());
      $message   =   '<P>Something   went   wrong   with   your   signup
attempt.</P>';
    }
  }

  // Show the form in every case except successful submission
  if (!$noform_var) {
    $thisfile = $_SERVER['PHP_SELF'];
    $message .= <<< EOMSG
<P>Enter  your  email  address  and  we  will  send  you  our  weekly
newsletter.</P>
```

```
<FORM METHOD="post" ACTION="$thisfile">
<INPUT TYPE="text" SIZE=25 NAME="email">
<BR><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Submit">
</FORM>
EOMSG;
  }
}
?>

<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
BODY, P      {color: black; font-family: verdana; font-size: 10 pt}
H1        {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>

<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=17%>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Newsletter sign-up form</H1>
<?php echo $message; ?>
</TD>
</TR>
</TABLE>

</BODY>
</HTML>
```

Exemplo 1.11 – Anexo III

A primeira vez que carregar a página, deverá aparecer um formulário de HTML normal exatamente como o da listagem anterior. Se enviá-lo sem dados ou com uma *string* muito longa (frequentemente um sinal de tentativa de invasão), aparecerá uma mensagem de erro e o formulário novamente. Se algo sair errado com o banco de dados, aparecerá uma mensagem de erro e o formulário aparecerá. Semente se INSERT completar com sucesso, não aparecerá o formulário, que é a navegação que desejamos porque não queremos que as pessoas se inscrevam novamente.

No exemplo acima, só precisamos verificar dois estados do formulário (não enviado ou enviado), então podemos utilizar o botão SUBMIT como nossa variável de estado. Mas se precisássemos verificar mais de um estado? Nesse caso precisaríamos de uma variável que seja capaz de aceitar mais de um valor. Poderíamos dar valores diferentes ao botão SUBMIT, os quais apareceriam como diferentes rótulos no próprio botão; ou poderia configurar uma variável oculta que fosse capaz de aceitar mais de um valor dependendo do estado. No próximo exemplo é demonstrada essa técnica, que coleta algumas informações e então permite avaliar seu patrão anonimamente.

```
<?php
```

```
// First set the form strings, which will be displayed
//in various cases below
$thisfile = $_SERVER['PHP_SELF']; //Have to set this for heredoc

$reg_form = <<< EOREGFORM
<P>We must ask for your name and email address to ensure that no one
votes more than once, but we do not associate your personal information
with your rating.</P>
<FORM METHOD="post" ACTION="$thisfile">
Name: <INPUT TYPE="text" SIZE=25 NAME="name"><BR><BR>
Email: <INPUT TYPE="text" SIZE=25 NAME="email">
<INPUT TYPE="hidden" NAME="stage" VALUE="register">
<BR><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Submit">
</FORM>
EOREGFORM;

$rate_form = <<< EORATEFORM
<P>My boss is:</P>
<FORM METHOD="post" ACTION="$thisfile">
<INPUT TYPE="radio" NAME="rating" VALUE=1> Driving me to look for a new
job.<BR>
<INPUT TYPE="radio" NAME="rating" VALUE=2> Not the worst, but pretty
bad.<BR>
<INPUT TYPE="radio" NAME="rating" VALUE=3> Just so-so.<BR>
<INPUT TYPE="radio" NAME="rating" VALUE=4> Pretty good.<BR>
<INPUT TYPE="radio" NAME="rating" VALUE=5> A pleasure to work
with.<BR><BR>
Boss's name: <INPUT TYPE="text" SIZE=25 NAME="boss"><BR>
<INPUT TYPE="hidden" NAME="stage" VALUE="rate">
<BR><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Submit">
</FORM>
EORATEFORM;

if (!$_POST['submit']) {
  // First time, just show the registration form
  $message = $reg_form;

} elseif ($_POST['submit'] == 'Submit' && $_POST['stage'] == 'register')
{
  // Second time, show the registration form again on error,
  // rating form on successful INSERT

  if (!$_POST['name'] || $_POST['name'] == "" || strlen($_POST['name'] >
30)    ||    !$_POST['email']    ||    $_POST['email']    ==    ""    ||
strlen($_POST['email'] > 30)) {
    $message = '<P>There is a problem.  Did you enter a name and email
address?</P>';
    $message .= $reg_form;
  } else {
    // Open connection to the database
    mysql_connect("localhost", "phpuser", "sesame") or die("Failure to
communicate with database");
    mysql_select_db("test");
```

