

**Edição revisada e atualizada • Inclui versão 5**

# **Desenvolvendo Websites com PHP**

**Aprenda a criar Websites dinâmicos e  
interativos com PHP e bancos de dados**

**Juliano Niederauer**

**novatec**



# **Desenvolvendo Websites com PHP**

**Aprenda a criar Websites dinâmicos e  
interativos com PHP e bancos de dados**



# **Desenvolvendo Websites com PHP**

**Aprenda a criar Websites dinâmicos e  
interativos com PHP e bancos de dados**

**Juliano Niederauer**

Novatec Editora

Copyright © 2004 Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998.  
É proibida a reprodução desta obra, mesmo parcial, por qualquer processo,  
sem prévia autorização, por escrito, do autor e da Editora.

Editor: RUBENS PRATES

Capa: CAMILA MESQUITA

ISBN: 85-7522-050-0

**Histórico de impressões**

|               |                      |
|---------------|----------------------|
| Março/2004    | Primeira impressão   |
| Dezembro/2004 | Primeira reimpressão |
| Dezembro/2005 | Segunda reimpressão  |
| Setembro/2006 | Terceira reimpressão |
| Setembro/2007 | Quarta reimpressão   |
| Maio/2008     | Quinta reimpressão   |
| Março/2009    | Sexta reimpressão    |
| Março/2010    | Sétima reimpressão   |

NOVATEC EDITORA LTDA.  
Rua Luís Antônio dos Santos 110  
02460-000 São Paulo SP – Brasil  
Tel.: +55 11 2959-6529  
Fax: +55 11 2950-8869  
E-mail: [novatec@novatec.com.br](mailto:novatec@novatec.com.br)  
Site: [www.novatec.com.br](http://www.novatec.com.br)

Há várias maneiras de ser entendido: ser claro é uma delas.  
*(autor desconhecido)*



*Dedico este livro a todos que lutam por um Brasil alfabetizado, pelo aumento do número de jovens concluindo a educação básica, pela formação e valorização dos professores e principalmente pela abolição das desigualdades de renda, de classe, de gênero, de região, de raça e de deficiência física no acesso à educação.*

# Sobre o autor

Juliano Niederauer ([www.niederauer.com.br](http://www.niederauer.com.br)) é graduado em Ciência da Computação pela PUCRS e pós-graduado (MBA) em Gestão Empresarial, com concentração em Tecnologia Aplicada a Negócios. É Diretor da Virtuous Tecnologia da Informação ([www.virtuous.com.br](http://www.virtuous.com.br)), empresa que presta serviços na área de TI. Possui larga experiência no desenvolvimento de aplicações para a Web, tendo ministrado diversos cursos de PHP em universidades e empresas. Já criou sites de sucesso na Web, sendo o principal deles o maior portal matemático do mundo, o Só Matemática ([www.somatematica.com.br](http://www.somatematica.com.br)), que hoje conta com mais de 3.000 páginas de Matemática e mais de 5 milhões de page-views mensais. Possui conhecimentos profundos sobre as linguagens PHP, ASP e sobre o desenvolvimento de programas em ambiente CGI utilizando a linguagem PERL. Possui ainda conhecimentos sobre bancos de dados relacionais, incluindo o MySQL, PostgreSQL, InterBase e Oracle. Na área de dispositivos móveis domina diversas ferramentas de desenvolvimento para PalmOS, como CodeWarrior e NSBasic. Pode ser contatado pelo e-mail [juliano@niederauer.com.br](mailto:juliano@niederauer.com.br).

# Sumário

|   |           |
|---|-----------|
| Sobre o autor .....                                   | 8         |
| Introdução .....                                      | 15        |
| <b>Capítulo 1 O que é o PHP? .....</b>                | <b>19</b> |
| Características do PHP .....                          | 20        |
| Gratuito e com código aberto .....                    | 20        |
| Embutido no HTML .....                                | 20        |
| Baseado no servidor .....                             | 21        |
| Bancos de dados .....                                 | 22        |
| Portabilidade .....                                   | 22        |
| <b>Capítulo 2 Instalação do PHP .....</b>             | <b>23</b> |
| Utilizando um provedor de hospedagem .....            | 23        |
| Instalando a partir de uma distribuição Linux .....   | 23        |
| Instalando manualmente no Linux ou Windows .....      | 24        |
| <b>Capítulo 3 Noções básicas de programação .....</b> | <b>25</b> |
| Começando a programar .....                           | 25        |
| Estrutura de um programa PHP .....                    | 27        |
| Código PHP e comandos HTML .....                      | 27        |
| Exibindo a página no browser .....                    | 28        |
| <b>Capítulo 4 Manipulando os dados em PHP .....</b>   | <b>31</b> |
| Dados numéricos .....                                 | 31        |
| Dados alfanuméricos (textos) .....                    | 32        |
| Aspas simples (') .....                               | 32        |
| Aspas duplas ("") .....                               | 34        |
| Aspas invertidas (`) .....                            | 35        |
| Constantes .....                                      | 36        |
| Variáveis em PHP .....                                | 37        |
| Maiúsculas e minúsculas (case-sensitive) .....        | 38        |
| Escopo das variáveis .....                            | 38        |
| Conversão de variáveis .....                          | 40        |
| Interpolação de variáveis .....                       | 41        |
| Variáveis criadas durante a execução .....            | 43        |
| Tipos das variáveis .....                             | 44        |
| Operadores .....                                      | 47        |
| Operadores aritméticos .....                          | 48        |
| Operadores binários .....                             | 50        |

|   |            |
|---|------------|
| Operadores de comparação .....                        | 52         |
| Operadores de atribuição .....                        | 52         |
| Operadores lógicos .....                              | 54         |
| Operador ternário .....                               | 56         |
| Precedência de operadores .....                       | 56         |
| <b>Capítulo 5 Estruturas de controle em PHP .....</b> | <b>59</b>  |
| Comandos condicionais .....                           | 59         |
| if .....  | 60         |
| switch .....  | 63         |
| Comandos de repetição .....                           | 65         |
| while .....   | 65         |
| do...while .....                                      | 67         |
| for .....   | 68         |
| foreach .....   | 71         |
| Controlando o fluxo de execução .....                 | 72         |
| break .....   | 72         |
| continue .....  | 74         |
| <b>Capítulo 6 Funções e classes .....</b>             | <b>75</b>  |
| Definição de função .....                             | 75         |
| Como criar uma função .....                           | 76         |
| Utilizando o comando return em uma função .....       | 77         |
| Utilizando funções para verificar um CPF .....        | 80         |
| Passagem de parâmetros: valor e referência .....      | 82         |
| Funções recursivas .....                              | 85         |
| Reutilizando funções .....                            | 87         |
| Definição de classe .....                             | 87         |
| Como criar uma classe .....                           | 88         |
| Programação orientada a objetos no PHP 5 .....        | 90         |
| As palavras-chave private e protected .....           | 90         |
| Métodos abstratos e interfaces .....                  | 92         |
| A palavra-chave final .....                           | 93         |
| Construtores e destrutores .....                      | 93         |
| Variáveis e métodos estáticos .....                   | 95         |
| <b>Capítulo 7 Utilizando includes em PHP .....</b>    | <b>97</b>  |
| Criando um menu para seu site .....                   | 97         |
| Exibindo a data atual com uma include .....           | 100        |
| Reutilização de código .....                          | 100        |
| Include x Require .....                               | 101        |
| <b>Capítulo 8 PHP e formulários HTML .....</b>        | <b>103</b> |
| Como criar um formulário .....                        | 103        |
| Enviando as informações para um programa PHP .....    | 107        |
| Método GET .....                                      | 107        |
| Método POST .....                                     | 108        |

## Sumário

|                    |   |            |
|--------------------|---|------------|
|                    | Como tratar as informações recebidas .....              | 109        |
|                    | Funções especiais para formatação de dados .....        | 110        |
|                    | Verificando os campos de um formulário .....            | 112        |
| <b>Capítulo 9</b>  | <b>Passando informações por várias páginas .....</b>    | <b>115</b> |
|                    | Utilizando o campo hidden dos formulários .....         | 116        |
|                    | Passando informações pela URL .....                     | 118        |
|                    | Dividindo o cadastramento de usuários em etapas .....   | 119        |
| <b>Capítulo 10</b> | <b>PHP e variáveis de ambiente .....</b>                | <b>125</b> |
|                    | Utilizando a função getenv .....                        | 125        |
|                    | Descobrindo o endereço IP do visitante .....            | 126        |
|                    | Lista das variáveis de ambiente .....                   | 127        |
| <b>Capítulo 11</b> | <b>Banco de dados: MySQL ou PostgreSQL .....</b>        | <b>129</b> |
|                    | Comparação entre MySQL e PostgreSQL .....               | 129        |
|                    | Outra alternativa: SQLite .....                         | 130        |
|                    | Como criar um banco de dados .....                      | 130        |
|                    | PostgreSQL .....  | 131        |
|                    | MySQL .....   | 132        |
|                    | Tipos de dados aceitos pelo MySQL e PostgreSQL .....    | 132        |
|                    | PostgreSQL .....  | 133        |
|                    | MySQL .....   | 134        |
|                    | Como criar tabelas em um banco de dados .....           | 135        |
|                    | Visualizando com o mysql .....                          | 137        |
|                    | Visualizando com o psql .....                           | 138        |
|                    | Inserindo informações em um banco de dados .....        | 139        |
|                    | Comando INSERT .....                                    | 139        |
|                    | Alterando um banco de dados .....                       | 141        |
|                    | Comando UPDATE .....                                    | 141        |
|                    | Comando ALTER TABLE .....                               | 142        |
|                    | Excluindo informações de um banco de dados .....        | 142        |
|                    | Comando DELETE .....                                    | 142        |
|                    | Comando DROP TABLE .....                                | 143        |
|                    | Fazendo consultas em um banco de dados .....            | 143        |
|                    | Comando SELECT .....                                    | 143        |
|                    | Ordenando os resultados de uma consulta .....           | 153        |
|                    | Determinando o número de linhas retornadas .....        | 154        |
|                    | Gravando os resultados em uma nova tabela .....         | 155        |
|                    | Utilizando INSERT e SELECT para inserir registros ..... | 156        |
|                    | Criando e utilizando seqüências .....                   | 156        |
| <b>Capítulo 12</b> | <b>PHP com banco de dados .....</b>                     | <b>159</b> |
|                    | Conectando com um banco de dados .....                  | 159        |
|                    | MySQL .....   | 159        |
|                    | PostgreSQL .....  | 161        |

|  |            |
|--|------------|
| Executando comandos SQL em um programa PHP .....         | 162        |
| MySQL .....  | 162        |
| PostgreSQL .....   | 163        |
| Exibindo os resultados de comandos SQL.....              | 163        |
| Gerenciando um banco de dados com PHP .....              | 168        |
| Exemplo utilizando a biblioteca SQLite .....             | 172        |
| <b>Capítulo 13 Cookies e sessões .....</b>               | <b>179</b> |
| Algumas utilidades de cookies e sessões .....            | 179        |
| Utilizando cookies .....                                 | 180        |
| Enviando cookies pelo PHP .....                          | 180        |
| O array superglobal \$_COOKIE .....                      | 181        |
| Criando um sistema de username/senha para seu site ..... | 182        |
| Utilizando sessões .....                                 | 189        |
| Criando uma sessão no PHP .....                          | 190        |
| Registrando variáveis em uma sessão .....                | 190        |
| Parâmetros de configuração .....                         | 192        |
| Usando sessões no sistema de username/senha .....        | 194        |
| <b>Capítulo 14 Manipulando arquivos em PHP .....</b>     | <b>197</b> |
| Quando utilizar arquivos no PHP .....                    | 197        |
| Funções para manipulação de arquivos .....               | 198        |
| Exemplo: contador de acessos .....                       | 202        |
| Outras funções para o sistema de arquivos .....          | 203        |
| <b>Capítulo 15 Enviando e-mails com o PHP .....</b>      | <b>213</b> |
| Por que enviar e-mails com o PHP? .....                  | 213        |
| Utilizando a função mail .....                           | 214        |
| Configurações no arquivo php.ini .....                   | 216        |
| Adicionando informações ao cabeçalho do e-mail .....     | 216        |
| Enviando e-mails em formato HTML .....                   | 217        |
| Lista dos cabeçalhos de e-mail (mail headers) .....      | 219        |
| <b>Apêndice A Comandos gerais do PHP .....</b>           | <b>221</b> |
| Arrays .....   | 221        |
| Classes e objetos .....                                  | 223        |
| Data e hora .....  | 223        |
| Diretórios .....   | 224        |
| FTP .....  | 224        |
| Funções .....  | 225        |
| HTTP .....   | 226        |
| Imagens .....  | 226        |
| Matemática .....   | 229        |
| Opções e informações do PHP .....                        | 231        |
| PDF .....  | 232        |
| Sessões .....  | 235        |
| Sistema de arquivos (Filesystem) .....                   | 236        |

|   |            |
|---|------------|
| <b>Sumário</b>                                    |            |
| Strings   | 238        |
| URL   | 240        |
| Variáveis   | 241        |
| <b>Apêndice B     Funções PHP/bancos de dados</b> | <b>243</b> |
| MySQL   | 243        |
| MySQLi  | 245        |
| PostgreSQL  | 248        |
| SQLite  | 250        |
| InterBase/Firebird                                | 252        |
| dbx   | 253        |
| Microsoft SQL Server                              | 254        |
| Oracle  | 255        |
| OCI8  | 255        |
| ODBC  | 257        |
| <b>Apêndice C     Tipos de recursos do PHP</b>    | <b>259</b> |
| <b>Apêndice D     Links interessantes</b>         | <b>263</b> |
| <b>Índice remissivo</b>                           | <b>265</b> |



# Introdução

O PHP é uma linguagem de programação que nasceu para gerar conteúdo dinâmico na Web. Ele é amplamente usado para criar sites interativos e aplicativos web. O PHP é uma linguagem de script interpretada, que pode ser executada diretamente no lado do servidor, o que significa que não precisa ser compilada antes de ser executada.

A primeira versão do PHP surgiu em 1995, quando Rasmus Lerdorf criou para uso pessoal uma ferramenta chamada PHP/PI (*Personal Home Page/Forms Interpreter*). Porém, ele não imaginava que estava criando uma das mais poderosas linguagens para o desenvolvimento de aplicações na Web. O PHP (sigla que hoje é um acrônimo recursivo para *PHP: Hypertext Preprocessor*) conquistou muito espaço nos últimos anos. Isso se deve principalmente à facilidade de utilização e grande diversidade de recursos que possui.

O PHP é uma linguagem totalmente voltada à Internet, possibilitando o desenvolvimento de sites realmente dinâmicos. Dominando essa linguagem, pode-se transformar aqueles sites estáticos, feitos de HTML puro, em sites interativos, utilizando todas as técnicas de programação que essa linguagem oferece.

Com este livro você vai aprender os conceitos de programação, como criar formulários HTML e tratar as informações enviadas por eles, como criar seu banco de dados e executar comandos SQL para acessá-lo mediante seus programas em PHP, como criar um banco de dados de usuários e autenticá-los em seu site, por um sistema de *username* e *password* (nome de usuário e senha), e muito mais.

O livro apresenta ainda uma explicação detalhada sobre a linguagem PHP e todas as suas características, além de noções de programação, sintaxe e estruturação de programas, manipulação de dados, entre outros assuntos. Não importa se você é iniciante ou se já é um usuário mais experiente. Esta obra será muito útil para ajudar você a dominar e criar excelentes sites com o PHP.

Durante a leitura podem-se encontrar diversas dicas do autor, e também programas-exemplo desenvolvidos por ele para facilitar o aprendizado da linguagem.

Todos os exemplos de programas apresentados neste livro estão disponíveis para download no site da Novatec Editora, no endereço <http://www.novateceditora.com.br/download/php>.

## Quem deve ler este livro

Este livro destina-se a pessoas que querem aprender a desenvolver páginas dinâmicas na Web utilizando o PHP. Como pré-requisito para poder compreender os conteúdos apresentados, é importante que o leitor tenha um conhecimento razoável sobre HTML, a linguagem padrão para a criação de páginas na Internet.

Esse conhecimento é importante porque o PHP funciona em conjunto com a HTML, isto é, o código PHP é embutido no meio de uma página HTML. Portanto, se você não tem nenhum conhecimento sobre essa linguagem, é recomendável a leitura de algum material (apostila, livro, site etc.) sobre o assunto. Isso não deve lhe tomar muito tempo, visto que a HTML é uma linguagem de marcação bastante simples.

Os assuntos abordados neste livro são úteis tanto para usuários iniciantes como para os mais experientes, pois abrangem desde noções básicas de programação até conteúdos mais avançados como a criação e manutenção de bancos de dados, manipulação de cookies, gerenciamento de sessões, operações sobre arquivos, envio de e-mails, entre outros.

A metodologia de ensino utilizada aqui será a mesma das demais obras do autor, onde a didática é o ponto forte. A abordagem dos temas é clara, objetiva e as explicações são feitas passo a passo para facilitar o aprendizado.

## Como este livro está organizado

O livro é composto por quinze capítulos e por quatro apêndices. É recomendável ler os capítulos de forma seqüencial, visto que muitos deles requerem o conhecimento de assuntos apresentados em capítulos anteriores. A seguir é apresentada uma rápida descrição dos conteúdos abordados.

**Capítulo 1 - O que é o PHP?**: explica o que é PHP, descreve as características da linguagem e apresenta situações nas quais é interessante o seu uso.

**Capítulo 2 - Instalação do PHP**: explica como obter a versão mais atual da linguagem e quais são os softwares necessários para colocá-la em funcionamento.

**Capítulo 3 - Noções básicas de programação**: mostra como você pode começar a programar, além de apresentar a estrutura básica de um programa PHP e como embutir código PHP em uma página HTML.

Impresso Padrão da Edição, 2001  
Agradecimentos ao Prof. Dr. Wilson Góes

**Capítulo 4 – Manipulando os dados em PHP:** apresenta os tipos de dados existentes no PHP, além de ensinar a utilizar constantes e variáveis. Mostra ainda os operadores que podem ser utilizados no PHP e a precedência entre eles.

**Capítulo 5 – Estruturas de controle em PHP:** ensina a utilizar os comandos condicionais (*if* e *switch*) e os comandos de repetição (*while*, *do...while*, *for* e *foreach*). Mostra também como controlar o fluxo de execução dos programas com as instruções *break* e *continue*.

**Capítulo 6 – Funções e classes:** ensina a criar funções no PHP, apresentando a sintaxe básica e mostrando como retornar valores com o comando *return*. Contém exemplos envolvendo validação de CPF e funções recursivas, além de ensinar a criar uma classe. Ao final, apresenta algumas características do novo modelo de objetos da versão 5 do PHP.

**Capítulo 7 – Utilizando includes em PHP:** explica a utilidade das *includes* e mostra como criar um menu para o seu site utilizando esse recurso. Apresenta ainda um exemplo envolvendo a exibição da data atual e explica a diferença entre os comandos *include* e *require*.

**Capítulo 8 – PHP e formulários HTML:** mostra como você pode tornar seu site mais interativo com a criação de formulários HTML, através dos quais o usuário poderá enviar informações para os seus programas PHP. Ensina a tratar os dados recebidos por meio de formulários, além de aplicar funções especiais para formatação desses dados.

**Capítulo 9 – Passando informações por várias páginas:** explica como pode ser feita a passagem de informações entre diversas páginas PHP, tanto através do campo *hidden* (escondido) dos formulários HTML, como também pelas URLs. Utilizando esses conceitos, apresenta um exemplo de divisão do cadastramento de usuários em várias etapas.

**Capítulo 10 – PHP e variáveis de ambiente:** explica o que são variáveis de ambiente e mostra como obter o valor dessas variáveis dentro de um programa PHP. Contém ainda a lista das principais variáveis de ambiente.

**Capítulo 11 – Banco de dados: MySQL ou PostgreSQL:** ensina a criar bancos de dados e tabelas nos SGBDs (Sistemas de Gerência de Bancos de Dados) MySQL e PostgreSQL, além de apresentar os comandos SQL para realizar operações (consultas, inclusões, alterações e exclusões) em cada um deles.

**Capítulo 12 – PHP com banco de dados:** ensina a utilizar o PHP para conectar-se a um banco de dados e executar comandos SQL sobre ele. Mostra também como utilizar as funções do MySQL e PostgreSQL para obter os dados resultantes de uma consulta e dar a eles um tratamento adequado.

**Capítulo 13 – Cookies e sessões:** ensina a utilizar esses recursos para manter informações através de acessos subseqüentes. Mostra como realizar a autenticação de usuários e ensina a criar um sistema de *username/password* (usuário/senha) para o seu site.

**Capítulo 14 – Manipulando arquivos em PHP:** explica em que situações é recomendável o uso de arquivos, além de apresentar as funções do PHP responsáveis pelas operações mais comuns (abertura, fechamento, leitura e escrita) sobre eles.

**Capítulo 15 – Enviando e-mails com o PHP:** explica a utilidade do envio de e-mails através de uma página Web e apresenta a função mail, responsável por executar essa tarefa no PHP. Mostra ainda como adicionar informações ao cabeçalho do e-mail e apresenta a lista dos principais cabeçalhos (mail headers) existentes.

**Apêndice A – Comandos gerais do PHP:** apresenta a descrição dos principais comandos do PHP. Pode ser utilizado como referência. Os comandos foram divididos em diversas categorias, como arrays, strings, variáveis, funções, classes e objetos, matemática, data e hora, sistema de arquivos (filesystem), diretórios, HTTP, FTP, URL, imagens, entre outras.

**Apêndice B – Funções PHP/bancos de dados:** apresenta uma lista de funções do PHP que podem ser utilizadas para realizar a integração com diversos SGBDs, como MySQL, PostgreSQL, InterBase, Oracle, SQL Server etc.

**Apêndice C – Tipos de recursos do PHP:** apresenta uma lista de funções que criam e finalizam os diversos recursos oferecidos pelo PHP.

**Apêndice D – Links interessantes:** indica diversos links para você se aprofundar ainda mais no estudo da linguagem PHP e de outras tecnologias. Os sites recomendados oferecem diversas ferramentas úteis para o desenvolvimento de sites dinâmicos e interativos, como tutoriais, scripts gratuitos e documentação detalhada.

# Capítulo 1

# O que é o PHP?

O PHP é uma das linguagens mais utilizadas na Web. Hoje mais de 10 milhões de sites no mundo inteiro utilizam PHP. A principal diferença em relação às outras linguagens é a capacidade que o PHP tem de interagir com o mundo Web, transformando totalmente os websites que possuem páginas estáticas.

Imagine, por exemplo, um website que deseja exibir notícias em sua página principal, mostrando a cada dia, ou a cada hora, notícias diferentes. Seria inviável fazer isso utilizando apenas HTML. As páginas seriam estáticas, e a cada notícia nova que aparecesse no site a página deveria ser alterada manualmente, e logo após enviada ao servidor por FTP (*File Transfer Protocol*) para que as novas notícias fossem mostradas no site. Com o PHP tudo isso poderia ser feito automaticamente. Bastaria criar um banco de dados onde ficariam armazenadas as notícias, e criar uma página que mostra essas notícias, “puxando-as” do banco de dados.

Agora imagine um site que possui cerca de 100 páginas. Suponha que no lado esquerdo das páginas há um menu com links para as seções do site. Se alguma seção for incluída ou excluída, o que você faria para atualizar as 100 páginas, incluindo ou excluindo esse novo link? Alteraria uma a uma, manualmente? Com certeza você demoraria horas para alterar todas as páginas. E isso deveria ser feito cada vez que houvesse alteração, inclusão ou exclusão de uma seção no site.

Para resolver esse problema utilizando PHP é muito simples. Basta construir um único menu, e fazer todas as 100 páginas acessarem esse arquivo e mostrá-lo em sua parte da esquerda. Quando alguma alteração fosse necessária, basta alterar um único arquivo, e as 100 páginas seriam alteradas automaticamente, já que todas acessam o mesmo menu.

Essas são apenas algumas das inúmeras vantagens das páginas que utilizam PHP. Você acabou de conhecer dois exemplos de sites em que a principal

característica é o dinamismo e a praticidade. Automatização de tarefas, economia de tempo e de mão-de-obra são características evidentes nos dois exemplos citados. Mais adiante veremos como implementar programas como os que foram citados aqui.

## Características do PHP

### Gratuito e com código aberto

Uma das grandes vantagens do PHP é que ele é gratuito. O arquivo de instalação pode ser obtido gratuitamente no site <http://www.php.net>. Este livro está baseado na documentação da versão 5 do PHP, pois esta versão oferece alguns recursos adicionais em relação às anteriores, como por exemplo o suporte à ferramenta SQLite (que veremos no capítulo 11). O PHP 5 apresenta ainda um desempenho superior às versões anteriores, principalmente no que diz respeito à programação orientada a objetos, que agora funciona de forma mais eficiente, não realizando cópias redundantes de dados.

No site oficial do PHP você encontrará sempre as versões mais atuais disponíveis para download, assim como as versões anteriores. A maioria dos conceitos e programas apresentados neste livro vale tanto para a versão 4 quanto para a versão 5 do PHP, porém é recomendável que você obtenha sempre a versão mais recente da linguagem, para poder aproveitar os novos recursos e instalar as correções para os defeitos (*bugs*) encontrados pelos desenvolvedores nas versões antigas.

Outra característica importante do PHP é que, além de ser gratuito, é um software com código-fonte aberto. O código-fonte do PHP assim como sua documentação detalhada também estão disponíveis no site oficial.

### Embutido no HTML

Outra característica do PHP é que ele é embutido no HTML. Veremos mais adiante as facilidades que isso pode nos trazer. Uma página que contém programação PHP normalmente possui extensão *.php* (isso depende da configuração do seu servidor Web). Sempre que o servidor Web receber solicitações de páginas que possuem essa extensão, ele saberá que essa página possui linhas de programação. Porém, você verá que o HTML e o PHP estão misturados, pois começa a escrever em PHP, de repente escreve um trecho em HTML, depois volta para o PHP, e assim por diante.

Há um exemplo simples para facilitar a compreensão: você já deve ter visto alguns sites que exibem a data e hora atual em suas páginas. Se essas informações forem escritas utilizando JavaScript, a data e hora mostradas serão retiradas do relógio do seu computador. Ou seja, para cada pessoa que acessar, a data e hora mostradas serão diferentes, pois nem todos os computadores marcam exatamente o mesmo horário. Agora, se a data e hora forem escritas utilizando PHP, essas informações serão retiradas do relógio do servidor, ou seja, há um relógio único, e por isso todos que acessarem o site ao mesmo tempo verão a mesma data e hora.

## Bancos de dados

Diversos bancos de dados são suportados pelo PHP, ou seja, o PHP possui código que executa funções de cada um. Entre eles temos MySQL, PostgreSQL, Sybase, Oracle, SQL Server e muitos outros. Cada um dos bancos de dados suportados pelo PHP possui uma série de funções que você poderá usar em seus programas para aproveitar todos os recursos. Os bancos de dados não suportados diretamente pelo PHP podem ser acessados via ODBC. Neste livro veremos exemplos de utilização do PostgreSQL e do MySQL, mas você poderá programar utilizando qualquer outro banco de dados, para isso basta fazer a adaptação dos comandos referentes a ele. Comandos utilizados por outros bancos de dados são encontrados na documentação do PHP, disponível para download no site oficial.

## Portabilidade

Podemos executar o PHP no Linux, Unix ou Windows NT. Vamos falar mais sobre a utilização do PHP no Linux, embora haja poucas diferenças em relação aos demais sistemas operacionais.

# Capítulo 2

# Instalação do PHP

Agora vamos ver o que é necessário em termos de hardware e software antes de você começar a escrever seus programas em PHP. Este livro não se destina a dar explicações detalhadas sobre instalações de softwares, no entanto este item esclarecerá muitas de suas dúvidas quanto aos recursos necessários para utilização do PHP. Vamos analisar cada caso.

## Utilizando um provedor de hospedagem

Se o objetivo é apenas aprender a utilizar a programação PHP e desenvolver um site utilizando essa linguagem, sem ficar se preocupando com instalações de softwares, é recomendável acessar diretamente um provedor de hospedagem. Existem dezenas desses provedores aqui no país, e você achará vários se procurar em qualquer site de busca. Alguns deles oferecem bons serviços por preços bem acessíveis, que variam mais ou menos de R\$5,00 a R\$39,90 por mês, dependendo do espaço (em MB) e dos recursos fornecidos. Esses provedores já possuem servidores conectados à Internet, com o PHP instalado e todas as ferramentas de bancos de dados necessárias. Alguns provedores de hospedagem oferecem suporte a PHP com banco de dados MySQL e outros oferecem o PostgreSQL. Essa é uma boa solução para quem quer criar seu site de forma rápida e eficiente, pois os provedores de hospedagem lhe fornecem uma conta de FTP (*File Transfer Protocol*), que você utilizará para enviar suas páginas ao servidor, e posteriormente acessá-las pela Web, sem precisar fazer nenhuma configuração de softwares e servidores.