```
    // Check to see this name and email have not appeared before
    $as_name = addslashes($_POST['name']);
    $tr_name = trim($as_name);
    $as_email = addslashes($_POST['email']);
    $tr_email = trim($as_email);
    $query = "SELECT sub_id FROM raters
              WHERE Name = '$tr_name'
              AND Email = '$tr_email'
             ";
    $result = mysql_query($query);
    if (mysql_num_rows($result) > 0) {
      error_log(mysql_error());
      $message = 'Someone with this name and password has already rated
.  If you think a mistake was made, please email help@example.com.';
    } else {
      // Insert name and email address
      $query = "INSERT INTO raters (ID, Name, Email)
                VALUES(NULL, '$tr_name', '$tr_email')
               ";
      $result = mysql_query($query);
      if (mysql_affected_rows() == 1) {
        $message = $rate_form;
      } else {
        error_log(mysql_error());
        $message   =   '<P>Something   went   wrong   with   your   signup
attempt.</P>';
        $message .= $reg_form;
      }
    }
  }

}  elseif ($_POST['submit'] == 'Submit' && $_POST['stage'] == 'rate') {
  // Third time, store the rating and boss's name

  // Open connection to the database
  mysql_connect("localhost",  "phpuser",  "sesame")  or  die("Failure  to
communicate with database");
  mysql_select_db("test");

  // Insert rating and boss's name
  $as_boss = addslashes($_POST['boss']);
  $tr_boss = trim($as_boss);
  $rating = $_POST['rating'];
  $query = "INSERT INTO ratings (ID, Rating, Boss)
            VALUES(NULL, '$rating', '$tr_boss')
           ";
  $result = mysql_query($query);
  if (mysql_affected_rows() == 1) {
    $message = '<P>Your rating has been submitted.</P>';
  } else {
    error_log(mysql_error());
    $message = '<P>Something went wrong with your rating attempt.  Try
again.</P>';
    $message .= $rate_form;
  }
}
```

```
?>

<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
BODY, P      {color: black; font-family: verdana; font-size: 10 pt}
H1        {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>

<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=17%>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Rate your boss anonymously</H1>
<?php echo $message; ?>
</TD>
</TR>
</TABLE>