## Instalando a partir de uma distribuição Linux

Se você tem um bom equipamento (hardware), e quiser montar seu próprio servidor para instalar nele os softwares necessários, uma boa alternativa é ins-

talar os sistemas operacionais que já acompanham todos os pacotes necessários, incluindo linguagens de programação, compiladores, servidor Web, banco de dados etc. Um exemplo que pode ser citado é o do Linux, em suas diversas versões. As distribuições mais atuais do Linux apresentam um programa de instalação bastante amigável, em que você pode escolher os pacotes que deseja instalar, e o programa de instalação faz tudo para você. Para utilizar o PHP você terá de selecionar os seguintes pacotes para instalação:

- PHP: a linguagem de programação.
- Apache: é o servidor Web. O Apache é o servidor Web mais indicado, pois o PHP roda como um módulo nativo dele.
- MySQL ou PostgreSQL: gerenciador de banco de dados. Selecione aquele que acompanha a distribuição Linux que você estiver instalando.

É importante lembrar que o PHP é uma linguagem voltada para a Web, portanto deve haver um servidor Web, que receba as solicitações das páginas, faça o processamento pelo PHP, e retorne ao browser um resultado.

## Instalando manualmente no Linux ou Windows

Se você quiser instalar o PHP manualmente, ele está disponível para download no site oficial, no endereço:

<http://www.php.net/>

Acessando a seção “downloads”, você poderá obter sempre a última versão da linguagem. Na versão para Linux, o PHP precisará ser compilado em seu sistema operacional. Para obter mais detalhes, consulte o arquivo `install.txt` que acompanha a distribuição. Na versão Windows, a distribuição está disponível em um arquivo compactado ZIP, que já contém os arquivos binários. Basta descompactá-lo em algum diretório do seu computador.

Para poder acessar seus programas pelo navegador, você precisará também de um servidor Web. O mais indicado é o Apache, que pode ser obtido em:

<http://httpd.apache.org/>

O arquivo `install.txt`, que acompanha o PHP, contém as instruções para a configuração do PHP no Apache. Se você pretende trabalhar com banco de dados, duas opções interessantes são o MySQL e o PostgreSQL, que podem ser obtidos respectivamente em <http://www.mysql.com> e <http://www.postgresql.org>.

Se você tiver dificuldades de instalar o PHP no Windows, pode consultar o roteiro de instalação disponibilizado no site pessoal do autor, em <http://www.niederauer.com.br/livros/php/roteiro.html>.

# Capítulo 3

## Noções básicas de programação

Você que está iniciando agora no mundo da programação deve prestar bastante atenção nos próximos tópicos, pois vamos falar de conceitos de programação e suas implementações, utilizando o PHP. Mesmo que você já tenha certa experiência em programação, leia os próximos tópicos para ir se acostumando com a sintaxe do PHP.

### Começando a programar

Bom, vamos ao que interessa: a parte prática. Agora começaremos a escrever programas em PHP. Abra qualquer editor de textos que você tiver (pode ser o Bloco de Notas do Windows, ou o VI do Linux). No próximo item veremos como é estruturado um programa em PHP, e você entenderá um pouco melhor. Agora digite as seguintes linhas em seu editor:

```
<html>
<body>
<?php
// Legal, estou escrevendo meu primeiro programa em PHP
echo "<h2 align='center'>Parabéns para mim!</h2>";
?>
</body>
</html>
```

Salve esse programa como *prog1.php* e envie para o diretório que você está utilizando para hospedar o site. Seu primeiro programa vai gerar como resultado a frase “*Parabéns para mim!*” centralizada na página. Para ver o resultado, basta você acessar pelo browser o endereço [http://<seu\\_endereço>/prog1.php](http://<seu_endereço>/prog1.php), onde você deve substituir *<seu\_endereço>* pelo endereço do servidor que está utilizando para rodar os programas PHP.

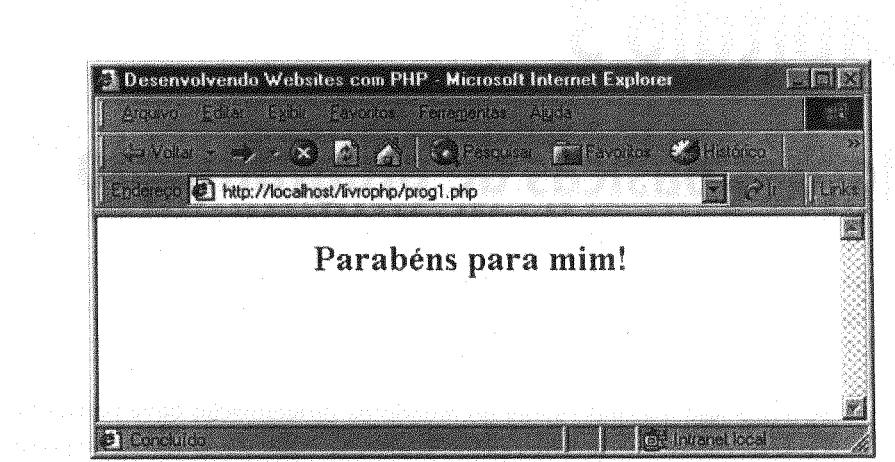


Figura 3.1 – Resultado do seu primeiro programa.

Fácil, não? A seguir veremos com detalhes o que significa cada uma das linhas que você digitou nesse programa.

Todo o trecho de programação PHP deve estar entre as tags `<?php` e `?>`, para que o servidor Web saiba que esse trecho deve ser processado. Vejamos então cada elemento do programa *prog1.php*.

| Elemento              | Descrição   |
|-----------------------|---|
| <code>&lt;?php</code> | Informa que aqui começa um programa PHP.  |
| <code>//</code>       | Representam uma linha de comentário. Tudo que vem após estas barras na mesma linha é ignorado pelo PHP. Os comentários são muito úteis para uma boa documentação do seu programa. As duas barras servem para transformar uma única linha em comentário, mas você pode usar o <code>/*</code> para iniciar uma seqüência de comentários, e depois finalizar os comentários com o <code>*/</code> . |
| <code>echo</code>     | É um dos comandos mais utilizados em PHP. Serve para escrever alguma coisa na tela.   |
| <code>?&gt;</code>    | Informa que aqui termina o programa PHP.  |

Se você escolher a opção *Exibir-Código-fonte* em seu browser, você verá o código que seu browser recebeu, que foi o seguinte:

```

<html>
  <body>
    <h2 align='center'>Parabéns para mim!</h2>
  </body>
</html>

```

Note que o browser não recebe nenhuma linha em PHP, somente recebe código HTML puro, pois, como já vimos, o PHP roda no servidor. Toda a programação PHP é processada no servidor, que retorna somente o resultado final para seu browser.

**Dica:** quando as páginas possuem extensão .html, o servidor Web as tratará como HTML puro, e não reconhecerá códigos PHP. Se a página possuir extensão .php, o servidor Web ativará o processador de hipertexto do PHP para verificar linha a linha em busca de códigos de programação, por isso o processo fica um pouco mais lento. A dica é a seguinte: só coloque extensão .php nas páginas que realmente possuem códigos PHP, senão você estará gastando um tempo desnecessário, procurando a cada linha códigos que não existem na página.

Em relação à velocidade de processamento, podemos fazer uma pequena comparação entre o PHP e a linguagem ASP (Active Server Pages), da Microsoft. Mesmo que as páginas .php demorem um pouco mais para serem processadas do que as páginas com HTML puro, elas são processadas muito mais rápido que aquelas que usam programação ASP. Diversos testes realizados por programadores americanos já comprovaram isso. Além disso, o PHP possui um gerenciamento de memória muito superior ao do ASP.

## Estrutura de um programa PHP

Um programa escrito em PHP pode possuir comandos HTML e códigos PHP.

### Código PHP e comandos HTML

Os comandos HTML devem aparecer fora das tags <?php e ?, pois estas limitam um trecho de programa PHP. Dentro dessas tags até podem aparecer comandos HTML, mas somente se utilizarmos o comando echo para escrevê-los.

Você pode ir concatenando scripts PHP com comandos HTML, podendo dessa forma, escrever vários scripts PHP em uma única página. Cada script PHP existente na página deve começar com a tag <?php, e terminar com ?. As linhas de programação que serão escritas entre as tags devem sempre terminar com ; (ponto-e-vírgula), senão ocorrerão erros no momento da execução da página. Entre essas tags você pode escrever programas, utilizando todos os recursos que o PHP lhe oferece, como definição e chamada de funções, acesso a banco de dados, atribuição de valores a variáveis, fórmulas matemáticas etc.

Toda essa mistura entre o HTML e o PHP é muito útil, pois nós utilizamos o PHP para gerar os dados dinamicamente, enquanto o HTML é usado para formatar e exibir esses dados nas páginas mostradas no browser.

Vamos ver um exemplo que mistura HTML e PHP para mostrar a data atual. Digite o seguinte programa em seu editor de textos, e salve-o como *prog2.php*:

```
<html>
<body>
<?php
    $data_de_hoje = date ("d/m/Y",time());
?>
<p align="center">Hoje é dia <?php echo $data_de_hoje; ?></p>
</body>
</html>
```

Perceba a combinação existente entre os comandos HTML e o código PHP. No início do programa atribuímos à variável *\$data\_de\_hoje* a data atual, utilizando o comando *date*. Essa variável estará disponível para uso em qualquer parte da página. Depois utilizamos comandos HTML para escrever “Hoje é dia”, e completamos abrindo um novo trecho de PHP, escrevendo a data atual armazenada na variável *\$data\_de\_hoje* por meio do comando *echo*.

## Exibindo a página no browser

Para o browser mostrar alguma coisa na tela é necessário que a página tenha pelo menos um comando *echo* para escrever algo, ou então comandos HTML que escrevam o conteúdo da página. Porém, somente se for usado o comando *echo* ou algum outro comando PHP que produza uma saída na tela, você realmente terá informações dinâmicas, pois o HTML é estático, imprime sempre as mesmas informações na tela.

Veja o seguinte exemplo:

```
<html>
<body>
<?php
    $dia = date ("d/m/Y",time());
    $base = 5.5;
    $altura = 10;
    $area = $base * $altura;
?>
</body>
</html>
```

Salve esse programa como *prog3.php*, e veja o resultado em seu navegador. Perceba que não há nenhum comando *echo* no programa, por isso seu navegador mostrará uma tela em branco. O que ocorreu foi que os valores atribuídos às variáveis ficaram armazenados apenas na memória, mas não foram mostrados na tela. Ao visualizar o código-fonte recebido pelo navegador, você verá apenas as tags do HTML:

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

Vejamos agora um exemplo parecido com o anterior, mas dessa vez vamos utilizar o comando *echo* para mostrar informações na tela. Digite o seguinte programa em seu editor de textos e salve-o como *prog4.php*:

```
<html>
<body>
<?php
    $time = "Grêmio";
    $ano = 1983;
    $frase1 = "o $time é o melhor clube de futebol do mundo!";
    $frase2 = "o $time foi campeão do mundo em $ano";
    echo "<h3>$frase1</h3>";
    echo "<h3>$frase2</h3>";
?
</body>
</html>
```

Abrindo essa página em seu browser, você obterá o seguinte resultado:

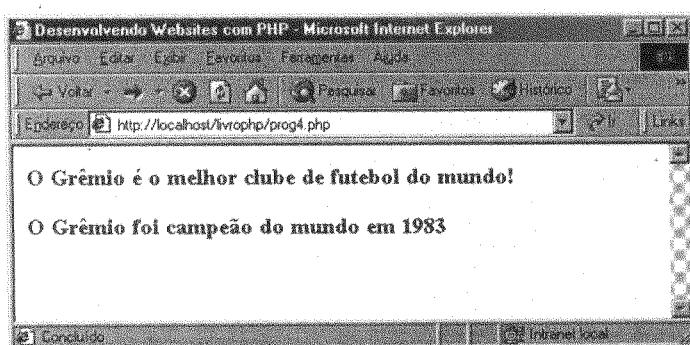


Figura 3.2 – Resultado gerado pelo programa *prog4.php*.

Com esses exemplos você descobriu dois dados fundamentais:

- 1) a importância do comando `echo`: para gerar dados dinâmicos é fundamental dominar esse comando.
  - 2) a interpolação de variáveis: daqui em diante você vai usar muito essa técnica. Trata-se da inclusão do valor de uma variável dentro da outra, como, por exemplo, em nosso programa `prog4.php`, em que na variável chamada `$frase1` incluímos o valor da variável `$time`, e na variável `$frase2` incluímos o valor da variável `$time`, e também da variável `$ano`. Ou seja, dentro da string que é atribuída a uma variável podemos colocar outra variável, e na hora do processamento do programa o PHP substituirá a variável pelo seu valor.

Lembrando que em PHP as variáveis começam sempre pelo símbolo de cifrão (\$). Quando houver um cifrão diante de um nome qualquer, o PHP interpretará isso como uma variável.

# Capítulo 4

## Manipulando os dados em PHP

Esse item é um dos mais importantes para você dominar a sintaxe da linguagem, pois agora veremos como devem ser tratados valores numéricos, strings, arrays (vetores), constantes, valores obtidos por meio de formulários etc. Veremos também como criar dinamicamente novas variáveis em seu programa. Ao contrário de outras linguagens, como, por exemplo, a linguagem C, no PHP não é necessário fazer declaração de variáveis, basta fazer-lhe diretamente a atribuição de um valor.

### Dados numéricos

São números inteiros, reais, positivos, decimais, octais e hexadecimais. Os dados numéricos são utilizados geralmente para efetuar cálculos. Para representar números muito pequenos ou muito grandes, pode-se usar a notação científica. Por exemplo: 1.500.000.000 pode ser expresso como 1.5E+9.

Veja alguns exemplos de dados numéricos:

| Dados    | Descrição   |
|----------|---|
| 5        | Valor inteiro na base decimal.  |
| 4.012    | Valor real (ou ponto flutuante) com três casas decimais.  |
| .14      | Valor real com duas casas decimais. É o mesmo que 0.14. Quando o número começa com um ponto o 0 é considerado como algarismo inicial.   |
| 033      | Valor inteiro na base octal. Todo valor iniciado com 0 é tratado como base 8 (na base 8 são utilizados apenas os algarismos 0,1,2,3,4,5,6 e 7).   |
| 0xBC     | Valor inteiro na base hexadecimal. Todo valor iniciado com 0x é tratado como base 16 (na base 16 são utilizados os 10 algarismos do sistema decimal e mais as letras A,B,C,D,E e F, que representam os valores de 10 (A), 11 (B), 12 (C), 13 (D), 14 (E) e 15 (F)). |
| 43000000 | É um número real grande, que pode ser expresso utilizando a notação científica: 4.3E+7.   |

## Dados alfanuméricos (textos)

Também conhecidos como strings. São seqüências de caracteres, que podem ser delimitadas por aspas simples ('), aspas duplas ("") ou aspas invertidas (''), dependendo da utilização desejada. O PHP trata os dados alfanuméricos de forma diferente de acordo com o delimitador utilizado. Vamos acompanhar exemplos envolvendo cada um dos delimitadores, e verificar as diferenças entre os três tipos:

### Aspas simples ('')

As aspas simples são muito parecidas com as aspas duplas, mas existem pequenas diferenças entre elas. Esse tipo de aspas pode ser usada para delimitar qualquer dado alfanumérico (seqüência de caracteres), como por exemplo:

```
'<p align=center>Texto utilizando aspas simples</p>'
```

Porém devemos tomar cuidado com textos que possuem o caractere ' em sua seqüência de caracteres, como, por exemplo, o seguinte:

```
'Welcome to the John's Page'
```

Como existe uma aspa simples no meio da seqüência de caracteres, o PHP vai interpretá-la como delimitador de finalização do texto, e isso causará um erro de execução. Esse problema pode ser facilmente resolvido, inserindo antes da aspa o caractere de controle \ (barra invertida), que indica ao PHP que aquela aspa deve ser tratada como um texto comum, e não como um delimitador.

O caractere de controle (\) também é bastante usado quando utilizamos aspas duplas e queremos incluir o símbolo \$ dentro de um dado alfanumérico. Sabemos que se colocarmos o símbolo \$ sem a barra invertida antes, o PHP interpretará isso como uma variável, e procurará o valor dela para inserir. Se quisermos que apareça o símbolo \$, basta escrever \\$ dentro da seqüência de caracteres.

Lembre-se de que no momento da exibição na tela não será mostrado o caractere \, somente a seqüência de caracteres, pois o caractere \ apenas indica ao PHP que o caractere faz parte da seqüência.

Você já deve ter visto em outras linguagens que para gerar uma quebra de linha em algum texto devemos inserir o caractere \n. Isso não é necessário quando utilizamos as aspas simples como delimitador. Se quisermos inserir uma quebra de linha em um texto com aspas simples, basta dar um ENTER no meio do texto, e você verá o resultado na saída do programa, conforme mostra o exemplo a seguir:

```
<html>
<body>
<?php
    echo '<p align=center>Isto é um teste
Estou aprendendo a escrever textos com aspas simples!</p>';
?>
</body>
</html>
```

Salve esse programa como *prog5.php*. É importante deixar bem claro que as quebras de linha existentes no meio do texto serão apresentadas apenas se a saída do programa for em formato texto, ou seja, no formato HTML você não verá nenhuma quebra de linha ao executar a página *prog5.php*, conforme mostrado na figura 4.1:

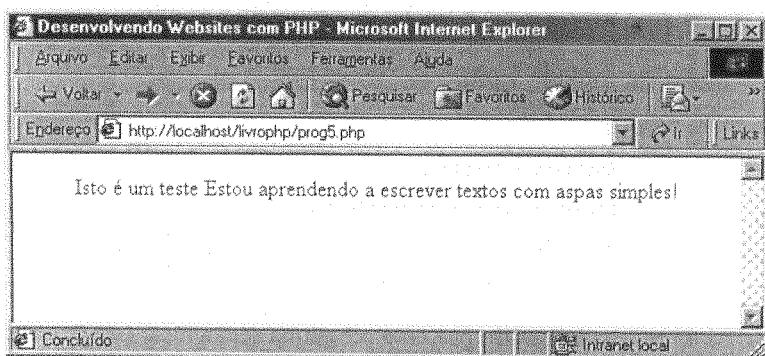


Figura 4.1 – Resultado gerado pelo programa *prog5.php*.

Se você pedir para visualizar o código-fonte que o browser recebeu será possível ver o seguinte resultado:

```
<html>
<body>
<p align=center>Isto é um teste
Estou aprendendo a escrever textos com aspas simples!</p>
</body>
</html>
```

Você pode notar que as quebras de linhas foram mostradas no código-fonte recebido, pois este está em formato texto. Se seu objetivo é apresentar quebras de linha no formato HTML, para que sejam mostradas na tela, deve-se usar a tag *<br>* do HTML. Logo a seguir temos um exemplo do comando *echo* em que são apresentadas as quebras de linhas no browser:

```

echo 'Isto é um teste';
echo '<br>';
echo 'Coloquei uma quebra de linha';

```

O comando apresentado produzirá em seu navegador a frase *Isto é um teste*, e na linha de baixo a frase *Coloquei uma quebra de linha*. A tag *<br>* gerou uma nova linha na saída HTML.

## Aspas duplas ("")

As aspas duplas são muito parecidas com as aspas simples. Uma diferença importante entre esses dois tipos de aspas é que, utilizando aspas duplas, podemos fazer uma interpolação de variáveis. Como já vimos anteriormente, a interpolação de variáveis consiste em incluir o valor de uma variável dentro da outra, como mostra o trecho de programa a seguir:

### exemplo4\_1.php

```

<?php
$palavra = "teste";
$frase = "Isto é um $palavra";
echo $frase;
?>

```

O resultado gerado por esse programa será a frase *Isto é um teste*.

Outra diferença entre as aspas simples e duplas é que, utilizando aspas duplas como delimitadores, podemos incluir seqüências de caracteres de controle nos dados alfanuméricos. A tabela a seguir mostra algumas seqüências de controle que podem ser inseridas quando utilizamos aspas duplas:

| Controle | Significado                        |
|----------|------------------------------------|
| \"       | Insere no texto o caractere "      |
| \n       | Nova linha                         |
| \r       | Retorno de carro (carriage return) |
| \t       | Tabulação                          |
| \\$      | Insere no texto o símbolo \$       |
| \\\      | Insere no texto o caractere \      |
| \0nnn    | Caractere ASCII octal              |
| \xnn     | Caractere ASCII hexadecimal        |

Além disso, dentro de um dado alfanumérico delimitado por aspas duplas podemos inserir normalmente o caractere ' (aspas simples), como, por exemplo, no seguinte texto:

"welcome to the John's Page"

Quando utilizávamos aspas simples, tínhamos de colocar o caractere de controle \ para não haver erros de execução nesse texto, mas agora com as aspas duplas isso não é necessário. Porém, devemos tomar cuidado ao inserir o caractere " dentro de um texto. Veja o exemplo:

"Estou colocando "aspas duplas" dentro de um texto"

Assim como ocorreu com as aspas simples, o texto anterior também apresentará problemas de execução, pois o PHP interpretará a segunda aspa como delimitador de finalização de texto. Para resolver esse problema, basta inserir o caractere de controle (ou caractere de escape) \, que indica ao PHP que aquelas aspas fazem parte do texto. O texto então ficaria da seguinte forma:

"Estou colocando \"aspas duplas\" dentro de um texto"

## Aspas invertidas ('')

Utilizando aspas invertidas como delimitadores, estamos usando uma função muito interessante que o PHP nos oferece: executar comandos do sistema operacional por meio de um programa PHP. Dessa forma podemos enviar ao Linux, ao Windows, ou qualquer outro sistema operacional que você esteja utilizando, comandos para serem executados por ele. Muitas vezes podemos exibir no browser o resultado gerado por comandos do sistema operacional. Podemos, por exemplo, ver a listagem de arquivos HTML existentes em determinado diretório do Linux, como mostra o exemplo a seguir:

### exemplo4\_2.php

```
<html>
<body>
<?php
echo 'ls -l *.html';
?>
</body>
</html>
```

É importante lembrar que você só conseguirá executar comandos do sistema operacional possuindo privilégios para executá-los, e isso depende de como o seu administrador configurou o sistema. Não será possível, por exemplo, ver o conteúdo de diretórios protegidos, aos quais você não tem acesso autorizado.

## Constantes

São valores que são predefinidos no início do programa, e que não mudam ao longo de sua execução. Você pode definir suas próprias constantes utilizando o comando `define`, que possui a seguinte sintaxe:

```
bool define (string nome, misto valor [, bool case_insensitive])
```

Onde “`nome`” é o nome que você vai utilizar para representar a constante, “`valor`” é um valor qualquer (numérico ou alfanumérico) a ser atribuído a ela e “`case_insensitive`” é um valor lógico (`true` ou `false`) que indica se o PHP deve diferenciar letras maiúsculas e minúsculas quando houver uma referência a essa constante. Veja o exemplo a seguir, que mostra como devemos usar as constantes:

### exemplo4\_3.php

```
<html>
<body>
<?php
    define ("meunome","Juliano");
    define ("peso",78);
    echo "O meu nome é " . meunome;
    echo "<br>";
    echo "O meu peso é " . peso . " quilos";
?
</body>
</html>
```

Executando esse programa, você terá o seguinte resultado em seu navegador:

```
O meu nome é Juliano
O meu peso é 78 quilos
```

Note que no exemplo que acabamos de ver, referenciamos as constantes diretamente pelo nome que escolhemos, e não utilizamos na frente delas o símbolo \$, pois esse símbolo é utilizado apenas para representar variáveis.

Outro recurso que utilizamos no exemplo foi a concatenação, representada pelo ponto (.). Podemos concatenar quantos dados quisermos, e todos eles serão exibidos como apenas uma seqüência de caracteres.

Além de você poder definir suas próprias constantes, o PHP já possui diversas constantes próprias predefinidas. Vamos conhecer algumas na tabela a seguir:

| Constante          | Descrição  |
|--------------------|--|
| <b>TRUE</b>        | Valor verdadeiro (utilizado para comparação).                                  |
| <b>FALSE</b>       | Valor falso.   |
| <b>_FILE</b>       | Contém o nome do script que está sendo executado.                              |
| <b>_LINE</b>       | Contém o número da linha do script que está sendo executado.                   |
| <b>PHP_VERSION</b> | Contém a versão corrente do PHP.   |
| <b>PHP_OS</b>      | Nome do sistema operacional no qual o PHP está rodando.                        |
| <b>E_ERROR</b>     | Exibe um erro ocorrido em um script. A execução é interrompida.                |
| <b>E_WARNING</b>   | Exibe uma mensagem de aviso do PHP. A execução não pára.                       |
| <b>E_PARSE</b>     | Exibe um erro de sintaxe. A execução é interrompida.                           |
| <b>E_NOTICE</b>    | Mostra que ocorreu algo, mas não necessariamente um erro. A execução não pára. |

## Variáveis em PHP

As variáveis servem para armazenar dados que podem ser usados em qualquer ponto do programa. Cada variável está associada a uma posição de memória de seu computador. Ao contrário de linguagens tradicionais, como C, Pascal e Delphi, no PHP não é necessário fazer declaração de variáveis. Basta atribuir diretamente um valor a ela, e a partir desse momento já está criada e associada a um tipo (numérico, alfanumérico etc.), dependendo do valor que lhe foi atribuído.

### Exemplos

Já vimos que em PHP as variáveis devem iniciar com o símbolo \$. Após esse símbolo deve vir o identificador da variável, ou seja, o nome pelo qual ela será referenciada durante a execução do programa. Esse identificador não pode iniciar com um número. Os números podem aparecer em qualquer posição do identificador, menos na primeira. Exemplos de variáveis válidas e inválidas:

| Válidas                          |
|----------------------------------|
| \$nota1                          |
| \$casal20                        |
| \$bisc8                          |
| \$gremio_2_vezes_campeao_américa |

| Inválidas     |
|---------------|
| \$100vergonha |
| \$5           |
| \$20assustar  |
| \$60nacadeira |

## Maiúsculas e minúsculas (case-sensitive)

É recomendável que você utilize sempre identificadores com letras minúsculas, pois o PHP faz a distinção entre maiúsculas e minúsculas. Se você começar a misturar os dois tipos de letras, pode ocorrer uma confusão na utilização da variável, já que `$nota_aluno` não é a mesma coisa que `$Nota_aluno`. Imagine se um aluno tirou nota 10, e esse valor foi armazenado na variável `$nota_aluno`. Na hora de imprimir a nota do aluno, você digita a seguinte linha:

```
<?php
```

```
echo $Nota_aluno;
```

```
?>
```

Pobre coitado, está reprovado, pois você imprimiu a variável errada. Por isso tome cuidado: o PHP distingue letras maiúsculas e minúsculas.

## Escopo das variáveis

Mais adiante veremos o uso de funções (*functions*) em PHP. Porém, você já deve conhecer um pouco sobre o assunto. Funções geralmente são trechos de código que utilizamos com freqüência, e para que não tenhamos de repetir o código a toda hora, cria-se uma rotina com aquele código, e quando necessário ativa-se essa rotina. Veremos com detalhes esse assunto mais adiante.

Quando uma variável é utilizada dentro de uma função, pode haver uma outra variável com o mesmo nome que é utilizada em outra função, ou no código do programa principal. São espaços de memória diferentes, e cada uma funciona dentro do seu contexto, ou seja, a variável usada dentro da função funciona só ali dentro. Fora dali seu valor não é acessível em nenhuma outra parte do programa.

No entanto, se quisermos, podemos usar dentro de uma função o valor de uma variável existente também no programa principal, e para isso há duas formas:

- 1) defini-la como global no início da função;
- 2) utilizar o array predefinido `$GLOBALS`, que utiliza os nomes das variáveis como chave associativa.

Vamos ver exemplos envolvendo todos os tipos citados:

### exemplo4\_4.php

```
<?php  
$num = 5000;  
function testa_escopo1 ()  
{  
    $num += 5;  
    echo $num . "<br>";  
}  
echo $num . "<br>";  
testa_escopo1();  
?>
```

Após a execução do exemplo serão mostrados na tela os valores 5000 e na linha de baixo 5. Dependendo da configuração do seu PHP, ele poderá mostrar até uma mensagem de aviso, informando que existe uma variável não definida (*Undefined variable*). Isso ocorre porque a variável `$num` existente dentro da função `testa_escopo1` é válida somente dentro da função, e quando somamos seu conteúdo (que dentro da função é nulo) com 5, temos como resultado o 5. O comando `echo` executado fora da função mostrará o valor da variável `$num` atribuído no início (5000), pois as alterações feitas dentro da função não alteram o valor da variável global.

Muitas vezes temos a necessidade de utilizar o valor da variável global dentro das funções. Vejamos um exemplo:

### exemplo4\_5.php

```
<?php  
$num = 5000;  
function testa_escopo1 ()  
{  
    global $num;  
    $num += 5;  
    echo $num . "<br>";  
}  
echo $num . "<br>";  
testa_escopo1();  
?>
```

Após a execução deste programa o resultado apresentado será o seguinte:

O Grêmio foi Campeão da América em 1983 e 1995

Com certeza você utilizará bastante a interpolação de variáveis. Deve-se tomar cuidado, porém, quando se escrever uma variável dentro de uma string e houver um ou mais caracteres logo ao lado da variável. Exemplo:

#### exemplo4\_9.php

```
<?php  
$x = "tri";  
echo "Eu sou $xcolor";  
?>
```

O objetivo desse programa seria imprimir na tela a frase *Eu sou tricolor*. Mas se você executá-lo a única coisa que será mostrada na tela é o seguinte:

Eu sou

É possível até que seja exibida uma mensagem de aviso, informando que existe uma variável não definida (*Undefined variable*). Isso ocorre porque o PHP procurará pelo valor de uma variável chamada *\$xcolor*, que na verdade não existe, e por isso nada será mostrado nesse local. Para resolver esse problema, existem duas maneiras. A primeira é a seguinte:

#### exemplo4\_10.php

```
<?php  
$x = "tri";  
echo "Eu sou ${x}color";  
?>
```

Uma maneira é colocar a variável entre chaves para que o PHP saiba onde termina o identificador. Outra maneira é escrever o comando de forma diferente, sem usar a interpolação de variáveis e usando a concatenação:

#### exemplo4\_11.php

```
<?php  
$x = "tri";  
echo "Eu sou " . $x . "color";  
?>
```

Escolha a forma que lhe parecer mais simples.

## Variáveis criadas durante a execução

Em diversas ocasiões é muito útil criarmos variáveis dinamicamente, ou seja, durante a execução do programa. Essa técnica funciona da seguinte maneira: utiliza-se o valor de uma variável para servir como identificador para outra que é criada. Para isso utilizamos duas vezes o símbolo \$, ou seja, devemos usar \$\$.

Acompanhe o exemplo a seguir:

```
<?php  
$texto = "Porto Alegre";  
$futuro_identificador = "cidade";  
$$futuro_identificador = $texto;  
  
echo "<h2 align=center>";  
echo "Minha cidade é $cidade";  
echo "</h2>";  
?>
```

Se você salvar esse programa como *prog6.php*, e executá-lo por meio de seu navegador, você verá a seguinte tela:

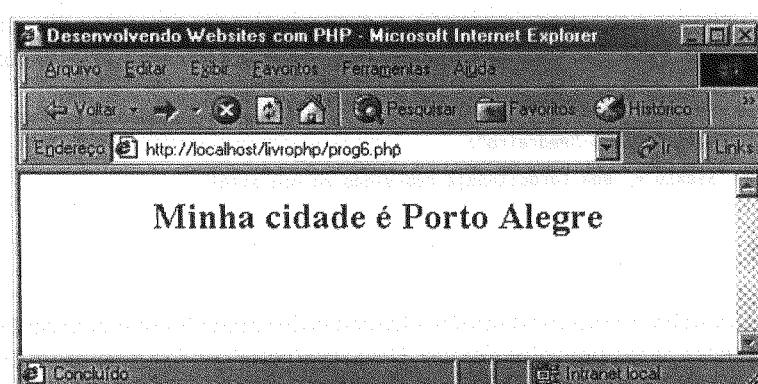


Figura 4.2 – Resultado gerado pelo programa *prog6.php*.

Veja que a variável \$ruim foi criada dinamicamente. Primeiro o valor “*cidade*” foi atribuído à variável \$futuro\_identificador, e depois com o uso de \$\$ o valor “*cidade*” tornou-se o identificador da variável recém-criada. A variável criada \$cidade recebeu o valor da variável \$texto, e logo após esse valor foi mostrado na tela após a frase “*Minha cidade é Porto Alegre*”.

Guarde bem esse conceito de variáveis com nomes dinâmicos, pois existirão situações em que esse recurso pode ser a única solução possível para resolver determinado problema.

## Tipos das variáveis

No PHP existem variáveis dos tipos numéricas, alfanuméricicas (strings), arrays e objetos. Vamos ver cada um desses tipos:

### Numéricas

As variáveis numéricas podem possuir valores inteiros ou reais (ponto flutuante). Uma variável é definida como numérica no momento em que atribuímos um valor numérico a ela. Veja alguns exemplos:

```
$numero = 10;
$x = 5;
$numero_hexa = 0x0b; // valor em hexadecimal
$y = 15.002;
$a = 200.3;
```

### Alfanuméricas (strings)

São cadeias de caracteres que, conforme vimos anteriormente, podem ser delimitadas por aspas simples ('') ou aspas duplas (""). Quando utilizamos aspas duplas, podemos incluir caracteres de controle no meio da string. Exemplos:

```
$nome = 'Cláudiomar';
$profissao = "Pedreiro";
$texto = "Boa tarde!\nSeja bem-vindo ao meu site!"
```

### Arrays

As variáveis comuns (também chamadas de variáveis escalares) podem armazenar apenas um valor por vez. Um array (vetor) pode armazenar vários valores ao mesmo tempo, pois se trata de uma estrutura de armazenamento que, assim como as variáveis, possui um identificador, mas além disso há um índice associado (que pode ser um número ou um texto), e cada índice indica uma posição de memória em que fica armazenado um elemento do array. O índice deve aparecer entre colchetes ([]]) logo após o identificador do array.

Vamos ver um exemplo para você entender melhor o conceito de array: na entrada de um edifício há um daqueles armários com diversas gavetas para guardar correspondências, uma para cada apartamento. Podemos comparar o armário com o array, e os apartamentos com os índices do array. Ou seja, exis-

te apenas um nome identificador, que é o armário, mas, se um morador do edifício chega para pegar suas correspondências, ele deve acessar a gaveta correspondente ao seu índice, que é o número do seu apartamento.

Os arrays são muito úteis quando precisamos realizar automatização de tarefas em nossos programas. Imagine que os nomes de todos os moradores de um edifício devem ser mostrados na tela. Obviamente não seria viável que utilizássemos variáveis escalares para armazenar os nomes. Se fossem 60 nomes, teríamos 60 variáveis, e para mostrar os valores na tela deveríamos usar 60 vezes o comando echo. Mas se os nomes dos 60 moradores estivessem guardados em um array, bastaria que utilizássemos um comando de repetição (que veremos mais adiante) para imprimir desde a primeira posição do array até a última, ou seja, variando o índice de 0 até 59. Veja a seguir alguns exemplos de armazenamento em arrays:

```
$vetor[0] = 30;  
$vetor[1] = 40;  
$vetor[5] = 50;  
$vetor[15] = 60;
```

Se não colocarmos o índice do vetor entre colchetes, o PHP procurará o último índice utilizado e incrementá-lo, armazenando assim o valor na posição seguinte do array, conforme mostra o exemplo a seguir:

```
$vet[] = "Grêmio";  
$vet[] = "Campeão";
```

Nesse exemplo teremos o valor “Grêmio” armazenado em \$vet[0] e o valor “Campeão” armazenado em \$vet[1].

Até agora só vimos exemplos em que o índice do array é um valor numérico, mas o índice também pode ser um texto, e nesse caso o texto é chamado de chave associativa.

```
$vetor["time"] = "Grêmio";  
$vetor["título"] = "Campeão da América";  
$vetor["ano"] = 1995;
```

Repare que cada posição do array pode ser de um tipo diferente. Os valores das posições referenciadas por *time* e *título* são do tipo string, mas o valor da posição referenciada por *ano* é numérico. Outra coisa que pode acontecer é o array possuir índices numéricos e strings ao mesmo tempo. Não há problema nenhum em usar os dois tipos de índices no mesmo array.

Existem também as matrizes, que são arrays multidimensionais. Essas estruturas de armazenamento também possuem um único identificador, mas possuem dois ou mais índices para referenciar uma posição de memória. Imagine que queremos armazenar na memória os nomes dos melhores clubes do futebol brasileiro, separando-os por Estados e cidades. Podemos fazer isso utilizando um array bidimensional, como mostra o exemplo a seguir:

```
$clube ["RS"] ["PortoAlegre"] = "Grêmio";
$clube ["RS"] ["Caxias"] = "Juventude";
$clube ["RS"] ["BentoGoncalves"] = "Esportivo";
$clube ["MG"] ["BeloHorizonte"] = "Atlético";
$clube ["MG"] ["Novalima"] = "Vila Nova";
$clube ["MG"] ["Ipatinga"] = "Ipatinga";
$clube ["SP"] ["SaoPaulo"] = "Corinthians";
$clube ["SP"] ["Americana"] = "Rio Branco";
```

O exemplo apresentado mostra um array bidimensional, mas podemos usar arrays com mais de duas dimensões, bastando acrescentar mais colchetes com seus respectivos índices. Outra forma de criar um array é por meio da função `array` do PHP. Veja o exemplo a seguir:

#### exemplo4\_12.php

```
<?php
    $vetor = array (10,50,100,150,200);
    echo $vetor[2] . "<br>";
    $vet = array (1, 2, 3, "nome"=>"Joaquim");
    echo $vet[0] . "<br>";
    echo $vet["nome"];
?>
```

Após a execução desse programa os resultados mostrados na tela serão os seguintes:

100

1  
Joaquim

Lembre-se de que o array se inicia na posição 0 (zero), por isso apesar de ser o terceiro elemento do array, o 100 foi o primeiro valor mostrado, pois seu índice é 2. Depois foi criado um array que possui índices numéricos e também uma chave associativa. Por meio do comando `echo` mostramos na tela os valores de duas posições desse array.

## Objetos

Veremos mais adiante o que são classes. Dentro das classes temos funções definidas. Criamos uma variável para instanciar uma classe, e essa variável é chamada de objeto. Um objeto pode acessar funções definidas dentro de uma classe. Se você alguma vez já estudou programação orientada a objetos, esse conceito deve lhe ser familiar. Veja um pequeno exemplo:

### exemplo4\_13.php

```
<?php  
    class Teste  
    {  
        function Saudacao() {  
            echo "Olá pessoal!";  
        }  
    }  
    $objeto = new Teste; // $objeto se torna uma instância da classe Teste  
    $objeto -> Saudacao();  
?>
```

Ao criar uma instância da classe *Teste* na variável *\$objeto*, podemos acessar as funções definidas dentro da classe. Esse programa mostrará a mensagem “Olá pessoal!”.

## Operadores

Por meio dos operadores nós informamos ao PHP o que deve ser executado. Exemplos: atribuir um valor a uma variável, realizar operações aritméticas (soma, subtração etc.), realizar comparação de valores, para testar se um é maior ou menor que o outro, etc. Vamos ver os seguintes tipos de operadores:

- Operadores aritméticos.
- Operadores binários.
- Operadores de comparação.
- Operadores de atribuição.
- Operadores lógicos.
- Operador ternário.

## Operadores aritméticos

Utilizando esses operadores, você poderá efetuar qualquer operação matemática com dados do tipo numérico, como, por exemplo, somar, subtrair, multiplicar, dividir etc. Confira a tabela com os operadores aritméticos do PHP:

| Operador | Operação         |
|----------|------------------|
| +        | Adição           |
| -        | Subtração        |
| *        | Multiplicação    |
| /        | Divisão          |
| %        | Resto da divisão |

O PHP possui também outros operadores aritméticos, que atuam em apenas um operando. Esses operadores são bastante úteis, pois nos permitem realizar de forma simples operações, como troca de sinal, incremento ou decremento de valor etc.

Se você já programou em linguagem C, deve lembrar do incremento utilizando o operador `++`. No PHP também é possível utilizá-lo. Vamos conhecer todos esses operadores com a tabela a seguir:

| Operador            | Descrição  |
|---------------------|--|
| <code>-oper</code>  | Troca o sinal do operando.   |
| <code>++oper</code> | Pré-incremento. Primeiro incrementa o valor do operando e depois realiza a operação. |
| <code>--oper</code> | Pré-decremento. Primeiro decremente o valor do operando e depois realiza a operação. |
| <code>oper++</code> | Pós-incremento. Primeiro realiza a operação e depois incrementa o operando.          |
| <code>oper--</code> | Pós-decremento. Primeiro realiza a operação e depois decremente o operando.          |

Os operadores mostrados na tabela também são conhecidos como operadores unários, pois necessitam apenas de um operando, ao contrário da adição, subtração e outras operações que necessitam de pelo menos dois operandos.

Se o objetivo for somente incrementar o valor de uma variável, pode-se simplesmente digitar o nome da variável seguida do operador `++`. Exemplo:

```
$contador++;
```

O pré-incremento (`++oper`) e o pós-incremento (`oper++`) diferem um do outro se tivermos uma atribuição de valor a uma variável ou a avaliação de uma expressão. Vamos ver um exemplo envolvendo os dois tipos para que seja mais bem entendida a distinção entre eles:

```
<html>
<body>
<?php
$a = 1;
$b = 3;
$c = 5;
$res1 = ++$b - $a;
$res2= $c- + $a;
$res3= -$a + $c++;
echo "a = $a<br> b = $b<br> c = $c<br><br>";
echo "res1 = $res1<br> res2 = $res2<br> res3 = $res3<br>";
?>
</body>
</html>
```

Salve esse programa como *prog7.php* e veja o resultado em seu navegador.

Você uma tela como a da figura 4.3:

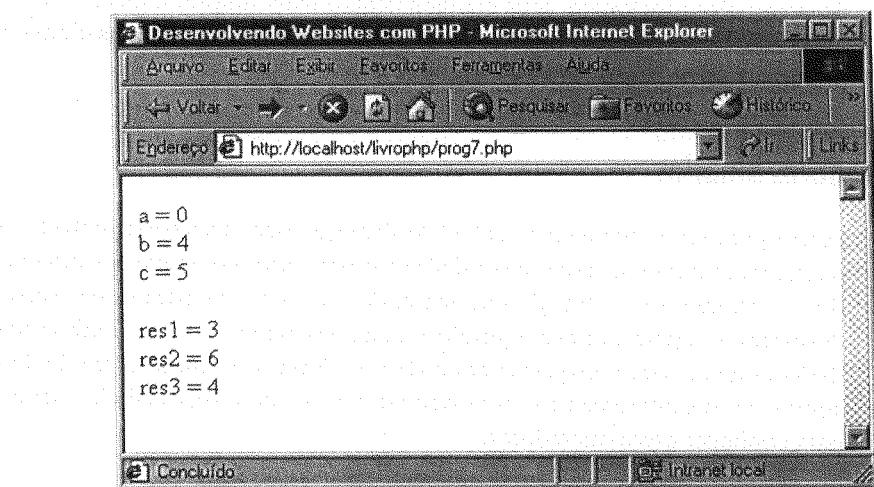


Figura 4.3 – Resultado gerado pelo programa *prog7.php*.

Agora vamos analisar os resultados. A variável *\$res1* recebeu o valor gerado pela expressão *++\$b - \$a*, ou seja, a variável *\$b* foi incrementada (passou do valor 3 para 4) e desse novo valor de *\$b* foi subtraído o valor da variável *\$a*, que vale 1. O resultado foi 4-1=3. O 3 foi atribuído à variável *\$res1*, como você pode ver na figura 4.3.

A variável \$res2 recebeu o valor gerado pela expressão `$c-- + $a`. Houve um pós-decremento na variável \$c, ou seja, primeiro foi feita a soma da variável \$c com a variável \$a, e o resultado foi atribuído à variável \$res2, e depois dessa atribuição o valor da variável \$c foi decrementado. Ao terminar a execução dessa linha a variável \$res2 ficou com o valor da soma `$c+$a`, que vale  $5+1 = 6$ , e a variável \$c, que possuía o valor 5 passou a valer 4 após o decremento.

A variável \$res3 recebeu o valor gerado pela expressão `--$a + $c++`. Houve um pré-decremento na variável \$a, que passou de 1 para 0, e esse valor foi somado ao valor da variável \$c, resultando em:  $0+4 = 4$ . Esse resultado (4) foi atribuído à variável \$res3, como você pode notar na figura apresentada. Após realizada essa operação foi feito o pós-incremento da variável \$c, que passou de 4 para 5. A Figura 4.3 mostra os valores finais de cada uma das variáveis após a execução do programa.

Na verdade esses operadores que acabamos de ver deixam seu programa muito mais simples em PHP, pois com o uso deles você pode fazer em apenas uma linha de código o que faria em duas ou mais linhas se não os usasse. Entendendo como eles funcionam, seus códigos ficarão mais simples e claros. Agora que você já está sabendo tudo de pré e pós incremento, vamos passar para o próximo tipo de operadores: os binários.

## Operadores binários

Esses operadores atuam em um nível de abstração bem mais baixo: trabalham diretamente com bits. Podem ser utilizados para fazer comparações binárias (bit a bit), inverter os bits de um operando, deslocar bits para direita (cada deslocamento para a direita equivale a uma divisão por 2) ou esquerda (cada deslocamento para a esquerda equivale a multiplicar o número por 2). Em alguns casos é interessante usar os operadores binários. Veja a tabela a seguir para conhecer esses operadores:

| Operador                    | Descrição                      |
|-----------------------------|--------------------------------|
| <code>~op1</code>           | Inverte os bits de op1.        |
| <code>op1 &amp; op2</code>  | Operação E (AND) bit a bit.    |
| <code>op1   op2</code>      | Operação OU (OR) bit a bit.    |
| <code>op1 ^ op2</code>      | Operação OU exclusivo (XOR).   |
| <code>op1 &gt;&gt; n</code> | Desloca op1 n bits à direita.  |
| <code>op1 &lt;&lt; n</code> | Desloca op1 n bits à esquerda. |

Vamos ver alguns exemplos:

**exemplo4\_14.php**

```
<html>
<body>
<?php
    $num = 14;
    $deslocado = $num >> 1; // desloca 1 bit para direita
    echo $deslocado;

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

No exemplo apresentado estamos pegando o valor 14 (que equivale a 1110 na base binária) e, deslocando um bit a direita, o que equivale a dividi-lo por 2. O resultado escrito na tela será a divisão de 14 por 2, que dá 7 (equivalente na base binária a 0111).

Perceba que o número que vem após o operando `>>` representa o número de bits que o operando será deslocado para a direita, portanto se tivermos o número 2, o número será dividido por 4 (duas divisões sucessivas por 2). Se após o operador `>>` houver um número  $n$ , estaremos dividindo o operando por  $2^n$ . Outro lembrete importante é que quando efetuarmos essas operações com bits, teremos sempre como resultado um número inteiro, conforme é mostrado no exemplo a seguir:

**exemplo4\_15.php**

```
<html>
<body>
<?php
    $num = 15;
    $resultado1 = $num >> 1; // desloca 1 bit para direita
    $resultado2 = $num << 2; // desloca 2 bits para esquerda
    echo $resultado1;
    echo "<br>";
    echo $resultado2;
?>
</body>
</html>
```

Após a execução desse programa, serão mostrados na tela os valores 7 e 60, que foram os valores armazenados respectivamente em `$resultado1` e `$resultado2`. Primeiro a variável `$resultado1` recebeu o número 15 (1111 em binário) deslocado 1 bit para a direita, que resulta em 7 (0111 em binário). Na linha seguinte a variável `$resultado2` recebeu o número 15 deslocado 2 bits para a esquerda, o que equivale a multiplicá-lo 2 vezes por 2, resultando em  $15 \times 2^2 = 60$  (111100 em binário).

## Operadores de comparação

Também chamados de condicionais. São aqueles que executam comparações entre o valor de duas variáveis, ou de uma variável e um texto, ou uma variável e um número. Com eles podemos testar por exemplo, se uma variável possui um valor maior que a outra ou se possui um valor maior que determinado número, ou se o retorno dado pela chamada de uma função é verdadeiro ou falso. Veja a tabela a seguir:

| Operador                      | Descrição  |
|-------------------------------|--|
| <code>op1 == op2</code>       | Verdadeiro se <code>op1</code> for igual a <code>op2</code> .          |
| <code>op1 &gt;= op2</code>    | Verdadeiro se <code>op1</code> for maior ou igual a <code>op2</code> . |
| <code>op1 &lt;= op2</code>    | Verdadeiro se <code>op1</code> for menor ou igual a <code>op2</code> . |
| <code>op1 != op2</code>       | Verdadeiro se <code>op1</code> for diferente de <code>op2</code> .     |
| <code>op1 &lt;&gt; op2</code> | Também serve para representar diferença.                               |
| <code>op1 &gt; op2</code>     | Verdadeiro se <code>op1</code> for maior que <code>op2</code> .        |
| <code>op1 &lt; op2</code>     | Verdadeiro se <code>op1</code> for menor que <code>op2</code> .        |

O operador de comparação `==` pode ser usado tanto na comparação de números como na comparação de textos, ao contrário de linguagens como C, que utilizam comandos específicos para comparação de dados alfanuméricos.

## Operadores de atribuição

Atribuição é o termo usado para representar a colocação de um valor em uma variável. A variável que receberá a atribuição encontra-se sempre do lado esquerdo do operador, e esta recebe o valor gerado pela expressão ou operador que está a direita. Além disso temos diversas variações dos comandos de atribuição, que podemos utilizar para facilitar a programação. São operadores que, assim como os operadores de incremento (`++`) e decremento (`--`), servem para deixar o código mais simples e mais fácil de ser programado. Veja a seguir a tabela dos comandos de atribuição:

| Operador     | Descrição                                |
|--------------|--|
| $op1 = op2$  | $op1$ recebe o valor de $op2$ .          |
| $op1 += op2$ | Equivale a $op1=op1+op2$ .               |
| $op1 -= op2$ | Equivale a $op1=op1-op2$ .               |
| $op1 *= op2$ | Equivale a $op1=op1*op2$ .               |
| $op1 /= op2$ | Equivale a $op1=op1/op2$ .               |
| $op1 .= op2$ | Concatenação: equivale a $op1=op1.op2$ . |
| $op1 %= op2$ | Equivale a $op1=op1 \% op2$ .            |
| $op1 <= op2$ | Equivale a $op1=op1 <= op2$ .            |
| $op1 >= op2$ | Equivale a $op1=op1 >= op2$ .            |
| $op1 &= op2$ | Equivale a $op1=op1 \& op2$ .            |
| $op1  = op2$ | Equivale a $op1=op1   op2$ .             |
| $op1 ^= op2$ | Equivale a $op1=op1 ^ op2$ .             |

**Dica:** muitas vezes seus programas podem apresentar problemas devido à troca do operador de comparação == pelo operador de atribuição =. O programa acaba gerando resultado incorreto, pois se queremos fazer uma comparação, por exemplo, entre as variáveis \$a e \$b, devemos usar a expressão \$a==\$b, e não \$a=\$b. Use com atenção esses operadores para evitar resultados errados na execução do programa.

Vamos ver um exemplo envolvendo operadores de atribuição:

#### exemplo4\_16.php

```
<html>
<body>
<?php
    $soma=0;
    $valor1 = 10;
    $valor2 = 20;
    $valor3 = 30;
    $soma += $valor1;      // $soma fica com 10
    $soma +=$valor2;       // $soma fica com 10+20 = 30
    $soma *= $valor3;      // $soma fica com 30*30 = 900
    $soma %= 100;          // $soma fica com 900%100 = 0
    echo $soma;
?>
</body>
</html>
```

Como você pode ver pelos comentários do programa, o valor que será mostrado na tela é zero. O último operador utilizado (%=) representa o resto da divisão (em outras linguagens é chamado de *MOD*). Dividindo 900 por 100 temos como resultado exato 9, portanto o resto da divisão é zero. No exemplo anterior utilizamos os operadores +=, \*= e %= para atribuir à variável \$soma resultados de operações realizadas entre a própria variável \$soma e outro operando. Perceba que o código fica bem mais claro e fácil de entender com o uso desses operadores de atribuição.

## Operadores lógicos

São aqueles que retornam o valor verdadeiro ou falso. Veja a tabela a seguir:

| Operador                        | Descrição  |
|---------------------------------|--|
| <code>!op1</code>               | Verdadeiro se <code>op1</code> for falso.                                |
| <code>op1 AND op2</code>        | Verdadeiro se <code>op1</code> E <code>op2</code> forem verdadeiros.     |
| <code>op1 OR op2</code>         | Verdadeiro se <code>op1</code> OU <code>op2</code> forem verdadeiros.    |
| <code>op1 XOR op2</code>        | Verdadeiro se só <code>op1</code> ou só <code>op2</code> for verdadeiro. |
| <code>op1 &amp;&amp; op2</code> | Verdadeiro se <code>op1</code> E <code>op2</code> forem verdadeiros.     |
| <code>op1    op2</code>         | Verdadeiro se <code>op1</code> OU <code>op2</code> forem verdadeiros.    |

Depois de observar essa tabela, você provavelmente está com dúvidas quanto à diferença entre os operadores AND e `&&`, e também os operadores OR e `||`. A diferença entre eles é a precedência dos operadores na avaliação de expressões. A precedência mais alta é dos operadores `&&` e `||`, enquanto os operadores AND e OR possuem precedência mais baixa. Por isso, tome muito cuidado com ao utilizá-los, pois podem gerar resultados diferentes dependendo da ordem em que forem colocados. Veremos mais diante um tópico especial sobre precedência de operadores.

Um exemplo típico no qual usamos operadores lógicos é o caso de testar se todos os campos obrigatórios de um formulário foram preenchidos. Suponha que tenhamos um formulário em que os campos nome, e-mail e CPF são obrigatórios. Certamente no programa que recebe os dados do formulário haveria um teste como o mostrado no trecho de programa a seguir:

### exemplo4\_17.php

```

<html>
<body>
<?php
...
if (empty($nome) OR empty($email) OR empty($cpf))
{
    echo "Você deve preencher os campos nome, e-mail e CPF!";
    exit;
}
...
P>
</body>
</html>

```

Nesse exemplo temos uma expressão sendo avaliada. A função `empty()`, que significa vazio em português, retorna verdadeiro se a variável estiver vazia, e retorna falso se houver algo na variável. Então estamos testando se a variável `$nome` está vazia ou a variável `$email` está vazia ou a variável `$cpf` está vazia. Se pelo menos uma das três estiver vazia o resultado será verdadeiro, e isso fará com que seja impressa a mensagem “Você deve preencher os campos nome, e-mail e CPF!”, e logo após o programa será encerrado por meio do comando `exit`.

Mais adiante veremos com detalhes os comandos condicionais, como o `if`, mas você já pode notar que os operadores lógicos têm relação direta com esses comandos, pois os comandos condicionais avaliam o resultado de expressões que utilizam os operadores lógicos.

Acompanhe as tabelas a seguir para ver os resultados gerados em cada um dos operadores de acordo com o tipo de expressão avaliada:

### Operador AND (E)

| Exp1 | Exp2 | Resultado |
|------|------|-----------|
| V    | V    | V         |
| V    | F    | F         |
| F    | V    | F         |
| F    | F    | F         |

### Operador OR (OU)

| Exp1 | Exp2 | Resultado |
|------|------|-----------|
| V    | V    | V         |
| V    | F    | V         |
| F    | V    | V         |
| F    | F    | F         |

### Operador XOR (OU exclusivo)

| Exp1 | Exp2 | Resultado |
|------|------|-----------|
| V    | V    | F         |
| V    | F    | V         |
| F    | V    | V         |
| F    | F    | F         |

### Operador ! (NOT)

O operador **!** é usado para negar o resultado de uma expressão lógica.

| Exp1           | Resultado      |
|----------------|----------------|
| V (verdadeiro) | F (falso)      |
| F (falso)      | V (verdadeiro) |

É importante lembrar que o resultado é sempre booleano, ou seja, é sempre verdadeiro ou falso.

### Operador ternário

É uma forma abreviada de usar o comando condicional *if*, que veremos mais adiante. Uma condição é avaliada, e, se ela for verdadeira, atribui-se um valor à variável, e se a condição for falsa atribui-se um outro valor. A sintaxe é a seguinte:

*cond ? exp1 : exp2*

Vamos ver um exemplo de uso desse operador, embora seja mais recomendado utilizar o comando condicional *if*, por ser mais simples de usar. Observe:

```
$nota = ($frequencia >= 0.75) ? ($nota+2) : ($nota-2);
```

Quando o PHP executar a linha anterior, se a variável *\$frequencia* possuir um valor maior ou igual a 0,75, a variável *\$nota* será aumentada de duas unidades, caso contrário haverá a diminuição de duas unidades. Escrevendo essa mesma operação por meio do comando *if*, temos o seguinte código:

```
<?php
    if ($frequencia >= 0.75)
        $nota = $nota+2;
    else
        $nota = $nota-2;
?>
```

Utilizando o comando *if* fica bem mais fácil entender qual é o objetivo do código, pois conseguimos fazer perfeitamente a distinção de qual é a condição e quais operações serão executadas após a avaliação desta. A utilização do operador ternário, apesar de utilizar um número menor de linhas para a operação, requer um pouco mais de prática para que nos acostumemos com sua notação.

## Precedência de operadores

Para evitar erros de lógica em seus programas é fundamental que você conheça a ordem utilizada pelo PHP para tratar os operadores. A tabela a seguir mostra a ordem decrescente de precedência que o PHP segue ao encontrar diversos operadores no programa:

| Operador                          | Descrição  |
|-----------------------------------|--|
| - ! ~ ++ --                       | Negativo, não-lógico, inversão de bits, incremento e decremento. |
| * / %                             | Multiplicação, divisão e resto da divisão.                       |
| + - .                             | Adição, subtração e concatenação.                                |
| << >>                             | Deslocamentos binários.  |
| > < >= <=                         | Maior, menor, maior ou igual, menor ou igual.                    |
| == != <>                          | Igual e diferente.   |
| &                                 | AND binário.   |
| ^                                 | XOR binário.   |
|                                   | OR binário.  |
| &&                                | AND lógico.  |
|                                   | OR lógico.   |
| :                                 | Operador ternário.   |
| = += -= *= /= %= &= ~= <<= >>= ^= | Operadores de atribuição.  |
| AND                               | AND lógico (de menor prioridade).                                |
| XOR                               | XOR lógico (de menor prioridade).                                |
| OR                                | OR lógico (de menor prioridade).                                 |

É importante lembrar que primeiro o PHP executará todas as operações que estiverem entre parênteses. Se dentro dos parênteses houver diversas operações, a precedência de operadores será utilizada para definir a ordem. Depois de resolver todas as operações que aparecem entre parênteses, o PHP resolverá o resto da expressão baseando-se na tabela anterior para determinar a ordem de avaliação dos operadores.

Quando houver operadores de mesma prioridade em uma expressão, e não existirem parênteses, o PHP resolverá a expressão da esquerda para a direita.

Observe o seguinte trecho de programa:

### exemplo4\_18.php

```
<?php
$numero = 5;
$resultado = 8 + 3 * 2 + ++$numero;
echo "$numero<br>";
echo $resultado;
?>
```

O resultado mostrado na tela será:

Observe que o operador `++` tem prioridade mais alta que os operadores `+` e `*`; por isso a primeira operação realizada pelo PHP foi o incremento do valor da variável `$num`. O segundo operador de maior prioridade no exemplo apresentado é o de multiplicação, portanto a segunda operação realizada foi  $3 * 2$ . Após essas duas operações, ficamos com a soma  $8 + 6 + 6$ . Note que temos dois operadores iguais (de adição), portanto, como têm a mesma prioridade, a expressão é avaliada da esquerda para a direita:  $8 + 6 = 14$ . E depois  $14 + 6 = 20$ , que foi o resultado mostrado na tela.

Como vimos na tabela anterior os operadores de multiplicação (`*`) e resto da divisão (`%`) têm a mesma precedência. Nesses casos é muito importante a utilização de parênteses para evitar resultados não desejados. Veja o exemplo a seguir:

#### exemplo4\_19.php

```
<?php  
$num = 7;  
$resultado = 8 * $num % 2;  
echo $resultado;  
?>
```

Após a execução desse programa será mostrado o resultado 0, pois primeiro foi realizada a multiplicação  $8 * 7 = 56$ . E depois foi calculado o resto da divisão de 56 por 2, que dá 0. Agora observe esse mesmo exemplo, mas utilizando os parênteses para determinar a ordem que deve ser utilizada para aplicar as operações:

#### exemplo4\_20.php

```
<?php  
$num = 7;  
$resultado = 8 * ($num % 2);  
echo $resultado;  
?>
```

Agora o resultado mostrado não será 0, e sim 8. Isso porque determinamos por meio dos parênteses que primeiro deveria ser realizada a operação do resto da divisão de 7 por 2, que resultou em 1. Depois disso foi feita a multiplicação desse valor por 8, resultando em  $8 * 1 = 8$ . Mediante esses dois exemplos, você pode perceber a importância da precedência dos operadores e do uso dos parênteses em expressões. Muitas vezes o uso incorreto dessas informações causa a exibição de dados incorretos na saída dos programas.

# Capítulo 5

## Estruturas de controle em PHP

Neste tópico veremos comandos que você utilizará para estruturar seus programas escritos em PHP. São comandos comuns à maioria das linguagens de programação, e o uso deles é fundamental para realizar decisões lógicas, testar se determinada expressão é verdadeira, repetir um bloco de comandos por um certo número de vezes ou até que uma condição seja atingida. Leia com atenção para ir se acostumando com a sintaxe desses comandos no PHP. Veremos com detalhes os comandos condicionais e os comandos de repetição.

Os comandos condicionais são:

- *if*
- *switch*

Os comandos de repetição são:

- *while*
- *do...while*
- *for*
- *foreach*

Além disso veremos o significado dos comandos *break* e *continue*, para controlar o fluxo dentro das repetições (loops).

### Comandos condicionais

Utilizando comandos condicionais temos a oportunidade de avaliar uma expressão e, dependendo do resultado obtido, executar um trecho de código diferente. Esses comandos são usados sempre que há necessidade de uma

tomada de decisão dentro de um programa. Por exemplo: se a nota for maior ou igual a 7, imprimir o valor aprovado, senão imprimir o valor reprovado. Os comandos condicionais são *if* e *switch*. Verificaremos exemplos envolvendo cada um deles:

## if

Comando que avalia uma expressão e, dependendo do resultado, é executado um conjunto diferente de instruções. O comando *if* pode possuir como complemento o *elseif* e/ou o *else*. Observe a sintaxe do comando *if*:

```
if ( exp1 )
    { bloco1 }
elseif ( exp2 )
    { bloco2 }
else
    { bloco3 }
```

Podemos ler essa sintaxe da seguinte maneira:

- se *exp1* for verdadeira, execute *bloco1*
- senão se *exp2* for verdadeira, execute *bloco2*
- senão execute *bloco3*

É importante lembrar que apenas um dos blocos será executado, e depois disso a execução continuará após o comando *if*.

Em português *if* significa “se” e o *else* significa “senão”. Dentro da construção do comando *if*, podem aparecer diversos *elseif*, cada um avaliando uma expressão. Quando a expressão avaliada pelo comando *if* resultar em valor verdadeiro (True), será executado o bloco de comandos definido logo a seguir, que aparece entre chaves ({}). Se não houver a delimitação do bloco por chaves, será executada apenas a primeira linha após o *if*. Após executar esse bloco de comandos, a execução do programa será desviada para o fim do comando *if*, e as demais expressões (contidas no *elseif* e no *else*) não serão avaliadas, pois o comando *if* escolhe apenas um entre vários conjuntos de instruções para execução.

Se a condição avaliada no comando *if* for falsa (False), o bloco de comandos seguinte não será executado, e será testada a expressão contida no primeiro

*elseif* (se houver). Se todas as expressões avaliadas (do *if* e do *elseif*) forem falsas, o bloco de comandos executado será aquele que vem após o *else*. Veja um exemplo no trecho de programa a seguir:

### exemplo5\_1.php

```
<?php
    $prova1 = 7;
    $prova2 = 5;
    $nota = ($prova1+$prova2) / 2;
    if ($nota<3)
        $desempenho = "PÉSSIMO";
    elseif ($nota<5)
        $desempenho = "RUIM";
    elseif ($nota<7)
        $desempenho = "MÉDIO";
    elseif ($nota<9)
        $desempenho = "BOM";
    else
        $desempenho = "EXCELENTE";
    echo "O seu desempenho foi $desempenho";
?>
```

Suponha que o aluno tenha tirado 7,0 na primeira prova, e 5,0 na segunda prova. No programa anterior seria feita a média aritmética para calcular a nota final:  $(7+5)/2 = 12/2 = 6$ . Portanto, a variável *\$nota* ficaria com o valor 6.

A expressão avaliada no *if* seria falsa, pois 6 não é menor que 3. A expressão avaliada no primeiro *elseif* também seria falsa, pois 6 não é menor que 5. No segundo *elseif*, a expressão avaliada retornaria verdadeiro, pois 6 é menor que 7, e isso faria com que fosse executada a linha:

```
$desempenho = "MÉDIO";
```

Perceba que não há chaves delimitando um bloco de comandos, e por isso a única linha executada foi a que aparecia logo após o *elseif*. Como a expressão avaliada resultou em verdadeiro, as demais não serão avaliadas, e o próximo comando executado será o *echo*, que imprimirá na tela o seguinte resultado:

```
O seu desempenho foi MÉDIO.
```

Note que o comando *else* não possui expressão a ser avaliada, pois este só será executado quando todas as outras expressões avaliadas resultarem em falso.

É importante lembrar também que não é obrigatório o uso de *elseif* e *else* com o comando *if*. O *if* pode aparecer sozinho, determinando se um bloco de instruções será executado ou não. Por exemplo:

### exemplo5\_2.php

```
<?php  
    if ($nota == 10)  
    {  
        echo "Parabéns! <br>";  
        echo "Você tirou a nota máxima!";  
    }  
?>
```

Nesse exemplo, se o valor da variável *\$nota* for igual a 10, será mostrada na tela a seguinte mensagem:

```
Parabéns!  
Você tirou a nota máxima!
```

Se o valor da variável *\$nota* não for igual a 10, esse bloco de comandos simplesmente não será executado, e a execução do programa seguirá normalmente. Poderíamos acrescentar um *else* ao comando, e imprimir outra mensagem caso o aluno não tivesse nota 10.

Se você já programou em outras linguagens, já deve ter usado alguma vez o *if* acompanhado de um *endif*, para determinar o fim do comando. Pois bem, há também uma sintaxe alternativa utilizando o *endif*:

```
if (exp1):  
    bloco1  
elseif (exp2):  
    bloco2  
else:  
    bloco3  
endif;
```

Na sintaxe apresentada não é necessário o uso de chaves, pois o PHP entende sendo um bloco de comandos desde os dois-pontos (:) até o próximo *elseif*, *else* ou *endif*.

## switch

Se você já programou na linguagem Pascal, deve se lembrar do comando *case*. O *switch* do PHP tem a mesma função. O comando *switch* é parecido com o *if*, pois ambos avaliam o valor de uma expressão para escolher qual bloco de instruções deve ser executado. Em algumas ocasiões, você tem uma mesma variável a ser testada com valores diferentes, e nesse caso é interessante utilizar o *switch*, que trabalha basicamente com o operador de igualdade, enquanto o *if* trabalha com qualquer tipo de operador.

O uso do comando *switch* torna o código um pouco mais organizado que o *if*, pois esse comando utiliza apenas uma cláusula (*case*), enquanto o *if* utiliza várias (*if*, *elseif*, *else* e as vezes *endif*). Veja a sintaxe do comando *switch*:

```
switch (operador)
{
    case valor1:
        <comandos>
        break;
    case valor2:
        <comandos>
        break;
    ...
    case valorN:
        <comandos>
        break;
    default:
        <comandos>
        break;
}
```

Perceba que após cada bloco de comandos deve ser utilizado o *break*, para que o comando *switch* seja encerrado e a execução continue após ele. Se não utilizarmos o *break* o PHP continuará a execução dentro do *switch*, avaliando as demais expressões.

Veja o exemplo a seguir, que mostra dois trechos de programa que fazem a mesma coisa, mas um deles utilizando *if* e o outro, o *switch*.

## Utilizando if

```
<?php  
...  
if ($numero == 0) {  
    echo "número vale 0";  
}  
elseif ($numero == 1) {  
    echo "número vale 1";  
}  
elseif ($numero == 2) {  
    echo "número vale 2";  
}  
...  
?>
```

## Utilizando switch

### exemplo5\_3.php

```
<?php  
...  
switch ($numero)  
{  
    case 0:  
        echo "número vale 0";  
        break;  
    case 1:  
        echo "número vale 1";  
        break;  
    case 2:  
        echo "número vale 2";  
        break;  
}
```

Observe que o comando `switch` é usado de maneira similar ao comando `if`, mas é mais eficiente para lidar com múltiplas alternativas. Ele é útil quando temos muitas alternativas e queremos evitar escrever muitos bloques `if`.

No comando `switch` temos também a opção `default`. Veja o exemplo a seguir:

Este exemplo mostra como usar o comando `default` no comando `switch`. Ele verifica se o valor de `$numero` é menor que zero ou maior que dois. Se for, ele imprime "número inválido". Caso contrário, ele imprime o valor do número.

### exemplo5\_4.php

```
<?php  
...  
switch ($opcao)  
{  
    case 's':  
        echo "Você escolheu a opção SIM";  
        break;  
    case 'n':  
        echo " Você escolheu a opção NÃO ";  
        break;  
    default:  
        echo " A opção digitada é inválida";  
        break;  
}  
?>
```

A opção *default* tem a mesma função da opção *else* no comando *if*. Se todas as expressões anteriores retornarem falso, será executado o bloco de comandos que aparece após o *default*. O uso do *default* não é obrigatório no comando *switch*.

## Comandos de repetição

São comandos utilizados para que um conjunto de instruções seja executado repetidamente por um número determinado de vezes, ou até que determinada condição seja atingida. Podemos, por exemplo, fazer repetições até que uma variável atinja determinado valor, ou até que uma variável seja diferente de um valor. Veremos com detalhes os comandos de repetição *while*, *do...while*, *for* e *foreach*.

### while

Traduzindo para o português, while significa enquanto. O comando *while* é composto por uma expressão e por um bloco de comandos. O comando avalia a expressão, e enquanto essa expressão retornar o valor verdadeiro, a execução do bloco de comandos em questão será repetida. Quando o valor retornado for falso, encerra-se o laço de repetição (*loop*), e a execução é transferida para o fim do comando *while*. Assim como no comando *if*, devemos utilizar chaves como delimitadores sempre que o bloco possuir mais de uma instrução para executar.

Veja a sintaxe do comando:

```
while (exp)
{
    comandos
}
```

Há também uma sintaxe alternativa, utilizando os dois-pontos:

```
while (exp):
    comandos
endwhile;
```

No segundo formato não há necessidade do uso de chaves. No primeiro formato as chaves devem ser utilizadas quando há mais de um comando a ser executado no bloco.

Devemos tomar cuidado para não colocarmos no comando *while* expressões que jamais se tornarão falsas, senão teremos um loop infinito, pois o PHP repetirá para sempre o bloco de instruções. Veja a seguir um exemplo de utilização do comando *while*:

#### exemplo5\_5.php

```
<?php
$cont = 1;
while ($cont<100)
{
    echo "O valor atual do contador é $cont <br>";
    $cont++;
}
?>
```

A execução desse programa resultará em 99 linhas mostradas na tela:

```
o valor atual do contador é 1
o valor atual do contador é 2
o valor atual do contador é 3
...
o valor atual do contador é 99
```

Quando a variável \$cont atingir o valor 100, a expressão retornará o valor falso, pois 100 não é menor que o próprio 100, e isso fará com que o loop seja encerrado. Se em vez de usarmos a expressão \$cont<100 usássemos a expressão \$cont!=0 (diferente de zero), teríamos um laço infinito, pois a cada repetição do laço o valor de \$cont seria incrementado e, desse modo, ele nunca chegaria ao valor zero, que é a condição necessária para o encerramento do loop.

## do...while

A diferença entre o *while* e o *do...while* é que o *while* avalia a expressão no início do laço, e o *do...while* avalia a expressão no final do laço. Portanto, você já pode perceber que utilizando *do...while* o laço será executado pelo menos uma vez, e utilizando somente *while* o laço pode não ser executado, caso a expressão avaliada retorne falso na primeira avaliação. Se você já programou em outras linguagens, deve conhecer o comando *repeat...until*, que tem a mesma função do comando *do...while*, mas com uma diferença: no comando *repeat...until* o laço é encerrado quando a expressão retorna verdadeiro, e no comando *do...while* o laço é encerrado quando a expressão avaliada é falsa. A sintaxe do comando é a seguinte:

```
do
{
    comandos
} while (exp);
```

Veja um exemplo:

### exemplo5\_6.php

```
<?php
$numero = 1;
do
{
    echo "O valor atual de número é $numero <br>";
    $numero++;
} while ($numero<4);
?>
```

O resultado gerado pela execução desse programa será:

```
O valor atual de número é 1
O valor atual de número é 2
O valor atual de número é 3
```

## for

Utilizamos o comando *for* quando queremos executar um conjunto de instruções um número determinado de vezes. É um comando muito útil que pode ser usado, por exemplo, para imprimir todos os elementos de um array, ou todos os registros retornados de uma consulta a um banco de dados. A sintaxe do comando *for* é a seguinte:

```
for ( inicialização ; condição ; operador )  
{
```

*comandos*

```
}
```

Assim como no *if* e no *while*, há uma sintaxe alternativa para o *for*:

```
for ( inicialização ; condição ; operador ):  
    comandos  
endfor;
```

Como *inicialização* geralmente determinamos o valor inicial da variável que controlará o loop. O parâmetro de inicialização poderia ser `$cont=0`. No segundo parâmetro devemos colocar a condição que deve ser atingida para que o loop continue. Se quiséssemos executar o loop 20 vezes, o valor do parâmetro de condição seria `$cont<20`. Quando essa condição retornar o valor falso, o loop é encerrado. O último parâmetro geralmente é usado para atualizar o valor da variável de controle do loop, fazendo um incremento ou um decremento (por exemplo, `$cont++`). Ao final de cada iteração do loop, o valor da variável de controle é atualizado automaticamente, dependendo do terceiro parâmetro que você definiu quando utilizou o comando *for*. Veja alguns exemplos de utilização do comando:

### exemplo5\_7.php

```
<?php  
for($cont=0 ; $cont<10 ; $cont++)  
{  
    echo "A variável \$cont vale $cont";  
    echo "<br>";  
}  
?>
```

O resultado gerado pela execução desse programa será o seguinte:

```
A variável $cont vale 0  
A variável $cont vale 1  
...  
A variável $cont vale 9
```

No comando *while* muitas vezes temos de incrementar a variável de controle ao final do laço. No comando *for* isso é feito de forma automática quando definimos o terceiro parâmetro do comando (*\$cont++*). Ao fim de cada laço a variável *\$cont* será incrementada. Veja outro exemplo, agora utilizando o decremento:

#### exemplo5\_8.php

```
<html>
<body>
<?php
    echo "Estou fazendo uma contagem regressiva: <br>";
    for($i=15 ; $i>=0 ; $i--)
    {
        echo $i . ", ";
    }
    echo "... FIM!";
?>
</body>
</html>
```

Executando essa página em seu browser você verá o seguinte resultado:

```
Estou fazendo uma contagem regressiva:
15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 ... FIM!
```

Podemos fazer também o aninhamento de loops. Isso é muito útil quando queremos, por exemplo, imprimir na tela todos os elementos de um array bidimensional. Suponha que temos uma array bidimensional de dimensões 2 x 2. Poderíamos imprimir todos os elementos desse array utilizando duas vezes o comando *for* de forma aninhada, como mostra o exemplo a seguir:

#### exemplo5\_9.php

```
<?php
    $vetor[0][0]= "elemento00";
    $vetor[0][1]= "elemento01";
    $vetor[1][0]= "elemento10";
    $vetor[1][1]= "elemento11";
    for($i=0 ; $i<2 ; $i++)
    {
        for($k=0 ; $k<2 ; $k++)
        {
            echo $vetor[$i][$k];
        }
    }
?>
```

```
<?php
    $vetor = array();
    for( $i=0 ; $i<10 ; $i++ )
        $vetor[$i] = $i;
    for( $i=0 ; $i<10 ; $i++ )
    {
        echo "O elemento da posição $i,$k é ";
        echo $vetor[$i][$k];
        echo "<br>";
    }
    echo "<br><br>Agora vou mudar o valor de $k para 5 e voltar a imprimir o vetor:
    <br><br>
    <?>
```

Apesar de não ser muito comum, podemos inicializar mais de uma variável no parâmetro de inicialização do comando *for*, e também podemos ter mais de um operador no terceiro parâmetro. Para isso basta utilizarmos a vírgula, conforme mostra o exemplo a seguir:

```
<?php
    for( $i=0, $k=10 ; $i<10 ; $i++, $k- )
    {
        echo "\$i vale $i e \$k vale $k";
        if ($i==$k)
            { echo " (os valores são iguais!)"; }
        echo "<br>";
    }
    <?>
```

Salve esse programa como *prog8.php* e execute em seu navegador. Você verá a seguinte tela:

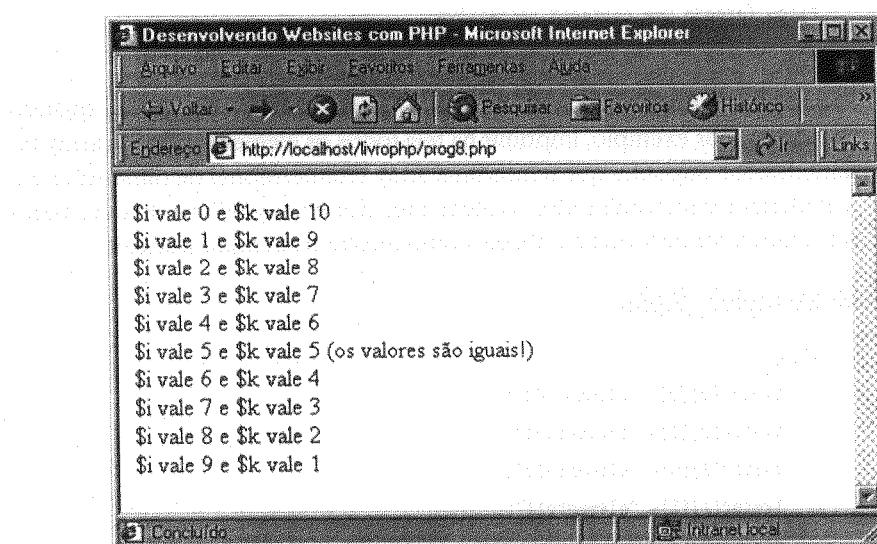


Figura 6.1 – Resultado gerado pelo programa *prog8.php*.

Ao final de cada iteração do laço, o valor das duas variáveis (*\$i* e *\$k*) é atualizado.

## foreach

O comando *foreach* funciona apenas na versão 4 do PHP. O PHP3 não oferece esse suporte. O *foreach* nos oferece uma maneira mais fácil de “navegar” entre os elementos de um array. Observe as duas sintaxes possíveis:

```
foreach ($nome_array as $elemento)
```

```
{
```

comandos

```
}
```

ou

```
foreach ($nome_array as $chave => $valor)
```

```
{
```

comandos

```
}
```

A primeira forma vai do primeiro ao último índice do array definido na variável *\$nome\_array*, e a cada iteração o valor do elemento correto do array é atribuído à variável *\$elemento*, e o ponteiro interno do array é avançado. Dessa forma, podemos trabalhar com todos os valores do array utilizando apenas a variável *\$elemento*.

A segunda forma faz a mesma coisa, mas com uma diferença: além do valor do elemento correto do array ser atribuído à variável *\$elemento*, a chave (ou índice) do elemento atual é atribuído à variável *\$chave*. Acompanhe o exemplo a seguir:

### exemplo5\_10.php

```
<?php
$vetor = array (1, 2, 3, 4);
foreach ($vetor as $v)
{
    print "O valor atual do vetor é $v. <br>";
}
$a = array ("um" => 1, "dois" => 2, "tres" => 3 );
foreach($a as $chave => $valor) {
    print "\$a[$chave] => $valor.<br>";
}
?>
```

O programa apresentado mostrará na tela todos os valores do array `$vetor`, e depois mostrará todos as chaves e valores do array `$a`. O segundo `foreach` desse exemplo mostrará o seguinte:

```
$a [um] => 1.  
$a [dois] => 2.  
$a [tres] => 3.
```

## Controlando o fluxo de execução

Existem comandos que podem ser usados em conjunto com essas estruturas de controle que acabamos de ver. Esses comandos são o `break` e o `continue`.

Vamos ver qual é a utilidade de cada um deles:

### `break`

O `break` termina a execução do comando atual, que pode ser um `if`, `for`, `while`, ou `switch`. Quando o `break` é encontrado dentro de algumas dessas estruturas, o fluxo de execução passa para o primeiro comando após o término dessa estrutura. É um recurso que podemos utilizar para forçar a saída de um laço ou de um comando condicional. Acompanhe o exemplo a seguir:

#### exemplo5\_11.php

```
<?php  
...  
$k = 1;  
while ($k < 20)  
{  
    if ($vetor[$k] == "sair")  
    { . . . break; . . . }  
    echo $vetor[$k] . "<br>";  
    $k++;  
}  
?>
```

O programa apresentado mostrará na tela os valores de um vetor qualquer. O laço poderá ser encerrado de duas formas:

- 1) se o valor de `$k` for maior ou igual a 20;
- 2) se algum elemento do vetor possuir o valor “sair”.

O comando *break* também aceita um argumento numérico opcional, que diz o número de estruturas que devem ser finalizadas. Portanto, utilizar o comando *break* equivale a utilizar *break 1*, pois quando não há um parâmetro definido temos a finalização de apenas uma estrutura. Veja a seguir um exemplo com a utilização desse parâmetro:

### exemplo5\_12.php

```
<?php
$i = 0;
$k = 0;
while ($k < 10)
{
    $i++;
    $k++;
    while ($i < 20)
    {
        if($i==10)
        {
            echo "Encerrando o primeiro while...<br>";
            break ;
        }
        echo " Essa linha não vai ser impressa!!!";
    }
    elseif ($i==15)
    {
        echo " Encerrando os dois whiles...";
        break 2;
    }
    $i++;
}
?>
```

Nesse exemplo, quando a variável *\$i* chegar ao valor 10, a execução passa para dentro do *if*, e quando o *break* é executado termina imediatamente a execução do laço atual. O comando *echo* que é utilizado após o *break* não será executado. Quando o valor da variável *\$i* chegar a 15, o comando executado será o *break 2*. Esse comando fará com que sejam finalizadas as duas estruturas de repetição, e a execução irá continuar após os dois *whiles*. Esse é o objetivo desse parâmetro opcional do *break*.

## continue

O comando *continue* é utilizado dentro dos comandos de repetição para ignorar o restante das instruções pertencentes ao laço corrente, e ir para a próxima iteração (voltando ao início do laço). Veja o exemplo a seguir:

### exemplo5\_13.php

```
<?php
    $vetor = array (1, 3, 5, 8, 11, 12, 15, 20);
    for($i=0 ; $i<sizeof($vetor) ; $i++)
    {
        if ($vetor[$i] % 2 != 0)      // é ímpar
        { continue; }
        $num_par = $vetor[$i];
        echo "O número $num_par é par. <br>";
    }
?>
```

A função *sizeof* retorna o número de elementos do array *\$vetor* (nesse caso retornará o valor 8). Esse programa utiliza um comando *for* para passar por todos os elementos desse array, mas somente os números pares são mostrados na tela. Sabemos que os números pares quando divididos por 2 apresentam resto 0, pois a divisão é exata. O programa testa por meio do comando *if* se o resto da divisão é diferente de zero, ou seja, se o número é ímpar. Ao encontrar um valor ímpar, é executado o comando *continue*, e os comandos seguintes não são executados. Com isso a execução retorna ao início do laço, e o próximo elemento do vetor é acessado. Dessa forma, ao final do laço teremos somente valores pares mostrados na tela, pois o comando *continue* evitou que os ímpares também fossem mostrados.

Assim como o comando *break*, o comando *continue* também aceita um argumento numérico opcional, que diz o número de níveis que devem ser reiniciados, ignorando os comandos seguintes. Se utilizarmos, por exemplo, o comando *continue 2* dentro de dois *whiles* aninhados, a execução volta ao início do primeiro *while*.

Capítulo 6: Funções e classes

# Capítulo 6

## Funções e classes

### Definição de função

As funções (functions) são muito úteis para deixar o código dos programas mais organizado e mais modular, e além disso elas nos pouparam da tarefa de ter de repetir determinado código toda vez que precisamos realizar a mesma tarefa.

Funções são trechos de código que podem realizar qualquer tipo de tarefa, como, por exemplo, somar dois números, testar se o valor de uma variável é válido, verificar se um número de CPF foi digitado corretamente, transformar uma string para letras maiúsculas etc. Funções podem ser ativadas a qualquer momento da execução do programa. A sintaxe para a construção de uma função é a seguinte:

```
function nome_função (arg1, arg2, arg3 ..., argn )  
{  
    comandos  
    [ return <expressão> ]  
}
```

Devemos substituir *nome\_função* por um identificador, que deve ser um nome único. O identificador não pode iniciar com número, e além disso não é permitido a utilização de caracteres como ponto, vírgula, espaço etc. Um caractere que é bastante utilizado para formar nomes de funções é o \_ (sublinhado). Definida uma função, podem haver diversas chamadas para ela, que será referenciada por meio do identificador escolhido.

Uma função pode, mediante uma chamada, receber diversos valores que chamamos de argumentos, que na sintaxe estão representados por *arg1, arg2, arg3 ... argn*. Esses argumentos são valores recebidos no momento da chamada, e

que de alguma forma serão processados no código da função. É opcional a utilização dos argumentos. Se a função não os recebe, devemos simplesmente escrever `function nome_função ()`.

Sempre que o PHP encontrar uma chamada para uma função, a execução do programa é interrompida e o fluxo de execução passa para o início do código da função. Após o término da função, a execução volta ao programa principal no mesmo ponto em que foi feita a chamada, e o próximo comando é executado.

Observe que na sintaxe de uma função existe um comando opcional, que é o `return`. Esse comando existe porque podemos fazer com que a função retorne um valor para o ponto em que foi chamada. Utilizamos o `return` em duas situações:

- 1) quando queremos atribuir o valor retornado a uma variável;
- 2) quando queremos testar (comandos condicionais) o valor de retorno de uma função.

A função pode simplesmente executar uma tarefa e não retornar nenhum valor, como, por exemplo, uma função que recebe dois valores realiza operações aritméticas com eles e mostra o resultado na tela. Esse tipo de função não retorna nenhum valor.

## Como criar uma função

Veja no exemplo a seguir como criar uma função e incluir uma chamada para ela no programa principal:

### exemplo6\_1.php

```
<?php  
    function soma_valores ($valor1 , $valor2 , $valor3)  
    {  
        $soma = $valor1 + $valor2 + $valor3;  
        echo "A soma dos valores $valor1, $valor2 e $valor3 ";  
        echo "é $soma";  
    }  
    $n1 = 10;  
    $n2 = 20;  
    $n3 = 50;  
    soma_valores ($n1, $n2, $n3);  
?>
```

Note que as variáveis que são passadas como parâmetros não precisam ter o mesmo nome dos argumentos definidos na função. Isso ocorre porque no momento da chamada da função *soma\_valores*, a variável *\$valor1* receberá o valor do primeiro argumento, a variável *\$valor2* receberá o valor do segundo e a variável *\$valor3* receberá o valor do terceiro.

A função apresentada não utiliza o comando *return*, pois seu objetivo é apenas somar três números e mostrar o valor na tela.

## Utilizando o comando *return* em uma função

O comando *return* devolve para o ponto de chamada da função o valor da variável ou expressão que aparece imediatamente após esse comando. Desse modo, podemos atribuir a uma variável (com o operador *=*) o valor retornado de uma função.

Suponha que queremos imprimir uma variável do tipo *string*, mas com todas as letras maiúsculas. O PHP possui a função *strtoupper*, que transforma uma string em letras maiúsculas. Porém, dependendo da versão, esse comando não transforma caracteres acentuados. Uma opção para resolver esse problema seria a criação de uma função que recebe como argumento uma string e faz a transformação de todos os caracteres dessa string em maiúsculos. Acompanhe no exemplo a seguir como seria essa função:

### exemplo6\_2.php

```
<?php  
function maiusculo($string) {  
    $string = strtoupper ($string);  
    $string = str_replace ("á", "Ã", $string);  
    $string = str_replace ("é", "Ë", $string);  
    $string = str_replace ("í", "Ï", $string);  
    $string = str_replace ("ô", "Ô", $string);  
    $string = str_replace ("ú", "Ù", $string);  
    $string = str_replace ("â", "Ã", $string);  
    $string = str_replace ("ê", "Ë", $string);  
    $string = str_replace ("õ", "Ô", $string);  
    $string = str_replace ("í", "Ï", $string);  
    $string = str_replace ("õ", "Ù", $string);  
    $string = str_replace ("â", "Ã", $string);  
    $string = str_replace ("õ", "Ô", $string);  
    $string = str_replace ("ç", "Ç", $string);
```

```
<?php
    $string = str_replace ("à", "A", $string);
    return $string;
}
$nome = "José Antônio";
$nome_m = maiusculo ($nome);

echo "O nome do rapaz é $nome_m";
?>
```

A execução do programa apresentado mostrará na tela o seguinte resultado:

```
O nome do rapaz é JOSÉ ANTÔNIO
```

Observe que o retorno da função foi atribuído à variável `$nome_m`, e depois esse valor foi mostrado na tela por meio do comando `echo`. No início da função utilizamos o comando `strtoupper` para transformar os caracteres não acentuados em maiúsculos. Para transformar os caracteres acentuados utilizamos o comando `str_replace`, que faz a substituição de determinada parte da string por outro dentro de uma variável. Existem várias outras funções que trabalham com strings, basta você consultar o manual do PHP para conhecê-las. Neste livro veremos as funções utilizadas com maior freqüência. Observe a utilização do comando `return` dentro da função. Se ele não fosse colocado, a função não retornaria nenhum valor, e todas as alterações feitas durante sua execução seriam perdidas.

O retorno de uma função não necessariamente deve ser atribuído a uma variável. Ele pode ser colocado diretamente na saída de um comando de impressão. Veja um exemplo no programa a seguir:

### exemplo\_3.php

```
<?php
function triplo ($numero)
{
    $x = $numero * 3;
    return $x;
}
$valor = 5;
echo "O triplo de $valor é " . triplo($valor);
?>
```

Uma função pode também, em vez de retornar um único valor, retornar um array contendo vários elementos. Veja a seguir um exemplo:

### exemplo6\_4.php

```
<?php
    function clubes ( )
    {
        $clubes [ ] = "Grêmio";
        $clubes [ ] = "Palmeiras";
        $clubes [ ] = "Flamengo";
        $clubes [ ] = "Atlético Mineiro";
        $clubes [ ] = "Bahia";
        return $clubes;
    }
    // início do programa principal
    $nomes = clubes( );
    for ($i=0 ; $i < sizeof($nomes) ; $i++)
    {
        echo "\$nomes[$i] vale $nomes[$i] <br>";
    }
?>
```

Após a execução desse programa, o resultado será o seguinte:

```
$nomes[0] vale Grêmio
$nomes[1] vale Palmeiras
$nomes[2] vale Flamengo
$nomes[3] vale Atlético Mineiro
$nomes[4] vale Bahia
```

Observe que a função *clubes* não recebe nenhum argumento, ela apenas é ativada e retorna um array para o ponto em que foi chamada.

Veremos agora um exemplo de função envolvendo a data atual. Para isso utilizaremos as funções *time* e *getdate* do PHP. A função *time* retorna o tempo corrente em número de segundos desde 1º de janeiro de 1970 (padrão Unix), e a função *getdate* transforma esse tempo em um array com o dia, mês, ano e outras informações sobre a data atual:

### exemplo6\_5.php

```
<?php
    function retorna_data ( )
    {
        $agora = time();
        $data = getdate($agora);
```

```

if($data["wday"]==0) { echo "Domingo, "; }
elseif($data["wday"]==1) { echo "Segunda-feira, "; }
elseif($data["wday"]==2) { echo "Terça-feira, "; }
elseif($data["wday"]==3) { echo "Quarta-feira, "; }
elseif($data["wday"]==4) { echo "Quinta-feira, "; }
elseif($data["wday"]==5) { echo "Sexta-feira, "; }
elseif($data["wday"]==6) { echo "Sábado, "; }
if($data["mon"]==1) { $mes="Janeiro"; }
elseif($data["mon"]==2) { $mes="Fevereiro"; }
elseif($data["mon"]==3) { $mes="Março"; }
elseif($data["mon"]==4) { $mes="Abril"; }
elseif($data["mon"]==5) { $mes="Maio"; }
elseif($data["mon"]==6) { $mes="Junho"; }
elseif($data["mon"]==7) { $mes="Julho"; }
elseif($data["mon"]==8) { $mes="Agosto"; }
elseif($data["mon"]==9) { $mes="Setembro"; }
elseif($data["mon"]==10) { $mes="Outubro"; }
elseif($data["mon"]==11) { $mes="Novembro"; }
elseif($data["mon"]==12) { $mes="Dezembro"; }
$data_atual = $data["mday"]." de ".$mes." de ".$data["year"];
return $data_atual;
}
$hoje = retorna_data();
echo $hoje;
?>

```

A execução desse programa imprimirá na tela o dia atual da semana, o dia do mês, o nome do mês e o ano atual. Por exemplo:

Quarta-feira, 21 de Março de 2001

## Utilizando funções para verificar um CPF

Outro exemplo em que as funções são bastante úteis é para verificar se um número de CPF foi digitado corretamente. Para isso existem diversas funções que fazem essa verificação com base no dígito verificador do CPF. Você encontrará facilmente alguma se fizer uma busca na Internet. A seguir temos um exemplo de função que testa o número do CPF:

**exemplo\_6.php**

```
<?php

function cpf_errado($cpf) {
    $erro = false;
    $aux_cpf = "";
    for($j=0;$j<strlen($cpf);$j++)
        if(substr($cpf,$j,1)>="0" AND substr($cpf,$j,1)<="9")
            $aux_cpf .= substr($cpf,$j,1);
    if(strlen($aux_cpf)!=11)
        $erro = true;
    else {
        $cpf1 = $aux_cpf;
        $cpf2 = substr($cpf,-2);
        $controle = "";
        $start = 2;
        $end = 10;
        for($i=1;$i<=2;$i++) {
            $soma = 0;
            for($j=$start;$j<=$end;$j++)
                $soma += substr($cpf1,($j-$i-1),1)*($end+1+$i-$j);
            if($i==2)
                $soma += $digito * 2;
            $digito = ($soma * 10) % 11;
            if($digito==10)
                $digito = 0;
            $controle .= $digito;
            $start = 3;
            $end = 11;
        }
        if($controle!=$cpf2)
            $erro = true;
    }
    return $erro;
}

?>
```

Um exemplo de utilização da função anterior seria o seguinte:

### exemplo\_7.php

```
<?php  
...  
if (cpf_errado($cpf))  
{  
    echo "O CPF digitado é inválido";  
    exit;  
}  
?>
```

Se o CPF estiver correto, a função `cpf_errado` retornará o valor `false` (falso), e os comandos dentro do `if` não serão executados. Se o CPF digitado estiver incorreto, será retornado o valor `true` (verdadeiro), fazendo com que o bloco de comandos do `if` seja executado. Então será mostrada a mensagem “O CPF digitado é inválido”, e a execução do programa será finalizada pelo comando `exit`.

Imagine se não existisse o conceito de funções e você precisasse fazer a validação de dois CPFs. O que fazer? Repetir duas vezes todo esse código para fazer uma verificação individual de cada CPF? Agora você já pôde perceber quais são os fatores que tornam importantes a utilização de funções em seus programas: a modularidade, a organização e a reutilização de código.

Além de funções que verificam a validade de um número de CPF, também existem várias que verificam a validade do CGC de uma empresa. Na Web você também encontra essas funções prontas para download em diversos sites, por isso não é necessário que você perca tempo fazendo essa programação.

## Passagem de parâmetros: valor e referência

Quando passamos uma variável como argumento para uma função, por padrão estamos passando apenas o valor dela, e isso faz com que as alterações realizadas dentro da função não se reflitam sobre a variável quando terminar a execução da função. Ou seja, se uma função recebe como argumento a variável `$teste`, que possui o valor 5, quando terminar a execução da função a variável `$teste` continuará valendo 5, independentemente das alterações feitas dentro da função. Chamamos esse processo de passagem de parâmetros por valor, pois estamos passando apenas o valor da variável para ser utilizado na função.

Mas existem situações em que queremos que a variável a ser passada como argumento seja alterada conforme as alterações feitas durante a execução da função. Esse processo é chamado de passagem de parâmetros por referência. Esse tipo de passagem de parâmetros requer que seja colocado o símbolo & antes do nome da variável. Vamos ver alguns exemplos para entender melhor o funcionamento dos dois modos:

#### exemplo6\_8.php

```
<?php
    function dobro ($valor)
    {
        $valor = 2 * $valor;
    }
    function duplica (&$valor)
    {
        $valor = 2 * $valor;
    }
    $valor = 5;
    dobro ($valor);
    echo $valor . "<br>";
    duplica ($valor);
    echo $valor;
?

```

O resultado mostrado após a execução do programa será:

5

10

Isso ocorreu porque quando chamamos a função *dobro*, o valor calculado dentro da função se perdeu quando a execução retornou ao programa principal. Quando chamamos a função *duplica*, a alteração feita dentro dessa função fez efeito na variável *\$valor*, mantendo o novo valor mesmo quando ocorreu o término da função; ou seja, na função *dobro* fizemos uma passagem de parâmetros por valor, e na função *duplica* fizemos uma passagem de parâmetros por referência.

Quando utilizamos a passagem por referência, o que o PHP faz é simplesmente criar uma referência para a variável original, e isso faz com que as alterações sejam feitas diretamente na própria variável. Na passagem por valor é feita uma cópia do conteúdo da variável, e as alterações são feitas nessa cópia, não se refletindo na variável original.

Ainda sobre passagem de parâmetros, há uma característica do PHP que permite a definição de valores-padrão. Se uma função possui determinado parâmetro e no momento da chamada esse parâmetro não é enviado, podemos utilizar valores-padrão. Para definir esses valores, basta colocar um operador de atribuição após o parâmetro definido na função, seguindo pelo valor que deve ser considerado o padrão. Veja um exemplo:

### exemplo\_9.php

```
<?php
    function teste ($time , $titulo = "Campeão Mundial")
    {
        echo "O $time é $titulo <br>";
    }
    teste ("Flamengo" , "Campeão Carioca");
    teste ("Atlético" , "Campeão Mineiro");
    teste ("Grêmio");
?>
```

Após a execução desse programa o resultado será:

```
O Flamengo é Campeão Carioca
O Atlético é Campeão Mineiro
O Grêmio é Campeão Mundial
```

Nas duas primeiras chamadas à função *teste* foram passados os dois parâmetros necessários, mas na terceira chamada foi enviado apenas o primeiro parâmetro, e isso fez com que o valor-padrão fosse assumido para o segundo parâmetro.

Uma observação importante sobre a utilização de valores-padrão é que esses devem ser sempre os últimos parâmetros definidos na função. Nunca devemos colocá-los na frente de outros parâmetros que não possuem um valor-padrão. Se uma função, por exemplo, possui três parâmetros, e um deles possui um valor-padrão, não podemos colocá-lo como primeiro ou segundo parâmetro.

Considere a seguinte função:

```
function nada ($a=10 , $b, $c)
{
}
```

Essa definição está errada, pois como o envio do parâmetro `$a` é opcional, poderíamos fazer a seguinte chamada:

```
nada(1, $);
```

Isso causaria um erro, porque o PHP interpretará os dois parâmetros passados como `$a` e `$b`, e ocorreria um erro de execução devido à falta do parâmetro `$c`. Portanto, o modo correto de definir a função seria:

```
function nada ($b, $c, $a=10)
```

Dessa forma, poderíamos fazer a chamada `nada(1, 5)`, fazendo com que `$a`, `$b` e `$c` recebam os valores 10, 1 e 5, respectivamente.

## Funções recursivas

Chamamos de funções recursivas aquelas funções que chamam a elas mesmas. Veja um exemplo simples desse tipo de função:

### exemplo6\_10.php

```
<?php  
function teste ($valor)  
{  
    if ($valor!=0)  
    {  
        echo "Foi chamada a função teste passando o valor $valor <br>";  
        teste ($valor-1);  
    }  
}  
teste (7);
```

O resultado gerado pela execução do programa será o seguinte:

```
Foi chamada a função teste passando o valor 7  
Foi chamada a função teste passando o valor 6  
Foi chamada a função teste passando o valor 5  
Foi chamada a função teste passando o valor 4  
Foi chamada a função teste passando o valor 3  
Foi chamada a função teste passando o valor 2  
Foi chamada a função teste passando o valor 1
```

O que ocorreu foi que a cada chamada da função *teste*, o valor passado era mostrado na tela e logo em seguida era feita uma nova chamada para essa função, mas passando o valor antecedente. Com isso fomos formando uma pilha de chamadas à função *teste*. Quando a variável *\$valor* atingiu o valor zero, ocorreu o desempilhamento de chamadas, e as funções em aberto foram terminando, cada uma devolvendo o controle para quem a chamou.

Umas das funções recursivas mais conhecidas é aquela que faz o cálculo do fatorial de um número. Se você não lembra bem o que é o fatorial, veja alguns exemplos:

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

$$3! = 3 \cdot 2 \cdot 1 = 6$$

$$2! = 2 \cdot 1 = 2$$

$$1! = 1$$

$$0! = 1$$

O fatorial de um número *n* é calculado por meio de *n* multiplicações, envolvendo o próprio número e os antecessores a ele. Por convenção, o fatorial do número 0 vale 1. A função *fatorial* poderia ser definida da seguinte maneira:

### exemplo6\_11.php

```
<?php
    function fatorial ($numero)
    {
        if ($numero<0)
            { return -1; }
        if($numero<=1)
            { return 1; }
        return $numero*fatorial($numero-1);
    }
    echo "O fatorial de 3 é " . fatorial(3);
    echo "<br>O fatorial de 4 é " . fatorial(4);
    echo "<br>O fatorial de 5 é " . fatorial(5);
?>
```

O primeiro teste realizado testa se o número passado como argumento não é negativo, pois não existe fatorial de números negativos. Caso isso ocorra, a função retorna o valor -1. Se o número passado como argumento for maior que 1, é feita a multiplicação do valor corrente pelo fatorial do número antecessor, que se obtém fazendo uma nova chamada para a função *fatorial*. Aí está a recursividade!

As chamadas serão empilhadas, e quando o valor passado como argumento se tornar menor ou igual a 1 é feito o caminho de volta, ou seja, as chamadas são retiradas da pilha e os valores calculados são retornados, até chegar no primeiro ponto do programa que realizou a chamada, retornando o valor do fatorial do número. Na chamada do fatorial de 5, por exemplo, foram feitas as seguintes chamadas:

| Chamada        | Retorno          |
|----------------|------------------|
| 1 <sup>a</sup> | 5 * factorial(4) |
| 2 <sup>a</sup> | 4 * factorial(3) |
| 3 <sup>a</sup> | 3 * factorial(2) |
| 4 <sup>a</sup> | 2 * factorial(1) |
| 5 <sup>a</sup> | 1                |

O valor de retorno da 5<sup>a</sup> chamada é utilizado na 4<sup>a</sup>. O valor de retorno da 4<sup>a</sup> chamada é utilizado na 3<sup>a</sup>, e assim por diante.

## Reutilizando funções

Imagine que seu site possui cerca de 50 páginas, e existe uma função que você utiliza em todas elas. Você não precisa repetir o código dessa função em cada uma das páginas. Se você fizesse isso, existiriam 50 funções iguais em seu site, e o pior, se fosse necessário fazer alguma alteração na função, você teria de alterar em todas as páginas.

Um exemplo que pode ser citado é a exibição da data atual. Se você quisesse trocar o formato de exibição da data, seria interessante alterar apenas um arquivo, e assim as demais páginas seriam também automaticamente alteradas.

Para que possamos reutilizar uma função em diversas páginas, existe o comando *include*, que veremos com detalhes mais adiante.

## Definição de classe

Se você já programou em alguma linguagem orientada a objetos, você deve ter uma noção desse conceito. Classes são agrupamentos de variáveis e funções. Uma classe é utilizada para descrever um objeto. Um objeto poderá utilizar todas as variáveis e funções existentes em sua classe. Veja o exemplo de uma classe que podemos definir para trabalhar com as características de um apartamento.

### Classe apartamento

Propriedades:

- numero\_quartos
- numero\_banheiros
- preco\_aluguel
- preco\_venda
- tamanho
- localizacao

Essas propriedades são atributos da classe apartamento, que poderão ser utilizados quando criarmos um objeto para essa classe.

## Como criar uma classe

A sintaxe é a seguinte:

```
class nome_classe
{
    <procedimentos>
}
```

Uma classe pode conter várias funções e variáveis. Para criar variáveis dentro de uma classe, usamos a instrução *var*. Dentro das classes, podemos também utilizar a variável *\$this*, que é uma variável interna do PHP responsável por referenciar o objeto utilizado.

Para criar um objeto de determinada classe, devemos atribuir à variável que representará o objeto a instrução *new* seguida do nome da classe. Para acessar valores de um array dentro de uma classe, devemos utilizar o símbolo *->*, seguido do nome do array com chave associativa desejada.

Veremos agora um exemplo de uma classe chamada *Loja*, que faz o controle dos artigos disponíveis, possibilitando inclusive controlar a inserção ou remoção de novos artigos por meio de funções definidas internamente. Veja o exemplo a seguir:

#### exemplo6\_12.php

```
<?php
class Loja
{
```

```
var $itens;
function adiciona ($codigo, $quantidade) {
    if(isset($this->itens[$codigo]))
        $this->itens[$codigo] += $quantidade;
    else
        $this->itens[$codigo] = $quantidade;
}
function remove ($codigo, $quantidade)
{
    if ($this->itens[$codigo] > $quantidade)
    {
        $this->itens[$codigo] -= $quantidade;
        return true;
    }
    else
    {
        return false;
    }
}
$estoque = new Loja;
?>
```

A classe *Loja* possui como variável o array *itens* que foi declarado no início com o uso da instrução *var*. Além disso, existem duas funções que têm por objetivo acrescentar ou remover itens do estoque. O objeto é criado no momento que fazemos a atribuição de *new Loja* à variável *\$estoque*. A partir desse momento o objeto *\$estoque* já pode acessar as variáveis e funções disponíveis na classe. Veja um exemplo em que esse acesso é feito:

### exemplo6\_13.php

```
<?php
...
$estoque->adiciona("bermuda",2);
$estoque->adiciona("camiseta",3);
echo "A loja já possui " . $estoque->itens["bermuda"] . " bermudas.<br>";
echo "A loja já possui " . $estoque->itens["camiseta"] . " camisetas.<br>";
...
?>
```

A execução desse trecho de programa gerará o seguinte resultado:

```
A loja já possui 2 bermudas.  
A loja já possui 3 camisetas.
```

Note que utilizamos o símbolo `->` para acessar funções e variáveis da classe `loja`. Esse é o símbolo que você deve utilizar sempre que estiver trabalhando com objetos.

## Programação orientada a objetos no PHP 5

Uma das principais mudanças da versão 5 do PHP consiste no amadurecimento da linguagem no que diz respeito à programação orientada a objetos (OOP). A partir dessa versão, o PHP passa a operar com a Zend Engine 2.0, utilizando um novo modelo de objetos. Esse modelo deve conferir ao PHP, em termos de OOP, significativos ganhos de velocidade e performance, visto que ele é mais otimizado e realiza muito menos cópias redundantes de dados, como ocorria no modelo antigo.

A seguir são apresentadas algumas das novas características da OOP existentes no PHP 5. Para obter mais informações sobre essas e outras alterações na OOP, consulte o site oficial do PHP (<http://www.php.net>), onde está disponível uma página que lista as características da Zend Engine 2.

### As palavras-chave `private` e `protected`

Utilizando as palavras “`private`” e “`protected`” podemos criar métodos ou variáveis privadas e protegidas em uma classe. Uma variável privada só poderá ser acessada pela própria classe onde ela foi declarada, enquanto que uma variável protegida poderá ser acessada também pelas subclasses da classe onde ela foi declarada. Veja um exemplo.

#### exemplo6\_14.php

```
<?php  
class Classe1 {  
    private $var1 = "Olá, var1!\n";  
    protected $var2 = "Olá, var2!\n";  
    protected $var3 = "Olá, var3!\n";  
    function bomDia() {  
        print "Classe1: ". $this->var1. "<br>";  
        print "Classe1: ". $this->var2. "<br>";  
        print "Classe1: ". $this->var3. "<br><br>";  
    }  
}
```

```
class Classe2 extends Classe1 {
    function bomDia() {
        Classe1::bomDia(); // Exibe
        print "Classe2: " . $this->var1. "<br>"; // Não exibe nada
        print "Classe2: " . $this->var2. "<br>"; // Exibe
        print "Classe2: " . $this->var3. "<br>"; // Exibe
    }
}
$obj = new Classe1();
$obj->bomDia();
$obj = new Classe2();
$obj->bomDia();
?>
```

Nesse exemplo, a *Classe2* foi declarada como subclasse da *Classe1*. A variável *\$var1* foi declarada como privada (*private*), e portanto ela só poderá ser acessada pela própria *Classe1*. As variáveis *\$var2* e *\$var3* são protegidas (*protected*), e portanto podem ser acessadas pela subclasse *Classe2*.

Uma classe pode conter também métodos privados ou protegidos. Por exemplo:

#### exemplo6\_15.php

```
<?php
class Classe1 {
    private function MetodoPrivado() {
        echo "Classe1::MetodoPrivado() chamado.<br>";
    }
    protected function MetodoProtegido() {
        echo "Classe1::MetodoProtegido() chamado.<br>";
        $this->MetodoPrivado();
    }
}
class Classe2 extends Classe1 {
    public function MetodoPublico() {
        echo "Classe2::MetodoPublico() chamado.<br>";
        $this->MetodoProtegido();
    }
}
$obj = new Classe2();
$obj->MetodoPublico();
?>
```

Nesse exemplo, foi criado um objeto para a subclasse *Classe2*, e foi chamado o método *MetodoPublico()*. Esse método chama o *MetodoProtegido()* de sua superclasse, que por sua vez chama o *MetodoPrivado()*. Uma chamada direta da *Classe2* ao método privado não iria funcionar.

## Métodos abstratos e interfaces

No PHP 5 é possível criar métodos abstratos. Isso significa que o método é apenas declarado, mas sua implementação não é fornecida. Isso será feito posteriormente em outra classe. Por exemplo:

### exemplo6\_16.php

```
<?php
abstract class ClasseAbstrata {
    abstract public function teste();
}
class ClasseImplementacao extends ClasseAbstrata {
    public function teste() {
        echo "Método teste() chamado!<br>";
    }
}
$obj = new ClasseImplementacao;
$obj->teste();
?>
```

Veja que a classe que contém o método abstrato também deve ser definida como abstrata, e não poderá ser utilizada para criar objetos diretamente.

No PHP 5 também temos a possibilidade de criar interfaces. Nelas, inserimos apenas as declarações dos métodos que farão parte da classe que irá implementá-la. É um recurso semelhante às classes abstratas. Uma das diferenças é que uma classe pode implementar diversas interfaces. Utilizamos a palavra reservada *implements* para indicar que uma classe implementa uma determinada interface. Exemplo:

```
<?php
interface MinhaInterface {
    public function Teste();
}
```

```
<?php  
class MinhaClasse implements MinhaInterface {  
    public function Teste() {  
        // ...  
    }  
}  
?>
```

## A palavra-chave *final*

Os métodos que forem declarados com a palavra *final* não poderão ser sobreescritos pelas subclasses. Exemplo:

```
<?php  
class Minhaclasse {  
    final function Teste() {  
        // ...  
    }  
}  
?>
```

Nesse exemplo, se fosse declarado um método com o nome *Teste* em uma subclass de *MinhaClasse*, ele não iria sobrecrever o método declarado como *final*. Também é possível declarar uma classe como *final*. Exemplo:

```
<?php  
final class MinhaClasse {  
    // ...  
}  
?>
```

Isso significa que ela não poderá ter subclasses. Portanto, a linha de código apresentada a seguir não iria funcionar.