</BODY>
</HTML>
```

Exemplo 1.11 – Anexo III

Funções de MySQL

A tabela abaixo reúne um resumo das principais funções de MySQL:

Nome da Função	Uso
`mysql_affected_rows( [ id _do_ link] )`	Utilize depois de uma cosua INSERT, UPDATE ou DELETE não zero para veificar o nu´mero de linhas alteradas.
`mysql_change_usuário (usuário, senha [,banco_de_dados] [, id_do_link] )`	Alterna o usuário do MySQL para um link aberto.
`mysql_close( [ id _do_link] )`	Fecha o link indentificado (normalmente desnecessário).
`mysql_connect( [host] [:port] [:soquete] [, nome_do_usuário] [, senha] )`	Abre um link no host, porta, soquete especificado; como o usuário especificado com a senha. Todos os argumentos são opcionais.
`mysql_create_db (nome_de_db [, id_do_link] )`	Cria um novo banco de dados MySQL no host associado com o link aberto mais próximo.
`mysql_data_seek( id_do_resutado, linha_num)`	Move ponteiro interno de linha para número especificado de linha. Utiliza uma função de busca para retornar os dados dessa linha.
`mysql_drop_db (nome_de_db [, id_do_link] )`	Exclui ("dropa") o banco de dados MySQL especificado.
`mysql_errno ( [ id_do_link] )`	Retorna o ID de Erro.
`mysql_error ( [id_do_link])`	Retorna uma mensagem de erro de texto.

mysql_fetch_array (id_do_resultado[, tipo_de_resultado])	Busca conjunto de resultados como array associativo. O tipo de resultado pode ser MYSQL_ASSOC,MYSQL_NUM ou MYSQL_BOTH (padrão).
mysql_fetch_field (id_resultado[, deslocamento_de_campo])	Retorna informações sobre um campo como um objetivo.
mysql_fetch_lengths (result_id)	Retorna o comprimento de cada campo em conjunto de resultados.
mysql_fetch_object (id_do _resultado[, tipo_de_resultado])	Busca o conjunto de resultados como um objeto. Veja mysql_fetch_array para o tipo de resultado.
mysql_fetch_row (id_do_resultado)	Busca (fetch) o conjunto de resultados como um array enumerado.
mysql_field_name (id_do_resultado, indice_de_ campo)	Retorna o nome de campo enumerado.
mysql_field_seek (resultado_de_id, deslocamento_de_campo)	Move o ponteiro de resultado para o deslocamento de campo especificado. Utilizando com mysql_fetch_field.
mysql_field_table (id_do_resultado, deslocamento_de_campo)	Retorna o nome da tabela do campo especificado.
mysql_field_type (resultado_id, deslocamento_de_campo)	Retorna o tipo de deslocamento de campo (por exemplo, TINYINT, BLOB, VARCHAR).
mysql_field_flags (id_do_resultado, deslocamento_de_campo)	Retorna flags associados com os campos enumerados (por exemplo, NOT NULL, AUTO_INCREMENT, BINARY).
mysql_field_len (id_do_resultado, deslocamento_de_campo)	Retorna o comprimento do campo enumerado.
mysql_free_result (id_do_resultado)	Libera memória usada pelo conjunto de resultados (normalmente desnecessário).
mysql_insert_id ([id_do_link])	Retorna o ID de AUTO_INCREMENTED de INSERT; ou FALSE se a inserção falhou ou se a ultima consulta não era inserção.
mysql_list_fields (banco_de_dados, tabela [,id_do_link])	Retorna ID do resultado para utilização nas funções mysql_field, sem realizar uma consulta real.
mysql_list_dbs ([id_do_link])	Retorna o ponteiro de resultados do banco de dados no mysqld. Utilizado com mysql_tablename.
mysql_list_tables (banco_de_dados, tabela [,id_do_link])	Retorna o ponteiro de resultados nas tabelas de banco de dados. Utilizado com mysql_tablename.
mysql_num_fields (id_do_resultado)	Retorna o numero de campos em um conjunto de resultados.
mysql_num_rows (id_do_resultado)	Retorna o numero de linhas em um conjunto de resultados.
mysql_proconnect ([host] [:porta] [:soquete] [, nome_do_usuario] [,password])	Abre a conexão persistente com o banco de dados. Todos os argumentos são opcionais. Tenha cuidado - mysql_close e o término de script não fecharão a conexão.
mysql_query (string_de_consulta [,id_do_link])	Envia uma consulta para o banco de dados. Lembre-se de colocar o ponto-e-vírgula fora da string de consulta entre aspas duplas.
mysql_result (resultado_de_id, id_do_coluna indentificador_de_campo)	Retorna resultado de campo único. O identificador de campo pode ser deslocamento de campo (0), nome de campo (FirstName) ou nome de tabela de ponto (myfield.mytable).
mysql_select_db (banco_de_dados [,id_do_link])	Seleciona o banco de dados para consultas.
mysql_telename (id_do_resultado,	Utilizado com qualquer função mysql_list para

`id_da_tabela)`	retornar o valor referenciado por um ponteiro de resultado.

Bibliografia

Sites:

Site Oficial (www.php.net)

Mirror brasileiro (br.php.net)

W3Schools (www.w3schools.com)

Tutoriais (pt-br.html.net/tutorials/)

PHP Orientado a Objetos (php.net/manual/en/language.oop5.php)

SQL Server Driver for PHP Documentation (msdn.microsoft.com/en-us/library/ff928321%28v=SQL.10%29.aspx

technet.microsoft.com/en-us/library/dn425064(v=sql.10).aspx)

SQL Server Driver for PHP (msdn.microsoft.com/en-us/sqlserver/cc299381.aspx)

Como habilitar conexões criptografadas no Mecanismo de Banco de Dados (SQL Server Configuration Manager): msdn.microsoft.com/pt-br/library/ms191192.aspx

SQL Injection: msdn.microsoft.com/pt-br/library/ms161953.aspx

Expressão Regular: aurelio.net/regex/guia/

Livros:

Ajax, Rich Internet Applications e desenvolvimento Web para programadores
Deitel, Paul J. e Deitel, Harvey M.
Pearson Prentice Hall

Construindo Aplicações Web com PHP e MySQL
Milani, André
Editora Novatec

PHP 5 e MySQL– a Bíblia
Tim Converse, Joyce Park e Clark Morgan
Editora Campus

Professional Php Programming
Castagnetto, Jesus
Wrox Press

PHP Programando com Orientação à Objetos
Dall'Oglio, Pablo
Novatec