```
class outraclasse extends Minhaclasse {}
```

## Construtores e destrutores

Um construtor consiste em um método que será chamado toda vez que for criado um objeto da classe onde ele foi declarado. Portanto, pode ser utilizado para inicializar um objeto antes dele ser usado. Ao contrário das versões anteriores, onde o construtor deveria possuir o mesmo nome da classe, no PHP 5 esse método deve ser criado com o nome *\_\_construct()*. Por exemplo:

### exemplo6\_17.php

```
<?php
class Classe {
    function __construct() {
        print "Este é o construtor da Classe<br>";
    }
}
class SubClasse extends Classe {
    function __construct() {
        parent::__construct();
        print "Este é o construtor da SubClasse<br>";
    }
}
$obj = new Classe();
$obj = new SubClasse();
?>
```

A vantagem dessa mudança é que, se a segunda classe tivesse que ser movida para ser subclasse de alguma outra, não haveria a necessidade de alterar o seu código, já que todos os construtores estariam nomeados como `__construct()`. Por questões de compatibilidade, se o PHP não encontrar o método `__construct()` ele irá procurar por um método que possua o mesmo nome da classe. Além de um construtor, uma classe pode ter também um destrutor, que deve ser nomeado como `__destruct()`. Um destrutor é o método que será chamado após a última referência feita a um objeto no programa, antes da liberação da memória. Veja o exemplo a seguir:

### exemplo6\_18.php

```
<?php
class MinhaClasse {
    function __construct() {
        $this->nome = "MinhaClasse";
        print "Este é o construtor da classe ". $this->nome."<br>";
    }
    function __destruct() {
        print "Este é o destrutor da classe ". $this->nome;
    }
}
$obj = new MinhaClasse();
?>
```

Um destrutor pode ser útil para fins de depuração, fechamento de conexão com banco de dados, entre outras tarefas.

## Variáveis e métodos estáticos

As variáveis estáticas (precedidas pela palavra *static*) agora podem ser inicializadas, como mostra o exemplo a seguir.

### exemplo\_19.php

```
<?php
class Classe {
    static $variavel_estatica = 10;
}
print Classe::$variavel_estatica;
?>
```

Além disso, no PHP 5 a palavra *static* também pode ser utilizada para definir um método como estático. Dessa forma, é possível chamá-lo mesmo sem a criação de um objeto para a classe onde ele foi declarado. Exemplo:

```
<?php
class Classe {
    public static function MetodoEstatico() {
        // ...
    }
    Classe::MetodoEstatico();
?>
```



# Capítulo 7

## Utilizando includes em PHP

As includes são uma excelente opção oferecida pelo PHP. Elas nos permitem reaproveitar uma ou mais funções ou arquivos, utilizando-os em diversas páginas do site. As situações em que as includes são mais usadas são aquelas em que uma alteração deve ser refletida em todas as páginas do site automaticamente, e para isso precisamos alterar apenas um arquivo, que será acessado por todas as páginas.

### Criando um menu para seu site

No início deste livro foi exposta a seguinte situação: imagine um site que possui cerca de 100 páginas, e que no lado esquerdo das páginas há um menu com links para as seções do site. Se alguma seção for incluída ou excluída, como fazer para atualizar as 100 páginas, incluindo ou excluindo esse novo link? Alterar manualmente as 100 páginas? Pois bem... chegou a hora de conhecermos a solução para esse problema.

Vamos supor que temos um site especialista em informações sobre cinema, que possui as seguintes seções: Em Cartaz, Destaques, Em Breve, Trilha Sônica, Ranking, Trailer, Bilheteria, Programação, Galeria de Fotos, Links, Arquivo e Opinião, conforme mostrado na figura 7.1.

O menu que você vê no lado esquerdo da página é o mesmo utilizado em todas as páginas do site, para que o visitante possa acessar a seção desejada a partir de qualquer página. Suponha que o nome da página apresentada seja *index.php*.

Só que agora resolvemos incluir uma nova seção no site: a seção “Crítica”. Não vamos alterar individualmente cada uma das páginas, pois com certeza estaremos utilizando esse menu como uma include.

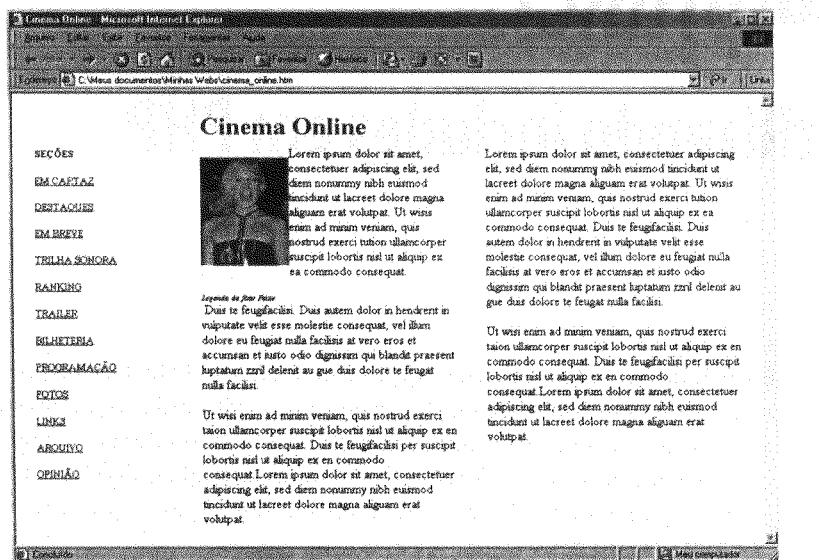


Figura 7.1 – Site de cinema dividido em seções.

A função *include* tem por objetivo incluir um arquivo dentro de outro, e sua sintaxe é a seguinte:

```
<?php include "nome_do_arquivo"; ?>
```

Então vamos transformar esse menu em uma include! Para isso devemos realizar quatro passos importantes:

- 1) identificar no código HTML onde está o menu;
- 2) retirar esse código HTML da página (pode ser com as teclas CTRL+X), abrir um novo arquivo e colar esse código no novo arquivo aberto (CTRL+C);
- 3) salvar esse novo arquivo como *menu.inc*;
- 4) no lugar de onde você retirou o código do menu, coloque uma chamada para a include entre as tags do PHP:

```
<?php include "menu.inc"; ?>
```

No caso desse nosso site de cinema, veja como ficaria o arquivo *menu.inc*:

```
<font size="2">
<p><strong>Seções<br></strong></p>
<p><a href="cartaz.php">EM CARTAZ</a></p>
```

```
<p><a href="destaques.php">DESTAQUES</a></p>
<p><a href="breve.php">EM BREVE</a></p>
<p><a href="trilha.php">TRILHA SONORA</a></p>
<p><a href="ranking.php">RANKING</a></p>
<p><a href="trailer.php">TRAILER</a></p>
<p><a href="bilheteria.php">BILHETERIA</a></p>
<p><a href="programacao.php">PROGRAMAÇÃO</a></p>
<p><a href="fotos.php">FOTOS</a></p>
<p><a href="links.php">LINKS</a></p>
<p><a href="arquivo.php">ARQUIVO</a></p>
<p><a href="opiniao.php">OPINIÃO</a></p>
</font>
```

O arquivo *menu.inc* seria uma include que contém os comandos HTML para formar o menu principal do site. Portanto, em todas as páginas do site, no local onde deve constar o menu, aparecerá uma chamada para a include:

```
<?php include "menu.inc"; ?>
```

Mas lembre-se de que todas as páginas que utilizam includes devem possuir extensão .php, pois elas executam comandos da linguagem.

Pronto! Agora ficou bem mais fácil incluir, alterar ou excluir novas seções no site, pois temos o menu em um único arquivo que é acessado por todas as páginas. Em nosso site de cinema, para incluir a seção “Crítica”, bastaria abrir o arquivo *menu.inc* e acrescentar a seguinte linha:

```
<p><a href="critica.php">CRÍTICA</a></p>
```

Do mesmo modo que criamos um menu lateral por meio de uma include, podemos criar um cabeçalho e um rodapé-padrão, incluindo o logotipo do site, o banner de um patrocinador, direitos autorais, links adicionais etc. Para isso primeiro você deve criar, por exemplo, os arquivos *cabecalho.inc* e *rodape.inc*, e dentro deles incluir o conteúdo que você desejar. Depois basta incluir em cada página, no topo e no rodapé, as chamadas para os dois arquivos criados:

```
<?php include "cabecalho.inc"; ?>
...
<?php include "rodape.inc"; ?>
```

Agora acabou seu sofrimento! Nunca mais você precisará alterar uma a uma todas as páginas de seu site.

## Exibindo a data atual com uma include

É muito comum hoje em dia a apresentação da data atual nas páginas de um site. Podemos criar uma include que mostre a data atual, e chamá-la de todas as páginas em que é necessária a exibição da data. Essa include poderia ser feita da seguinte maneira:

### data.inc

```
<?php  
$meses = array ("Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",  
"Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro");  
$dia = date ("d", time());  
$mes = date ("m", time());  
$ano = date ("Y", time());  
echo $dia . " de " . $meses [$mes-1] . " de " . $ano;  
?>
```

Construímos um array com os nomes de todos os meses, e logo após utilizamos a função *date* do PHP para obter os valores atuais do dia, mês e ano. O comando *date* retorna o valor numérico do mês, e nós utilizamos esse valor decrementado (pois o array começa em 0) como índice para acessar o array dos meses. O ano será mostrado com 4 dígitos, pois utilizamos o “Y” maiúsculo. O “Y” minúsculo retorna o ano em 2 dígitos apenas.

Salvando o programa acima como *data.inc*, você terá sua include pronta. O próximo passo é colocar as chamadas para essa include nas páginas em que a data deve aparecer:

```
<?php include "data.inc"; ?>
```

Quando a include é ativada, a data atual é mostrada na tela, no formato que especificamos quando utilizamos o comando *echo*. Exemplo:

22 de Março de 2001

## Reutilização de código

Dentro das includes também podemos definir funções, e essas funções estarão disponíveis para todas as páginas que chamarem a include. Dessa forma estamos reutilizando o código das funções, em vez de escrevê-lo novamente. Já vimos anteriormente uma função que converte uma string em letras maiúsculas, alterando caracteres acentuados e não-acentuados. Vimos também uma função que faz a validação de um número de CPF digitado. Se precisássemos

utilizar essas funções em outras páginas, bastaria colocá-las dentro de uma include, e no início das páginas colocar uma chamada para essa include.

Se colocássemos essas duas funções dentro de uma include chamada *funcoes\_aux.inc*, poderíamos utilizá-las nas páginas da seguinte maneira:

```
<?php include "funcoes_aux.inc"; ?>
<html>
<body>
...
<?php
    $nome = maiusculo ($nome);
    if (cpf_errado($cpf))
        { echo "O CPF digitado é inválido"; }
?
</body>
</html>
```

## Include x Require

Além do comando *include*, existe também o comando *require*. Os dois comandos têm a mesma função, mas possuem uma pequena diferença: o comando *include* é reavaliado a cada chamada, ou seja, uma mesma instrução *include* pode incluir vários arquivos (por meio de um laço, por exemplo); enquanto a instrução *require* pode incluir apenas um arquivo.

Suponha que temos um array com o nome de todos os arquivos que devem ser incluídos durante a execução da página, como é mostrado a seguir:

```
$incluir = array ("funcoes_aux.inc" , "cabecalho.inc" , "data.inc");
```

Para realizar a inclusão desses arquivos, resolvemos fazer um laço com o comando *for*:

```
for ($i=0 ; $i<sizeof($incluir) ; $i++)
{
    require "{$incluir[$i]}";
}
```

Executando esse laço, o único arquivo que será incluído é o *funcoes\_aux.inc*. Os arquivos *cabecalho.inc* e *data.inc* não serão incluídos, pois a função *require* não inclui arquivos dinamicamente.

Para resolver esse problema, devemos utilizar a função *include* no lugar da função *require*, pois a cada iteração do laço a função *include* é reavaliada. Portanto, o correto seria escrever o programa da seguinte maneira:

```
for ($i=0 ; $i<sizeof($incluir) ; $i++)
{
    include "$incluir[$i]";
}
```

Uma observação importante sobre o comando *include* é que quando utilizado com comandos condicionais ele deve sempre aparecer entre chaves, pois geralmente uma *include* executa mais de um comando. Exemplo:

```
<?php
if ($op==1)
{ include "menu1.inc"; }
elseif ($op==2)
{ include "menu2.inc"; }
else
{ include "menu3.inc"; }
?>
```

O resultado desse código é que se \$op for igual a 1, o menu1.inc é executado, se \$op for igual a 2, o menu2.inc é executado e se \$op não for 1 ou 2, o menu3.inc é executado. Isso ocorre porque o comando *include* é executado dentro de um bloco condicional, portanto, o bloco condicional é executado antes de o comando *include* ser executado.

Outro exemplo é o de uma estrutura de decisão que inclui uma operação matemática.

```
<?php
if ($a>$b)
{ $c=$a-$b;
  $d=$a*$b;
}
else
{ $c=$b-$a;
  $d=$b*$a;
}
?>
```

O resultado desse código é que se \$a for maior que \$b, o resultado da operação é \$c e \$d é \$a multiplicado por \$b.

Se \$a for menor que \$b, o resultado da operação é \$c e \$d é \$b multiplicado por \$a.

# Capítulo 8

## PHP e formulários HTML

Talvez esta seja uma das principais razões que levou você a adquirir este livro: como criar um formulário para os usuários de meu site preencherem? Como faço para enviar essas informações para o meu programa PHP? E como devo tratá-las dentro do meu programa? Chegou a hora de conhecer as respostas para todas essas perguntas. Após a leitura deste tópico, você estará pronto para criar formulários que deixarão seu site muito mais interativo.

### Como criar um formulário

Os formulários são criados por meio da linguagem HTML, que você já deve conhecer bem. Um formulário é composto de no mínimo um campo para entrada de dados, e um botão para enviar as informações contidas nele. Veja um exemplo de formulário:

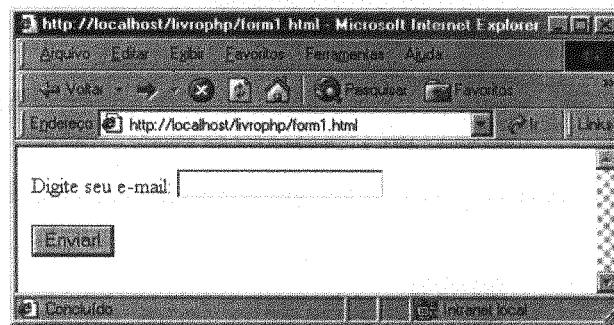


Figura 8.1 – Exemplo de formulário HTML.

O código HTML para esse formulário é o seguinte:

```
<form>
    <p>Digite seu e-mail: <input type="text" name="email" size="20"></p>
    <p><input type="submit" value="Enviar!" name="enviar"></p>
</form>
```

Se você clicar no botão "Enviar!" desse formulário, o valor que foi preenchido no campo e-mail não será enviado a lugar nenhum, pois seu browser não sabe para onde enviar as informações. Portanto, o formulário definido anteriormente não tem nenhuma utilidade, pois perdemos as informações quando clicamos no botão "Enviar!".

Para tornar esse formulário útil, devemos informar ao browser para onde devem ser enviadas as informações. Você utilizará os formulários para enviar dados aos seus programas PHP, então o correto é informar ao browser qual programa receberá esses dados. Isso é feito com a opção *action*, utilizada na tag *form* do HTML. Veja o exemplo a seguir:

```
<form action="recebe_dados.php">
<p>Digite seu e-mail: <input type="text" name="email" size="20"></p>
<p><input type="submit" value="Enviar!" name="enviar"></p>
</form>
```

Agora quando você clicar no botão "Enviar!", o conteúdo do campo e-mail será enviado ao programa *recebe\_dados.php*, e esse programa tratará a informação recebida. Só falta você criar o programa especificado, e nele fazer o processamento das informações do modo que achar melhor.

Observe que para definir um campo de entrada de dados utilizamos a tag *input*. Essa tag pode vir acompanhada de diversas opções. Vamos ver as principais:

| Opção            | Descrição   |
|------------------|---|
| <b>name</b>      | Informa qual é o nome do campo.                                       |
| <b>value</b>     | Informa um valor padrão para o campo.                                 |
| <b>size</b>      | Informa o tamanho do campo exibido na tela.                           |
| <b>maxlength</b> | Informa o número máximo de caracteres que pode ser digitado no campo. |
| <b>type</b>      | Informa qual é o tipo do campo de entrada de dados.                   |

Os valores possíveis para a opção *type* da tag *input* são mostrados na tabela a seguir:

| Valor           | Descrição  |
|-----------------|--|
| <b>text</b>     | Mostra uma caixa de texto de uma linha, e permite a entrada de valores numéricos ou alfanuméricos.   |
| <b>password</b> | Utilizado para a digitação de senhas. São mostrados asteriscos (*) no lugar dos caracteres digitados, mas a informação é enviada normalmente.  |
| <b>hidden</b>   | É um campo escondido. Não aparece na tela. Podemos utilizá-lo para passar informações aos programas que recebem os dados. Veremos mais adiante que esses campos são de grande utilidade. |

| Valor           | Descrição (continuação)  |
|-----------------|--|
| <b>select</b>   | Mostra uma lista de seleção (também conhecida como drop-down).   |
| <b>checkbox</b> | Exibe uma caixa de seleção, que pode ser marcada ou desmarcada.  |
| <b>radio</b>    | São botões de seleção, em que o usuário escolhe uma entre várias opções disponíveis.                       |
| <b>textarea</b> | Caixa de texto com várias linhas.  |
| <b>file</b>     | Permite o envio de arquivos.   |
| <b>submit</b>   | Botão que aciona o envio dos dados do formulário.  |
| <b>image</b>    | Tem a mesma função que o <b>submit</b> , mas utiliza uma imagem em vez do botão tradicional do formulário. |
| <b>reset</b>    | Limpa todos os campos de um formulário e retorna ao valor-padrão (se existir).                             |

Veja na figura 8.2 um exemplo de formulário que utiliza diversos tipos de campos:

The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost/fivrophp/form2.html>. The form contains the following fields:

- A question "O que você achou do site?" followed by a group of radio buttons: "Muito Bom", "Bom", "Regular", and "Um Lixo".
- A question "Qual a seção que você mais gostou?" with an input field containing "Em Cartaz" and a link "Outra: \_\_\_\_\_".
- A question "Digite seus comentários no espaço abaixo:" with a large text area.
- A question "Diga-nos como entrar em contato com você:" with four input fields labeled "Nome", "E-mail", "Fone", and "FAX".
- A checkbox labeled "Quero receber as novidades do site por e-mail".
- Two buttons at the bottom: "Enviar Dados" and "Limpar Campos".
- A "Concluído" button at the bottom right.

Figura 8.2 – Formulário HTML com diferentes tipos de campos.

O código HTML para esse formulário é o seguinte:

```
<form action="processa.php">
<p><strong>O que você achou do site?</strong></p>
<dl><dd>
<input type="radio" name="avaliacao" value="muitobom" checked>Muito Bom
<input type="radio" name="avaliacao" value="bom">Bom
<input type="radio" name="avaliacao" value="regular">Regular
<input type="radio" name="avaliacao" value="umlixo">Um Lixo
</dd></dl>
<p><strong>Qual a seção que você mais gostou?</strong></p>
<dl><dd>
<select name="secao" size="1">
<option value="emcartaz">Em Cartaz</option>
<option value="trilhasonora">Trilha Sonora</option>
<option value="fotos">Galeria de Fotos</option>
<option value="bilheteria">Bilheteria</option>
<option value="outra">Outra</option>
</select>
Outra: <input type="text" size="26" maxlength="256" name="outra">
</dd></dl>
<p><strong>Digite seus comentários no espaço abaixo:</strong></p>
<dl><dd>
<textarea name="Comentarios" rows="5" cols="42"></textarea>
</dd></dl>
<p><strong>Diga-nos como entrar em contato com você:</strong></p>
<dl><dd>
<pre>
Nome    <input type="text" size="35" maxlength="256" name="nome">
E-mail  <input type="text" size="35" maxlength="256" name="email">
Fone    <input type="text" size="35" maxlength="256" name="fone">
FAX     <input type="text" size="35" maxlength="256" name="fax">
</pre>
</dd></dl>
<dl><dd>
<input type="checkbox" name="novidades" value="nov">
Quero receber as novidades do site por e-mail
</dd></dl>
<p><input type="submit" value="Enviar Dados">
<input type="reset" value="Limpar Formulário"></p>
</form>
```

## Enviando as informações para um programa PHP

Vimos no tópico anterior que para especificar qual programa PHP receberá os dados do formulário utilizamos a opção *action* da tag *form* do HTML. Exemplo:

```
<form action="processa_dados.php">
```

Resta saber como esses dados são passados ao programa PHP. Existem dois métodos de passagem de parâmetros: *GET* e *POST*. No caso de um formulário, o tipo de método a ser utilizado é especificado na opção *method* da tag *form*. Exemplo:

```
<form action="processa_dados.php" method="POST">
```

Vamos ver com detalhes cada um deles para que você entenda a diferença.

### Método GET

Esse é o método-padrão para o envio de dados. Se no momento da criação de um formulário nenhum método for especificado na opção *method* da tag *form*, estaremos utilizando o método *GET* para o envio dos dados.

Nesse método, os dados serão enviados juntamente com o nome da página (na URL) que processará os dados recebidos. Considere o seguinte formulário como exemplo:

```
<form action="recebe_dados.php">
  <p>Digite seu nome: <input type="text" name="nome" size="30"></p>
  <p>Digite sua idade: <input type="text" name="idade" size="3"></p>
  <p><input type="submit" value="Enviar!" name="enviar"></p>
</form>
```

Note que esse formulário não mostra a opção *method*, portanto o padrão é adotado (*method="GET"*). Suponha que preenchemos o campo *nome* com o valor *Joaquim*, e o campo *idade* com o valor *20*. Logo após clicarmos no botão “Enviar!”, o endereço ativado será o seguinte:

[http://www.seusite.com.br/recebe\\_dados.php?nome=Joaquim&idade=20](http://www.seusite.com.br/recebe_dados.php?nome=Joaquim&idade=20)

Os campos do formulário serão passados como parâmetros após o endereço de destino. O caractere ? representa o início de uma cadeia de variáveis, e o símbolo & identifica o início de uma nova variável. As variáveis e seus respectivos valores são separadas pelo caractere =.

Existem alguns inconvenientes de utilizar o método *GET*. Primeiramente porque há um limite de caracteres que podem ser enviados (em torno de 2.000 caracteres). Outro problema é que o usuário enxergará todos os parâmetros por meio da barra de endereço do browser, o que não é muito agradável. Para resolver esses problemas existe o método *POST*.

Em compensação, o método *GET* possui uma grande vantagem: além de enviar informações via formulários, esse método pode ser utilizado também para a passagem de parâmetros por meio de links. Imagine, por exemplo, uma loja virtual, onde há um link para cada produto. Certamente cada link passa como parâmetro um número identificador do produto, para ser tratado por determinado programa. O link para um produto específico poderia ser o seguinte:

[http://www.lojinhadojoao.com.br/produto.php?id\\_produto=50](http://www.lojinhadojoao.com.br/produto.php?id_produto=50)

Dessa forma, um programa chamado *produto.php* receberia o número identificador do produto e realizaria uma consulta ao banco de dados para buscar informações detalhadas, como descrição, preço etc. Podemos, se necessário, passar mais de um parâmetro através do método *GET*. Para isso basta utilizar o símbolo & fazendo a separação. Por exemplo, o link para consulta de uma subcategoria da loja poderia ser o seguinte:

<http://www.lojinhadojoao.com.br/produto.php?categoria=2&subcategoria=5>

Somente com o método *GET* podemos passar parâmetros por links. O método *POST* trabalha somente com formulários.

## Método POST

Para utilizar o método *POST* devemos utilizar opção *method* da tag *form* para informar ao browser. Exemplo:

```
<form action="recebe_dados.php" method="POST">
    <p>Digite seu nome: <input type="text" name="nome" size="30"></p>
    <p>Digite sua idade: <input type="text" name="idade" size="3"></p>
    <p><input type="submit" value="Enviar!" name="enviar"></p>
</form>
```

Ao contrário do método *GET*, que envia os dados por uma cadeia de variáveis após o endereço-destino, o método *POST* envia os dados do formulário por meio do corpo da mensagem encaminhada ao servidor.

Como os dados são enviados no corpo da mensagem, quando o usuário clicar no botão "Enviar!" ele não verá em sua barra de endereços aquele endereço enorme contendo uma cadeia de variáveis. Ele verá apenas o endereço do programa ativado:

[http://www.seusite.com.br/recebe\\_dados.php](http://www.seusite.com.br/recebe_dados.php)

Outra grande vantagem do método *POST* é que não há limitação de tamanho dos dados que estão sendo enviados, ao contrário do método *GET*, que envia os dados por uma cadeia de variáveis de tamanho limitado. Portanto é recomendado utilizar o método *POST* para formulários que possuem muitas informações a serem enviadas. O método *GET* pode ser usado em formulários mais simples, que possuem poucos campos.

Por meio do método *POST* podemos enviar outros tipos de dados que não podem ser enviados pelo método *GET*, como, por exemplo, imagens ou outros arquivos (para isso utilizamos o valor *file* na opção *type* da tag *input* do HTML).

## Como tratar as informações recebidas

Depois de especificado qual o programa PHP que receberá os dados (na opção *action* da tag *form*), basta saber como trabalhar com esses dados dentro do programa PHP. Existem duas maneiras de acessá-los:

- 1) Tratá-los como se fossem variáveis, colocando o símbolo \$ seguido do próprio nome do campo definido no formulário. Exemplo: o campo *nome* do formulário poderia ser referenciado dentro do programa PHP pela variável *\$nome*. O campo *email* seria referenciado pela variável *\$email*, e assim por diante. Porém, esse método só funciona se a opção *register\_globals* estiver habilitada no arquivo de configuração *php.ini* (por padrão, desde o PHP 4.2.0 essa opção está desabilitada).
- 2) Utilizar os arrays superglobais predefinidos pelo PHP. Existem dois arrays que o PHP utiliza: um para armazenar os valores enviados pelo método *GET* e outro para armazenar informações enviadas pelo método *POST*. Esses arrays são o *\$\_GET* e o *\$\_POST*.

Nesse caso, o nome dos campos do formulário é usado como chave associativa, e o valor dos campos é armazenado como os valores do array. Se usássemos, por exemplo, o método *POST* para enviar um formulário com os campos *nome* e *email*, dentro do programa PHP acessaríamos esses dados da seguinte maneira:

```
$_POST["nome"]  
$_POST["email"]
```

Se o método utilizado fosse o *GET* usaríamos:

```
$_GET["nome"]  
$_GET["email"]
```

Por questões de segurança, os desenvolvedores do PHP recomendam o uso da segunda opção, ou seja, acessar os dados enviados a partir dos arrays `$_GET` e `$_POST`. Porém, se você preferir utilizar a primeira opção, tome cuidado para não criar dentro do programa outras variáveis com o mesmo nome dos campos do formulário, pois isso fará com que o conteúdo do campo seja sobreescrito.

## Funções especiais para formatação de dados

Há algumas providências a serem tomadas quando um programa recebe um conjunto de dados. Imagine que você resolve desenvolver um fórum de discussão. Um fórum é um local onde todos os usuários podem colocar mensagens para debater sobre determinado assunto.

Então você desenvolve um programa que recebe as informações de um formulário e as grava em um banco de dados ou em simples arquivos em formato texto. Terminado esse programa, você coloca seu fórum para funcionar na Internet. No dia seguinte você diz: "Vou lá ver se os usuários estão colocando mensagens no meu fórum!". Você entra em seu site, acessa o fórum e encontra uma foto pornô colocada em uma das mensagens. Isso ocorreu porque um usuário, ao escrever sua mensagem, digitou um comando HTML que exibe uma imagem existente em outro site. Ele poderia, por exemplo, ter digitado:

```

```

Como nenhuma precaução foi tomada, os usuários poderiam utilizar qualquer comando HTML no meio de suas mensagens, e isso possivelmente faria com que conteúdos não desejados fossem acrescentados ao seu site. A solução para esse problema é utilizar uma função do PHP para transformar os comandos HTML, fazendo com que estes sejam exibidos na tela como texto comum. Essa função é a `htmlspecialchars`, cuja sintaxe é a seguinte:

```
htmlspecialchars(<string>);
```

Essa função retira as tags HTML e coloca caracteres especiais em seus lugares. As mudanças feitas são as seguintes:

- & é substituído por &amp;
- " é substituído por &quot;
- < é substituído por &lt;
- > é substituído por &gt;

Ao aplicar a função *htmlspecialchars* sobre uma string e exibi-la na tela veremos apenas os comandos HTML digitados, e não os resultados gerados por esses comandos, pois a função transformou as tags em caracteres especiais que foram reconhecidos pelo browser. Acompanhe o exemplo a seguir:

### exemplo8\_1.php

```
<?php  
    $texto = "<img src=http://www.siteporno.com.br/foto1.jpg>";  
    $novo_texto = htmlspecialchars($texto);  
    echo $texto . "<br>";  
    echo $novo_texto;  
?>
```

A variável *\$texto* desse programa contém o valor original (digitado por um usuário, por exemplo) da string, e a variável *\$novo\_texto* contém o valor da variável *\$texto* após a aplicação da função *htmlspecialchars*. A execução desse programa mostra duas linhas na tela: a primeira linha mostrará a imagem *foto1.jpg*, que é “puxada” de outro servidor. A segunda linha mostra o valor da string em formato texto:

```
<img src=http://www.siteporno.com.br/foto1.jpg>
```

Isso ocorreu porque a função *htmlspecialchars* transformou os caracteres > e < em &gt; e &lt;. O browser leu esses códigos e apresentou na tela novamente os símbolos > e <, mas não os executou como se fossem parte do código HTML.

Outra precaução a ser tomada no momento de receber os dados é a utilização de dados que contêm caracteres especiais. Se, por exemplo, o usuário digitar no campo *nome* do formulário a seguinte informação:

João da Silva, vulgo “João da Maloca”

O valor recebido pelo programa PHP será:

```
João da Silva, vulgo \"João da Maloca\"
```

O PHP coloca o caractere de controle \ antecedendo caracteres especiais para evitar um erro na leitura das variáveis. Existe uma função que, aplicada a uma string, retira seus caracteres de controle. Essa função é a *stripslashes*, cuja sintaxe é a seguinte:

```
stripslashes (<string>);
```

Ao aplicar essa função sobre uma string, \" é substituído por ", \' é substituído por ', e assim por diante. Dois caracteres de controle seguidos (\\) são substituídos por apenas uma barra invertida (\).

Quando utilizamos o método *GET*, outras funções que podem ser utilizadas no tratamento dos dados recebidos são as funções *urldecode* e *urlencode*. Essas funções são úteis porque todos os caracteres que não são alfanuméricos (com exceção do *\_*) são convertidos para um código hexadecimal antecedido pelo símbolo %. Por exemplo: passando o nome "Joaquim de Souza" pelo método *GET*, teremos a seguinte URL:

[http://www.seusite.com.br/recebe\\_dados.php?nome=Joaquim%20de%20Souza](http://www.seusite.com.br/recebe_dados.php?nome=Joaquim%20de%20Souza)

Então dentro do programa *recebe\_dados.php* deveríamos ter a seguinte linha:

```
$nome = urldecode ($nome);
```

Aplicando a função *urldecode* sobre a variável *\$nome*, os códigos hexadecimais serão substituídos pelos caracteres que eles representam. Já a função *urlencode* faz o processo contrário, transformando os caracteres não-alfanuméricos para seus respectivos códigos. Se a variável *\$nome* possuísse o valor "Joaquim de Souza", após a aplicação da função *urlencode* teríamos o valor Joaquim%20de%20Souza. Essa função é útil quando queremos redirecionar esses dados para outro programa PHP por meio do método *GET*.

## Verificando os campos de um formulário

Vamos ver um exemplo de verificação dos campos de um formulário. Suponha que temos um formulário com os seguintes campos: *username*, *senha*, *nome*, *email*, *cidade* e *estado*, conforme mostrado na figura 8.3:

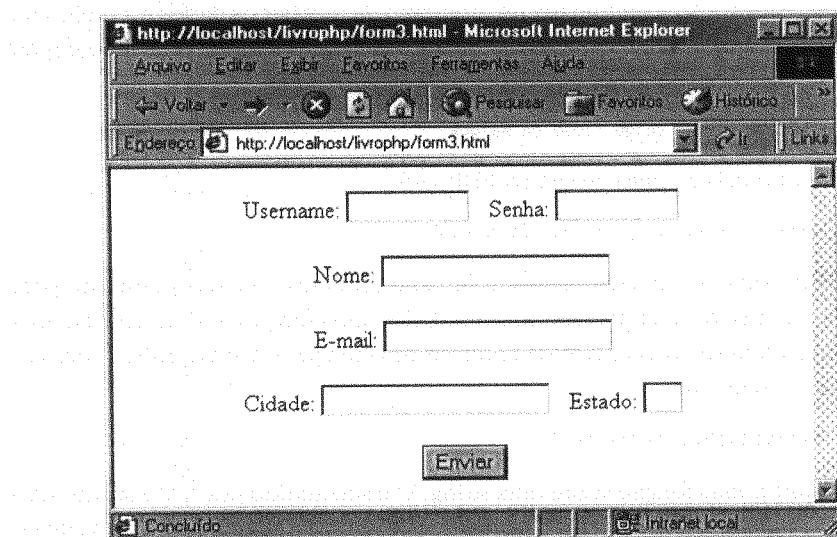


Figura 8.3 – Formulário a ser validado.

O código para esse formulário HTML é o seguinte:

```
<form method="POST" action="recebe_dados.php">
<div align="center"><center>
<p>Username: <input type="text" name="username" size="10">&nbsp;&nbsp;
    Senha: <input type="password" name="senha" size="10"></p>
</center></div>
<div align="center"><center>
    <p>Nome: <input type="text" name="nome" size="20"></p>
</center></div>
<div align="center"><center>
    <p>E-mail: <input type="text" name="email" size="20"></p>
</center></div>
<div align="center"><center>
    <p>Cidade: <input type="text" name="cidade" size="20">&nbsp;&nbsp; Estado:<br>
        <input type="text" name="estado" size="2" maxlength="2"></p>
</center></div>
<div align="center"><center>
    <p><input type="submit" value="Enviar" name="enviar"></p>
</center></div>
</form>
```

Após o usuário preencher o formulário e clicar no botão “Enviar”, os dados serão submetidos ao programa *recebe\_dados.php*, que fará a verificação. Vamos ver então como poderia ser esse programa. Lembre-se que primeiramente temos que obter os dados enviados pelo usuário acessando o array predefinido *\$\_POST*, visto que o formulário está utilizando esse método de envio (*method="POST"*).

### **recebe\_dados.php**

```
<html>
<body>
<?php
    $username = $_POST["username"];
    $senha = $_POST["senha"];
    $nome = $_POST["nome"];
    $email = $_POST["email"];
    $cidade = $_POST["cidade"];
    $estado = $_POST["estado"];
    $erro=0;
    if($username==""){$erro=1;}
    if($senha==""){$erro=1;}
    if($nome==""){$erro=1;}
    if($email==""){$erro=1;}
    if($cidade==""){$erro=1;}
    if($estado==""){$erro=1;}
    if($erro==0){echo "Parabéns! Seus dados foram enviados com sucesso!"}
    else{echo "Por favor, preencha todos os campos."}
?>
```

```

if (strlen($username)<5)
{ echo "O username deve possuir no mínimo 5 caracteres.<br>"; $erro=1; }
if (strlen($senha)<5)
{ echo "A senha deve possuir no mínimo 5 caracteres.<br>"; $erro=1; }
if ($username == $senha)
{ echo "O username e a senha devem ser diferentes.<br>"; $erro=1; }
if (empty($nome) OR strstr ($nome, ' ')==FALSE)
{ echo "Favor digitar seu nome corretamente.<br>"; $erro=1; }
if (strlen($email)<8 || strstr ($email, '@')==FALSE)
{ echo "Favor digitar seu e-mail corretamente.<br>"; $erro=1; }
if (empty($cidade))
{ echo "Favor digitar sua cidade.<br>"; $erro=1; }
if (strlen($estado)!=2)
{ echo "Favor digitar seu estado corretamente.<br>"; $erro=1; }
// VERIFICA SE NÃO HOUVE ERRO
if($erro==0)
{ echo "Todos os dados foram digitados corretamente!"; }
?>
</body>
</html>

```

Esse programa testa cada um dos campos do formulário, e imprime mensagens de erro para cada campo que foi digitado incorretamente ou foi deixado em branco. Algumas das funções utilizadas devem ser desconhecidas para você, por isso veremos uma rápida explicação sobre cada uma:

| Função        | Descrição   |
|---------------|---|
| <b>empty</b>  | Verifica se determinada string está vazia (nula).                                 |
| <b>strlen</b> | Retorna o número de caracteres de uma string.                                     |
| <b>strstr</b> | Acha a primeira ocorrência de uma string. Se não encontrar retorna falso (FALSE). |

No programa *recebe\_dados.php* estamos fazendo os seguintes testes: verificando se o *username* e a *senha* possuem no mínimo 5 caracteres; se o *username* e a *senha* não são iguais; se o *nome* não está vazio e também se o *nome* possui o caractere espaço (se não houver, não foi digitado o nome completo); se o e-mail tem no mínimo 8 caracteres e possui o símbolo @; se o campo *cidade* não está em branco; se o campo *Estado* não possui dois caracteres.

Ess programa não tem muita utilidade porque ele apenas faz a verificação dos dados recebidos, mas não dá nenhum destino a eles. Mais adiante você verá como gravar esses dados em um banco de dados para acessá-los posteriormente.

# Capítulo 9

## Passando informações por várias páginas

No capítulo sobre formulários vimos como um programa PHP recebe os dados enviados por um formulário ou por um link. No entanto existem diversas situações em que precisamos repassar esses dados para as próximas páginas que serão chamadas. Um exemplo que é comum nos dias de hoje é o do cadastramento de usuários em um site. Como geralmente os formulários de cadastro possuem muitos campos a serem preenchidos, os desenvolvedores optam por dividir esse cadastramento em duas ou mais etapas (páginas). Desse forma deve haver uma comunicação entre as páginas, de modo que quando o usuário chegar na última etapa, saibamos todos os dados fornecidos por ele desde a primeira etapa.

Outro exemplo que pode ser citado é o de um site de busca, que possui uma página para cadastramento de novos sites. Geralmente esse cadastramento é dividido em etapas. Na primeira página você digita o nome do seu site, descrição, endereço, categoria etc. Depois você clica em um botão para prosseguir o cadastramento. Na segunda página são pedidas outras informações, como seu nome, e-mail, cidade, Estado etc. Na próxima etapa aparecem alguns campos de preenchimento opcional, como, por exemplo, áreas de interesse. Por fim é ativada uma última página, que recebe todos os dados enviados pelo usuário desde a primeira página e efetua o cadastramento dele no site.

Veremos agora como pode ser feita essa troca de informações entre as páginas, pelos métodos *GET* e *POST*. Logo após veremos um exemplo em que é feita a divisão de um cadastramento em etapas.

## Utilizando o campo *hidden* dos formulários

*Hidden* traduzido para o português significa escondido. Portanto, você já pode perceber que o tipo *hidden* serve para passar informações escondidas para a página que será acionada pelo formulário. Por ser um tipo de campo de um formulário, o *hidden* é geralmente utilizado quando estamos trabalhando com o método *POST* para o envio dos dados.

Veremos agora um exemplo simples em que podemos utilizar esse recurso. Vamos supor que temos um formulário para administrar uma loja, onde é possível incluir, alterar e excluir produtos. Os dados dos produtos são representados pelos seguintes campos do formulário: *codigo\_produto*, *nome\_produto* e *descricao\_produto*. Então poderíamos ter uma página HTML com três formulários, um para cada função, mas todos ativando o mesmo programa. Nesse caso, observe a utilização do campo *hidden*:

```
<p><strong>Inclusão de produtos</strong></p>
<form method="POST" action="gerencia.php">
<input type="hidden" name="operacao" value="inclusao">
<p>
Código do produto: <input type="text" name="codigo_produto" size="5"><br>
Nome do produto: <input type="text" name="nome_produto" size="20"><br>
Descrição do produto: <input type="text" name="descricao_produto" size="20">
</p>
<p><input type="submit" value="Incluir" name="incluir"></p>
</form>

<p><strong>Alteração de produtos</strong></p>
<form method="POST" action="gerencia.php">
<input type="hidden" name="operacao" value="alteracao">
Código do produto: <input type="text" name="codigo_produto" size="5"><br>
Novo Nome do produto: <input type="text" name="nome_produto" size="20"><br>
Nova Descrição do produto: <input type="text" name="descricao_produto" size="20">
</p>
<p><input type="submit" value="Alterar" name="alterar"></p>
</form>

<p><strong>Exclusão de produtos</strong></p>
<form method="POST" action="gerencia.php">
<input type="hidden" name="operacao" value="exclusao">
```

```
código do produto: <input type="text" name="codigo_produto" size="5"><br>
<input type="submit" value="Excluir" name="excluir"></p>
</form>
```

As opções de um campo do tipo *hidden* são:

| Opção              | Descrição   |
|--------------------|---|
| <code>name</code>  | Nome do campo, que posteriormente se tornará uma variável no programa PHP. O nome deve ser o mesmo que o campo <code>name</code> do formulário. |
| <code>value</code> | Valor do campo indicado por <code>name</code> .   |

Note que os três formulários acima ativam o programa *gerencia.php*. Portanto, esse programa tem de ser capaz de realizar as funções de inclusão, alteração e exclusão de produtos. O programa saberá qual das três funções executar quando consultar o campo *operacao*, passado no modo escondido (*hidden*) pelo formulário. Logo, o programa poderia ser escrito da seguinte maneira:

### gerencia.php

```
<?php
$operacao = $_POST["operacao"];
if ($operacao == "inclusao")
{
    // realiza a inclusão do produto
    echo "Produto incluído!";
}
elseif ($operacao == "alteracao")
{
    // realiza a alteração do produto
    echo "Produto alterado!";
}
elseif ($operacao == "exclusao")
{
    // realiza a exclusão do produto
    echo "Produto excluído!";
}
```

Esse programa executa diversas funções, e isso é possível graças ao parâmetro que passamos pelo campo *hidden*. Se não utilizássemos esse campo, teríamos de criar três programas: um para incluir, outro para alterar e outro para excluir produtos.

## Passando informações pela URL

Outra forma de fazer uma comunicação entre as páginas é a passagem de valores pela URL, que utilizamos somente no método *GET* de envio de dados.

Imagine um site que possui um jogo de perguntas e respostas, em que os usuários se cadastram e logo após começam a responder as perguntas. A cada resposta correta o usuário soma 100 pontos. O dono do site resolve fazer a seguinte promoção para estimular os usuários:

“Recomende um amigo para participar do jogo, e ganhe 500 pontos!”

Para isso cada usuário, que possui um número identificador recebido no momento do cadastro, deve enviar um link especial para seus amigos. O link deve possuir o endereço do site e logo após o número identificando o usuário que faz a recomendação. O usuário de identificação 405, por exemplo, deve enviar o seguinte link para seus amigos clicarem:

<http://www.perguntas-respostas.com/index.php?id=405>

Provavelmente antes de se cadastrar no jogo, a pessoa que foi recomendada vai querer ler o regulamento, ver quais são os prêmios que o site oferece etc. E para isso ela navega pelo site. Então vem a pergunta: se a pessoa está navegando pelo site, quando chegar no momento do cadastro, como o sistema vai saber quem foi que recomendou essa pessoa? Quem deve receber os 500 pontos pela recomendação? A solução é utilizar a passagem de informações pela URL.

Dessa forma, todo link que for clicado dentro do site deve passar como parâmetro o identificador recebido, no caso o 405. O identificador poderá passar por várias páginas, até chegar na página de cadastramento. Quando a pessoa efetuar o cadastramento, o identificador do usuário pode ser enviado no formulário do cadastro pelo campo *hidden*, conforme vimos anteriormente. Um programa receberá os dados do formulário, cadastrará esse novo usuário e computará mais 500 pontos para o usuário de identificador 405.

Essa é uma maneira muito útil de trocar informações entre as páginas. Dependendo do tipo de aplicações que você precisa desenvolver, muitas vezes deve-se recorrer a essa alternativa.

## Dividindo o cadastramento de usuários em etapas

Como já foi dito no início deste tópico, muitas vezes os sites possuem formulários com muitos campos a serem preenchidos, e isso leva os desenvolvedores a fazer uma divisão do cadastramento em mais de uma página. Agora vamos ver como podemos dividir um cadastramento e utilizar o campo *hidden* do formulário para passar informações entre as páginas. Os campos que temos são os seguintes: *nome*, *email*, *datanascimento*, *sexo*, *profissao*, *endereco*, *telefone*, *cidade*, *estado*, *cep*, *username*, *senha* e *confirma\_senha*. Dividiremos esses campos em três páginas. Na primeira página colocamos somente os dados pessoais: *nome*, *email*, *datanascimento*, *sexo* e *profissao*. Na segunda página colocamos os campos *telefone*, *endereco*, *cidade*, *estado* e *cep*. Por fim, na terceira página colocamos os dados de identificação no site: *username*, *senha* e *confirma\_senha*. A divisão é mostrada nas figuras a seguir:

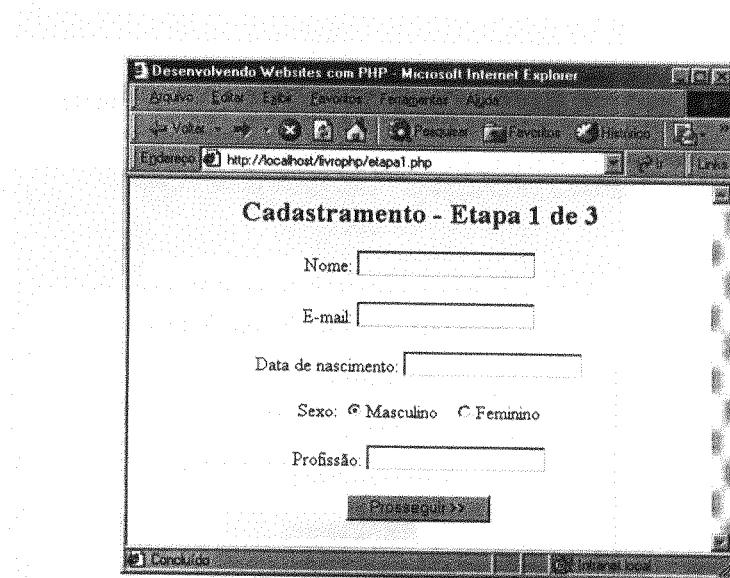


Figura 9.1 – Primeira etapa do cadastramento.

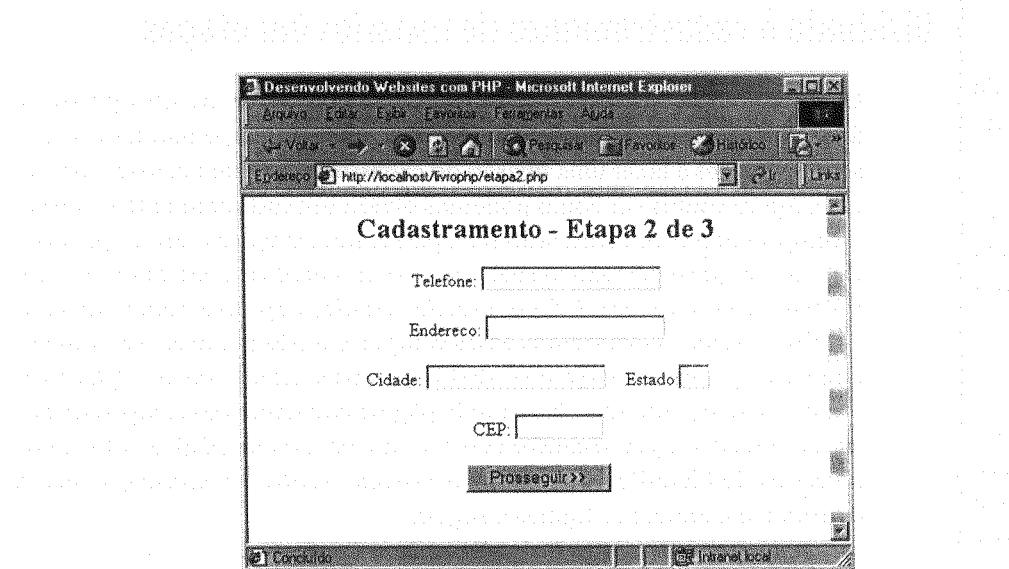


Figura 9.2 – Segunda etapa do cadastramento.

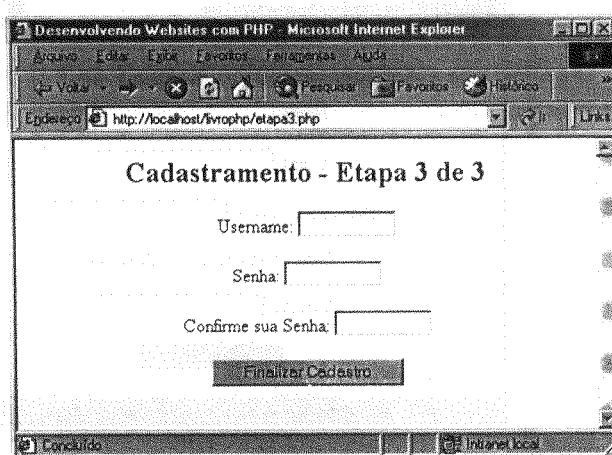


Figura 9.3 – Terceira etapa do cadastramento.

Essas páginas foram nomeadas como *etapa1.php*, *etapa2.php* e *etapa3.php*. É importante lembrar que na etapa 2 devemos receber os dados digitados na etapa 1, e na etapa 3 devemos receber os dados digitados nas etapas 1 e 2. Veja o código de cada uma das páginas, e preste atenção na utilização do campo *hidden* para passagem de dados:

 etapa1.php

(poderia ser html ao invés de php, pois aqui ainda não há código PHP)

```
<html>
<body>
<p align="center"><big><big>Cadastro de Usuário</big></big></p>
<strong>Cadastramento - Etapa 1 de 3</strong></big></big></p>
<form method="POST" action="etapa2.php">
<div align="center"><center><p>
Nome: <input type="text" name="nome" size="20"></p></center></div>
<div align="center"><center><p>
E-mail: <input type="text" name="email" size="20"></p></center></div>
<div align="center"><center><p>
Data de nascimento: <input type="text" name="datanascimento" size="20"></p>
</center></div>
<div align="center"><center><p>
Sexo: <input type="radio" value="masculino" checked name="sexo">
Masculino&ampnbsp&ampnbsp
<input type="radio" name="sexo" value="feminino">Feminino</p></center></div>
<div align="center"><center><p>
Profissão: <input type="text" name="profissao" size="20"></p></center></div>
<div align="center"><center><p>
<input type="submit" value="Prosseguir &gt;&gt;" name="prosseguir"></p>
</center></div>
</form>
</body>
</html>
```

Na opção *action* da tag form indicamos a página *etapa2.php* para receber os dados digitados. A página *etapa2.php*, que também possui um formulário, deve utilizar na opção *action* a página *etapa3.php*. Como a página *etapa2.php* também possui campos a serem preenchidos, quando for ativada a *etapa3.php*, essa página deverá receber os dados fornecidos na primeira e na segunda etapa.

Para isso utilizaremos campos do tipo *hidden* no formulário da segunda etapa. Desse modo estaremos incluindo os dados da primeira etapa com os da segunda. Veja como ficaria o código:

### etapa2.php

```
<html>
<body>
<p align="center"><big><big><strong>
Cadastramento - Etapa 2 de 3</strong></big></big></p>
<form method="POST" action="etapa3.php">
<input type="hidden" name="nome" value="<?php echo $nome; ?>">
<input type="hidden" name="email" value="<?php echo $email; ?>">
<input type="hidden" name="datanascimento" value="<?php echo $datanascimento; ?>">
<input type="hidden" name="sexo" value="<?php echo $sexo; ?>">
<input type="hidden" name="profissao" value="<?php echo $profissao; ?>">
<div align="center"><center><p>
Telefone: <input type="text" name="telefone" size="20"></p></center></div>
<div align="center"><center><p>
Endereço: <input type="text" name="endereco" size="20"></p></center></div>
<div align="center"><center><p>
Cidade: <input type="text" name="cidade" size="20">&ampnbsp&ampnbsp&ampnbsp
Estado: <input type="text" name="estado" size="2"></p></center></div>
<div align="center"><center><p>
CEP: <input type="text" name="cep" size="9"></p></center></div>
<div align="center"><center><p>
<input type="submit" value="Prosseguir >&gt;" name="prosseguir"></p>
</center></div>
</form>
</body>
</html>
```

Observe que após a tag *form* estamos passando todos os campos obtidos na primeira etapa. Isso é feito por meio da opção *value* dos campos de entrada do tipo *hidden*. Logo após estão os campos do formulário que devem ser preenchidos nesta etapa (telefone, cidade etc.). Quando o usuário clicar no botão "Prosseguir", todos os campos serão enviados para a página *etapa3.php*, em que será realizada a última etapa do cadastramento. A página *etapa3.php* poderá ativar uma quarta página, que recebe os dados digitados nas três primeiras e realiza o cadastramento do usuário no banco de dados do site.

Vamos supor que existe a página *cadastra.php*, que recebe todos esses dados e efetua o cadastramento. Então, o código para a página *etapa3.php* seria o seguinte:

 **etapa3.php**

```
<html>
<body>
<p align="center"><big><big><strong>
Cadastramento - Etapa 3 de 3</strong></big></big></p>
<form method="POST" action="cadastra.php">
<input type="hidden" name="nome" value="<?php echo $nome; ?>">
<input type="hidden" name="email" value="<?php echo $email; ?>">
<input type="hidden" name="datanascimento" value="<?php echo $datanascimento; ?>">
<input type="hidden" name="sexo" value="<?php echo $sexo; ?>">
<input type="hidden" name="profissao" value="<?php echo $profissao; ?>">
<input type="hidden" name="telefone" value="<?php echo $telefone; ?>">
<input type="hidden" name="endereco" value="<?php echo $endereco; ?>">
<input type="hidden" name="cidade" value="<?php echo $cidade; ?>">
<input type="hidden" name="estado" value="<?php echo $estado; ?>">
<input type="hidden" name="cep" value="<?php echo $cep; ?>">
<div align="center"><center><p>
Username: <input type="text" name="username" size="10"></p></center></div>
<div align="center"><center><p>
Senha: <input type="text" name="senha" size="10"></p></center></div>
<div align="center"><center><p>
Confirme sua Senha: <input type="text" name="confirma_senha" size="10"></p>
</center></div>
<div align="center"><center><p>
<input type="submit" value="Finalizar Cadastro" name="fim"></p></center></div>
</form>
</body>
</html>
```

Quando o usuário clicar no botão “Finalizar Cadastro”, será ativada a página *cadastra.php*, que receberá os dados digitados desde a primeira etapa, que foram sendo passados por meio dos campos do tipo *hidden*. Essa página será encarregada de cadastrar o usuário no banco de dados do site utilizando as informações recebidas. Veremos mais adiante como fazer inserções em um banco de dados.

É claro que esse é um exemplo mais simples, pois as páginas não fazem nenhuma verificação se os valores digitados nos formulários são válidos ou se algum campo foi deixado em branco. A idéia deste tópico era apenas explicar a passagem de informações entre diversas páginas. Já vimos em um tópico anterior como fazer a validação dos campos de um formulário. Se você preferir, essa verificação também pode ser feita por meio da linguagem Javascript, porém isso não é recomendado. Como o Javascript roda no próprio browser do usuário, alguém pode simplesmente retirar o código de validação e enviar o formulário preenchido incorretamente. Utilizando o PHP não temos esse problema, pois o PHP é *server-based*, ou seja, roda no servidor. Nenhum usuário poderá ver ou alterar códigos PHP de seu site.

Na verdade, é possível fazer a validação dos campos de um formulário usando o PHP, mas é uma tarefa que demanda mais tempo e complexidade. No entanto, é possível fazer isso de forma muito simples. Por exemplo, se quisermos validar se o campo "Nome" está vazio, podemos usar o seguinte código:

```
<?php  
if (empty($_POST['Nome'])) {  
    // O campo 'Nome' está vazio.  
}  
?>
```

Neste código, a função `empty()` verifica se o valor do campo "Nome" é vazio. Se o campo estiver vazio, o bloco de código dentro das chaves será executado. Caso contrário, o bloco de código não será executado. Isso significa que se o campo "Nome" estiver preenchido, o bloco de código dentro das chaves não será executado. Se o campo "Nome" estiver vazio, o bloco de código dentro das chaves será executado. Isso é útil para validar se um campo está preenchido ou não. Por exemplo, se quisermos validar se o campo "Nome" está preenchido, podemos usar o seguinte código:

```
<?php  
if (!empty($_POST['Nome'])) {  
    // O campo 'Nome' está preenchido.  
}  
?>
```

Neste código, a função `!empty()` verifica se o valor do campo "Nome" não é vazio. Se o campo estiver preenchido, o bloco de código dentro das chaves será executado. Caso contrário, o bloco de código não será executado. Isso é útil para validar se um campo está preenchido ou não. Por exemplo, se quisermos validar se o campo "Nome" está preenchido, podemos usar o seguinte código:

```
<?php  
if (isset($_POST['Nome'])) {  
    // O campo 'Nome' está preenchido.  
}  
?>
```

Neste código, a função `isset()` verifica se o campo "Nome" está definido. Se o campo estiver definido, o bloco de código dentro das chaves será executado. Caso contrário, o bloco de código não será executado. Isso é útil para validar se um campo está definido ou não. Por exemplo, se quisermos validar se o campo "Nome" está preenchido, podemos usar o seguinte código:

```
<?php  
if (!empty($_POST['Nome'])) {  
    // O campo 'Nome' está preenchido.  
}  
?>
```

Neste código, a função `!empty()` verifica se o valor do campo "Nome" não é vazio. Se o campo estiver preenchido, o bloco de código dentro das chaves será executado. Caso contrário, o bloco de código não será executado. Isso é útil para validar se um campo está preenchido ou não. Por exemplo, se quisermos validar se o campo "Nome" está preenchido, podemos usar o seguinte código:

```
<?php  
if (trim($_POST['Nome']) != '') {  
    // O campo 'Nome' está preenchido.  
}  
?>
```

Neste código, a função `trim()` remove todos os espaços em branco do valor do campo "Nome". Depois disso, a função `!= ''` verifica se o valor do campo "Nome" não é vazio. Se o campo estiver preenchido, o bloco de código dentro das chaves será executado. Caso contrário, o bloco de código não será executado. Isso é útil para validar se um campo está preenchido ou não.

É importante lembrar que a validação de campos é uma tarefa muito importante para garantir a segurança de seu website. Sempre é recomendável validar os campos de entrada para evitar ataques de hackers. Além disso, é sempre bom validar os campos de entrada para evitar erros de digitação ou入力. Por exemplo, se o usuário digitar "John Doe" no campo "Nome", o bloco de código dentro das chaves não será executado. Se o usuário digitar "John Doe" no campo "Nome", o bloco de código dentro das chaves será executado. Isso é útil para validar se um campo está preenchido ou não.

## Capítulo 10

### PHP e variáveis de ambiente

O próprio nome já diz. Embora possam ser utilizadas nos programas PHP, essas variáveis são do ambiente do servidor Web, e não da linguagem PHP. Com o uso de variáveis de ambiente, podemos descobrir diversas informações úteis, como, por exemplo, qual o tipo de browser que o visitante está utilizando, qual o endereço IP do visitante, de onde foi feita a requisição da página, qual o tipo de conteúdo solicitado etc. Ainda neste tópico você encontra a lista das principais variáveis de ambiente e a descrição de cada uma.

#### Utilizando a função *getenv*

Para obter o valor das variáveis de ambiente, utilizamos a função *getenv* do PHP. Sua sintaxe é a seguinte:

```
getenv ("nome_da_variável");
```

Podemos atribuir o resultado retornado por essa função a uma variável qualquer do PHP. Se quisermos, por exemplo, saber qual foi o método de envio de dados utilizado, podemos realizar a seguinte atribuição:

```
$metodo = getenv ("REQUEST_METHOD");
```

Se tivermos, por exemplo, um formulário que utiliza o método *POST* para o envio de dados, podemos proibir alguém de enviar informações pela URL com o seguinte teste:

```
if ($metodo == "GET")
{
    // Códigos para processar o formulário
}

echo "Erro! O método GET não é aceito para enviar os dados desse formulário!";
}
```

O teste também pode ser feito diretamente com o uso da função, sem que seja feita uma atribuição antes. Exemplo:

```
if (getenv ("REQUEST_METHOD") == "GET")
{
    echo "Erro! O método GET não é aceito para enviar os dados desse formulário!";
}
```

O getenv é uma abreviação para “Get Environment”. Get em português significa obter. Environment significa ambiente. Portanto é por meio da função getenv que podemos acessar essas variáveis especiais denominadas variáveis de ambiente.

## Descobrindo o endereço IP do visitante

Cada máquina conectada na Internet possui um endereço IP que a identifica na rede. Muitas vezes precisamos descobrir o endereço IP de um visitante do site para realizar algumas tarefas.

Como exemplo podemos citar um programa que gerencia uma enquete em um site. O programa exibe uma pergunta e diversas alternativas de resposta. O visitante escolhe uma das alternativas e clica no botão “Votar”. Ao clicar nesse botão, o programa computa o voto do usuário e exibe o resultado parcial da enquete. Mas... e se o usuário tentar votar novamente? Imagine a seguinte enquete: “Qual é o melhor time de futebol do Brasil?”. Então aparece um torcedor fanático do Flamengo e tenta votar 1.000 vezes só para ver seu time na frente.

Para evitar esse tipo de problema, quando o visitante realizar seu primeiro voto, devemos armazenar seu endereço IP em algum lugar (pode ser um banco de dados ou em um simples arquivo-texto). Na próxima vez que ele tentar votar, fazemos a verificação para ver se o IP desse usuário já está cadastrado no banco de dados. Se estiver, exibimos uma mensagem dizendo que não pode votar novamente, pois seu voto já foi registrado.

Para obter o endereço IP de um usuário utilizamos a variável de ambiente *REMOTE\_ADDR*, como é mostrado a seguir:

```
$ip = getenv("REMOTE_ADDR");
```

Outro local em que o endereço IP é utilizado como forma de controle são em salas de bate-papo (chat). Se um usuário entra em uma sala de conversação, e começa a digitar palavrões e xingar todos os outros, o administrador do chat pode proibi-lo de acessar a sala, bloqueando seu endereço IP.

Também podemos citar alguns concursos existentes na Internet que elegem mensalmente os melhores sites do país, com um selo colocado em cada site participante. Os visitantes clicam nesse selo e dão o voto para seu site preferido. Para não permitir que sejam realizados vários votos consecutivos pela mesma pessoa, o sistema desenvolvido deve verificar o endereço IP de quem está votando, para ver se esta pessoa está autorizada a votar.

Portanto é importante que você saiba utilizar o endereço IP do usuário como um mecanismo de controle para manter a confiabilidade das informações existentes em seu site, além de identificar usuários que de alguma forma tentam desrespeitar determinados regulamentos.

## Listas das variáveis de ambiente

Já vimos que a função `getenv` é responsável por retornar os valores das diversas variáveis de ambiente para um programa PHP. Veja na tabela a seguir a lista das principais variáveis de ambiente existentes:

| Variável                       | Descrição   |
|--------------------------------|---|
| <code>SERVER_SOFTWARE</code>   | Servidor e softwares utilizados (ex: Apache, PHP, etc.).  |
| <code>SERVER_NAME</code>       | Nome do servidor (hostname) ou endereço IP.   |
| <code>GATEWAY_INTERFACE</code> | Versão da especificação CGI que o servidor fornece.   |
| <code>SERVER_PROTOCOL</code>   | Nome e versão do protocolo utilizado (ex: HTTP/1.1).  |
| <code>SERVER_PORT</code>       | Número da porta para a qual a requisição foi feita.   |
| <code>REQUEST_METHOD</code>    | O método de envio de dados utilizado (ex: GET, POST).   |
| <code>PATH_INFO</code>         | Informação extra de caminho dada pelo cliente para acesso de alguns scripts.  |
| <code>PATH_TRANSLATED</code>   | Versão traduzida do <code>PATH_INFO</code> . Faz um mapeamento de virtual para físico.                                    |
| <code>SCRIPT_NAME</code>       | Caminho virtual para o script que está sendo executado. Usado por páginas que referenciam elas mesmas.                    |
| <code>QUERY_STRING</code>      | Armazena tudo o que vem após o ? em uma URL chamada.  |
| <code>REMOTE_HOST</code>       | Nome do host (hostname) de onde veio a requisição.  |
| <code>REMOTE_ADDR</code>       | Contém o endereço IP do visitante que solicitou a página.   |
| <code>AUTH_TYPE</code>         | Método de autenticação (usado por servidores que suportam autenticação de usuários).                                      |
| <code>REMOTE_USER</code>       | Nome do usuário (se o servidor suporta autenticação).   |
| <code>REMOTE_IDENT</code>      | Se o servidor suporta identificação-padrão RFC 931, esta variável conterá nome do utilizador remoto que faz a requisição. |
| <code>CONTENT_TYPE</code>      | Tipo MIME dos dados enviados. (ex: "text/html").  |
| <code>CONTENT_LENGTH</code>    | Tamanho dos dados (em bytes) recebidos pelo servidor.   |
| <code>HTTP_ACCEPT</code>       | Lista dos tipos MIME que o cliente pode aceitar.  |
| <code>HTTP_USER_AGENT</code>   | Nome e versão do browser utilizado pelo cliente.  |



# Capítulo 11

## Banco de dados: MySQL ou PostgreSQL

Alguns provedores de hospedagem oferecem servidores com suporte ao MySQL, e outros com suporte ao PostgreSQL. Se você mesmo realizar a instalação do sistema operacional (Linux, Windows etc.), dependendo da distribuição, será possível optar na hora da instalação pelo servidor de banco de dados que deve ser instalado. Faremos uma comparação entre esses dois servidores, para que você conheça um pouco mais de cada um deles, e possa optar pelo que mais se adapta às necessidades de seu site.

### Comparação entre MySQL e PostgreSQL

São dois excelentes SGBDs (Sistemas de Gerência de Bancos de Dados) gratuitos que podem ser usados com o PHP. O MySQL está disponível sob a GPL (licença pública GNU), além de possuir uma licença convencional, para quem não quiser estar limitado aos termos da GPL. Já o PostgreSQL está disponível sob a flexível licença BSD.

O MySQL é mais utilizado no desenvolvimento de aplicações onde a velocidade é importante, enquanto que o PostgreSQL se destaca por ser mais robusto e possuir muito mais recursos. Esses recursos tornam o PostgreSQL um pouco mais qualificado do que o MySQL.

Nas últimas versões do MySQL, os desenvolvedores acrescentaram diversos recursos que já existiam no PostgreSQL, como transações (confirmação ou cancelamento de operações realizadas), *triggers* (gatilhos), *stored procedures* (procedimentos armazenados), *views* (visões), lock de linha (bloqueio em nível de linha) e *constraints* (cláusulas de integridade).

No entanto, o PostgreSQL continua sendo mais eficiente em vários aspectos. Possui um sofisticado mecanismo de bloqueio (MVCC), suporta tamanhos ilimitados de linhas, bancos de dados e tabelas (até 16TB), aceita vários tipos de subconsultas, possui mais tipos de dados e conta com um bom mecanismo de *failsafe* (segurança contra falhas, como por exemplo no desligamento repentino do sistema).

Como já foi dito no início desse tópico, a vantagem do MySQL é a velocidade de acesso. Para bases de dados muito grandes, o MySQL faz um acesso mais rápido que o PostgreSQL. Portanto se seu site possuir um banco de dados muito grande, vale a pena analisar a possibilidade de usar o MySQL. Para base de dados menores, não há diferença na velocidade de acesso entre os dois SGBDs.

## Outra alternativa: SQLite

Além do MySQL e PostgreSQL, a versão 5 do PHP nos disponibiliza uma alternativa mais simples, porém limitada, para o armazenamento de dados. Trata-se do suporte ao *SQLite*, uma ferramenta SQL embutida. Os programas que utilizarem a extensão *SQLite* do PHP poderão trabalhar com bancos de dados SQL, sem que haja a necessidade de ter um SGBD instalado. A biblioteca *SQLi-te* lê e grava dados diretamente em arquivos de bancos de dados no disco.

O SQLite aceita a maioria das construções do padrão SQL92. Portanto, é uma boa alternativa para quem precisa de velocidade e não necessita de todos os recursos oferecidos pelos SGBDs mais “pesados”. Mais informações sobre essa ferramenta podem ser obtidas no site:

<http://sqlite.org>

Inclusive, nesse endereço você pode fazer o download do programa *sqlite*, que serve para administrar bancos de dados *SQLite* pela linha de comando. Neste capítulo, vamos nos concentrar na criação e manipulação de bancos de dados em MySQL e PostgreSQL. Porém, se você tiver interesse em utilizar o *SQLite*, no próximo capítulo será apresentado um exemplo completo de utilização dessa ferramenta no PHP.

## Como criar um banco de dados

Veremos agora como fazer para criar um banco de dados no MySQL ou no PostgreSQL, para posteriormente começar a criação de tabelas e manipulação de dados dentro desse banco de dados.

## PostgreSQL

Para criar um banco de dados no PostgreSQL, devemos utilizar o comando do Linux *createdb* ou o utilitário *psql* (utilizando o comando *CREATE DATABASE*). Se você não tiver a permissão para a criação de um banco de dados, deve-se solicitar ao seu administrador para que ele crie seu banco de dados inicial.

Para criar, por exemplo, um banco de dados com o nome *bdteste*, na linha de comando do Linux digitamos o seguinte comando:

```
$ createdb bdteste
```

Digitando essa linha estará criado o banco de dados *bdteste*, em que serão armazenadas as informações que você desejar. O PostgreSQL possui um gerenciador de conexões chamado *postmaster*, que deve estar rodando para que o banco de dados aceite as conexões solicitadas.

Para acessar o banco de dados criado e começar a inserir as informações utilizamos o utilitário *psql*, que é um gerenciador que acompanha o PostgreSQL. Devemos digitar *psql* seguido do nome do banco de dados que desejamos acessar:

```
$ psql bdteste
```

Ao digitarmos esse comando será mostrada uma tela para o gerenciamento do banco de dados, em que poderemos criar novas tabelas, inserir informações nas tabelas, realizar consultas e efetuar qualquer outra operação suportada pelo PostgreSQL.

Devemos recorrer ao *psql* quando desejamos saber quais são os bancos de dados que já foram criados e quais as tabelas e índices existentes em cada um. Veja a seguir a tela mostrada quando acessamos o banco de dados *bdteste* utilizando o *psql*:

```
Welcome to psql 7.3.4, the PostgreSQL interactive terminal.  
Type: \copyright for distribution terms  
      \h for help with SQL commands  
      \? for help on internal slash commands  
      \g or terminate with semicolon to execute query  
      \q to quit  
bdteste=#
```

Digitando *\?* você tem acesso a todas as opções oferecidas pelo gerenciador, tais como visualização do nome das tabelas existentes (*\dt*), visualização dos índices (*\di*), listagem dos bancos de dados existentes (*\l*) e diversas outras opções. Você ainda pode utilizar a opção *\h* para obter ajuda na sintaxe de comandos SQL.

## MySQL

Para criar um banco de dados no MySQL, utilizamos o utilitário *mysql*. Assim como no PostgreSQL, se você não tiver a permissão para a criação de um banco de dados, deve-se solicitar ao seu administrador para que ele crie seu banco de dados inicial.

Caso você tenha permissão para criação de novos bancos de dados, basta digitar:

```
mysql  
> create database bdteste;  
> use bdteste;
```

A utilização do comando *use* é uma das diferenças em relação ao PostgreSQL. No MySQL devemos primeiramente executar o utilitário gerenciador, para depois escolher o banco de dados que será utilizado, e isso é feito por meio do comando *use*.

Se a criação do banco de dados foi feita por seu administrador, provavelmente você terá um nome de usuário e uma senha para acesso à sua base de dados.

Para acessá-la, você deverá digitar o seguinte comando:

```
mysql -u username -p
```

Por exemplo, supondo que no Linux o *mysql* esteja localizado em */usr/bin* e você queira se conectar com o usuário *root*, bastaria digitar:

```
/usr/bin/mysql -u root -p
```

A opção *-u* indica que o parâmetro seguinte é o nome de usuário utilizado para o acesso, e a opção *-p* indica que será digitado um *password* (senha) para conexão. Após a digitação da senha, você já estará conectado ao banco de dados criado, podendo inserir e manipular livremente seus dados.

Toda a manipulação de dados deve ser feita por meio de comandos SQL (Structured Query Language). Veremos ao longo deste capítulo os principais comandos SQL que podemos utilizar.

## Tipos de dados aceitos pelo MySQL e PostgreSQL

Existem vários tipos de dados comuns entre o PostgreSQL e o MySQL, porém há alguns que estão disponíveis em apenas um deles. Vamos ver separadamente os tipos de dados aceitos em cada um desses SGBDs.

## PostgreSQL

Existe uma grande variedade de tipos aceitos pelo PostgreSQL. A tabela a seguir descreve todos eles:

| Tipo            | Descrição   |
|-----------------|---|
| bit(n)          | String de bits (0s e 1s) com tamanho fixo.  |
| varbit(n)       | String de bits com tamanho variável.  |
| bool            | É o único tipo lógico. Utiliza 1 byte e pode armazenar o valor verdadeiro (t) ou falso (f).   |
| box             | Utiliza 32 bytes para armazenar um quadrado definido pelas coordenadas de seus vértices opostos $((x_1,y_1),(x_2,y_2))$ .   |
| bytea           | Dados binários.   |
| char(n)         | Seqüência de caracteres com tamanho fixo.   |
| cidr            | Armazena endereços de redes IPv4 ou IPv6. O formato é $x.x.x.x/y$ , onde $x.x.x.x$ é o endereço da rede e $y$ o número de bits na máscara de rede.  |
| circle          | Utiliza 24 bytes para armazenar um círculo definido pelo seu centro $(x,y)$ e seu raio $(r)$ . A sintaxe é $<(x,y),r>$ .  |
| date            | Utiliza 4 bytes para armazenar uma data. Pode variar de 13/11/4713 a.C. até 31/12/32767 d.C.  |
| float4, real    | Ponto flutuante que utiliza 4 bytes para armazenamento (6 casas decimais de precisão).  |
| float8          | Ponto flutuante que utiliza 8 bytes para armazenamento (15 casas decimais de precisão).   |
| inet            | Armazena o endereço IP de um ponto da rede, no padrão CIDR. O formato é $x.x.x.x/y$ . Se $y$ for omitido, será considerado 32.  |
| int8, bigint    | Inteiro que utiliza 8 bytes para armazenamento (intervalo varia de -9223372036854775808 a 9223372036854775807).   |
| int, int4       | Inteiro que utiliza 4 bytes para armazenamento (intervalo varia de -2147483648 a +2147483647).  |
| interval(p)     | Utiliza 12 bytes para armazenar um intervalo de tempo, podendo variar de -178000000 anos até +178000000 anos.   |
| line            | Utiliza 32 bytes para armazenar uma linha infinita, definida pelas coordenadas de seus dois pontos $((x_1,y_1),(x_2,y_2))$ .  |
| lseg            | Utiliza 32 bytes para armazenar uma linha finita, definida pelas coordenadas de seus pontos extremos $((x_1,y_1),(x_2,y_2))$ .  |
| macaddr         | Endereço físico de um host (MAC).   |
| numeric[(p, s)] | Valor numérico exato, com precisão e escala definida pelo usuário.  |
| path            | Define um caminho fechado, utilizando os pontos $((x_1,y_1),(x_2,y_2),\dots,(x_n,y_n))$ fornecidos. Para definir um caminho aberto, usa-se os colchetes $[(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)]$ . Utiliza 16+16n bytes. |
| point           | É um dado geométrico. Utiliza 16 bytes para armazenar um ponto no espaço definido por suas coordenadas $(x,y)$ .  |
| polygon         | Armazena um polígono definido por suas coordenadas $((x_1,y_1),(x_2,y_2),\dots,(x_n,y_n))$ . Utiliza 40+16n bytes.  |

| Tipo                | Descrição (cont.)   |
|---------------------|---|
| smallint, int2      | Inteiro que utiliza 2 bytes para armazenamento (intervalo varia de -32768 a +32767).  |
| serial              | Campo de autoincremento. Utiliza 4 bytes para o armazenamento.  |
| serial8             | Campo de autoincremento. Utiliza 8 bytes para o armazenamento.  |
| text                | Seqüência variável de caracteres, sem tamanho máximo.   |
| time [(p)]          | Utiliza 8 bytes para armazenar uma hora do dia. Pode variar de 0:00:00.00 a 23:59:59.99. O parâmetro opcional <i>p</i> indica o número de dígitos fracionários do campo segundos. |
| timetz [(p)]        | Utiliza 12 bytes para armazenar uma hora do dia, incluindo o <i>time zone</i> .   |
| timestamp [(p)]     | Utiliza 8 bytes para armazenar data e hora.   |
| timestampz [(p)]    | Utiliza 8 bytes para armazenar data e hora, incluindo o <i>time zone</i> .  |
| varchar( <i>n</i> ) | Seqüência de caracteres com tamanho variável. O <i>n</i> define o tamanho máximo do campo.  |

## MySQL

Os tipos aceitos pelo MySQL são os seguintes:

| Tipo               | Descrição   |
|--------------------|---|
| tinyint [(M)]      | Inteiro pequeno. Varia de -128 até +127.  |
| bit, bool, boolean | Sinônimo de tinyint(1).   |
| smallint [(M)]     | Inteiro variando de -32768 até +32767.  |
| mediumint [(M)]    | Inteiro variando de -8388608 até +8388607.  |
| int [(M)]          | Inteiro normal, variando de -2147483648 até +2147483647.  |
| bigint [(M)]       | Inteiro grande, variando de -9223372036854775808 até +9223372036854775807.  |
| float (precisão)   | Ponto flutuante com precisão definida.  |
| float [(M,D)]      | Ponto flutuante de precisão simples. Pode armazenar os seguintes intervalos: -3.402823466E+38 até -1.175494351E-38, 0, e 1.175494351E-38 até 3.402823466E+38.                 |
| double [(M,D)]     | Ponto flutuante normal. Os valores permitidos são de 1.7976931348623157E+308 até -2.2250738585072014E-308,0, e também de 2.2250738585072014E-308 até 1.7976931348623157E+308. |
| real [(M,D)]       | É um sinônimo para double.  |
| decimal [(M,D)]    | Esse tipo pode ser usado como uma string. Cada número corresponde a um caractere.   |
| date               | Armazena uma data no formato yyyy-mm-dd. Pode variar de 1000-01-01 até 9999-12-31.  |
| datetime           | Armazena data e hora. O intervalo suportado é de 1000-01-01 00:00:00 até 9999-12-31 23:59:59.   |

| Tipo                | Descrição (cont.)   |
|---------------------|---|
| timestamp [(M)]     | Armazena a data em formato-padrão UNIX (número de segundos após a data base).   |
| time                | Armazena um tempo. Varia de -838:59:59 até 838:59:59.   |
| year [(2 4)]        | Armazena um ano com 2 ou 4 dígitos. O padrão é 4 dígitos. Varia entre 1901 e 2155 no formato e 4 dígitos, e de 1970 a 2069 no formato de 2 dígitos. |
| char(M)             | String de tamanho fixo.   |
| varchar(M)          | String de tamanho variável. Possui no máximo 255 caracteres.  |
| tinytext            | Texto contendo até 255 caracteres.  |
| tinyblob            | Objeto BLOB com tamanho máximo de 255 caracteres.   |
| text                | Texto com até 65535 caracteres.   |
| blob                | Objeto BLOB com até 65535 caracteres.   |
| mediumtext          | Texto com o tamanho máximo de 16777215 ( $2^{24} - 1$ ) caracteres.   |
| mediumblob          | Objeto BLOB com o tamanho máximo de 16777215 ( $2^{24} - 1$ ) caracteres.   |
| longtext            | Texto com o tamanho máximo de 4294967295 ( $2^{32} - 1$ ) caracteres.   |
| longblob            | Objeto BLOB com o tamanho máximo de 4294967295 ( $2^{32} - 1$ ) caracteres.   |
| enum ('valor1',...) | String que pode conter apenas um dos valores listados ou NULL. Podem ser definidos até 65535 valores.   |
| set ('valor1',...)  | String que pode conter 0 ou mais valores, cada um deles pertencente à lista informada. No máximo 64 valores podem ser digitados.                    |

Observações:

- M representa o tamanho máximo para exibição.
- D representa o número de dígitos após o ponto decimal.
- Os colchetes indicam as partes opcionais.

## Como criar tabelas em um banco de dados

Após a criação do banco de dados, podemos acessá-lo (com o *psql* para PostgreSQL ou o *mysql* para MySQL) e começar a criar as tabelas que armazenarão os dados. Isso é feito por meio do comando *CREATE TABLE*, que possui a seguinte sintaxe:

```
CREATE TABLE <nome_tabela> (
    <nome_campo> tipo_de_dado [NULL | NOT NULL]
        [DEFAULT valor_padrão], ...
);
```

Foram apresentadas apenas as opções mais utilizadas do comando *CREATE TABLE*. No manual do SGBD, você verá uma descrição completa do comando, incluindo chaves primárias e outras opções. Veja o significado de cada uma das partes desse comando:

| Parte                  | Descrição   |
|------------------------|---|
| <i>nome_tabela</i>     | Representa o nome da tabela que será criada. Não pode haver nomes de tabelas repetidos.                                   |
| <i>nome_campo</i>      | Representa o nome pelo qual o campo será referenciado.  |
| <i>tipo_de_dado</i>    | Deve ser substituído por um dos tipos apresentados no tópico anterior.  |
| <i>NULL   NOT NULL</i> | Define se o campo pode aceitar valores nulos ou não.  |
| <i>DEFAULT</i>         | Define um valor-padrão para inserções na tabela. Esse valor será utilizado se nenhum valor para este campo for informado. |

Vamos agora utilizar uma loja virtual para exemplificar a criação de tabelas. Precisaremos de uma tabela para armazenar as informações dos produtos. Suponha que as informações referentes aos produtos são:

| Código | Nome | Descrição | Preço | Peso | Categoria | Subcategoria | Inf. adicionais |
|--------|------|-----------|-------|------|-----------|--------------|-----------------|
|        |      |           |       |      |           |              |                 |

Precisaremos também criar uma tabela para armazenar as categorias existentes na loja. Para essa tabela as informações necessárias são:

| Código da Categoria | Nome da Categoria | Código da Subcategoria | Nome da Subcategoria | Código da Categoria a que pertence |
|---------------------|-------------------|------------------------|----------------------|------------------------------------|
|                     |                   |                        |                      |                                    |

Teremos também que armazenar as subcategorias da loja em uma tabela, e para isso precisaremos dos seguintes dados:

Com o comando *CREATE TABLE* vamos criar três tabelas, que serão nomeadas como produtos, categorias e subcategorias. Observe os comandos a seguir, que devem funcionar tanto no MySQL como no PostgreSQL.

```
CREATE TABLE produtos (
    codigo_produto smallint NOT NULL,
    nome_produto varchar(80) NOT NULL,
    descricao_produto text,
    preco float NOT NULL,
    peso float,
    cod_categoria smallint NOT NULL,
    cod_subcategoria smallint NOT NULL,
    adicionais text
);
CREATE TABLE categorias (
    codigo_categoria smallint NOT NULL,
    nome_categoria varchar(60) NOT NULL
);
CREATE TABLE subcategorias (
    codigo_subcategoria smallint NOT NULL,
    nome_subcategoria varchar(60) NOT NULL,
    codigo_categoria smallint NOT NULL
);
```

Os campos seguidos da cláusula NOT NULL não aceitarão valores nulos no momento da inserção no banco de dados. Após a execução dos três comandos apresentados, estarão criadas as tabelas produtos, categorias e subcategorias.

A qualquer momento podemos visualizar o nome das tabelas que já criadas, assim como suas estruturas. Isso é feito por meio do terminal gerenciador que estivermos utilizando (*mysql* ou *psql*).

## Visualizando com o mysql

Utilizando o gerenciador do MySQL (*mysql*), para mostrar o nome das tabelas criadas digitamos:

```
show tables;
```

E para visualizar a estrutura de determinada tabela, basta digitar:

```
describe <nome_tabela>;
```

ou a sintaxe alternativa:

```
desc <nome_tabela>;
```

Devemos substituir `<nome_tabela>` nome da tabela da qual você deseja ver a estrutura. Por exemplo:

```
describe produtos;
```

Você verá uma tela com os nomes dos campos, o tipo, se aceitam valores NULL, os valores-padrão e outras características da tabela consultada.

## Visualizando com o psql

Utilizando o gerenciador do PostgreSQL (psql), para mostrar o nome das tabelas já criadas digitamos `\dt`.

Se utilizarmos somente a opção `\d`, o gerenciador mostrará o nome de todas as tabelas, índices e seqüências existentes nesse banco de dados. Para consultar individualmente cada uma dessas estruturas, utilizamos as opções `\dt`, `\di` e `\ds`.

Para visualizar a estrutura de uma tabela digitamos:

```
\d <nome_tabela>
```

Para visualizar, por exemplo, a tabela dos produtos digitamos:

```
\d produtos
```

Digitando esse comando você verá uma tela contendo o nome dos campos, o tipo e o tamanho de cada um deles, conforme é mostrado a seguir:

| Table "public.produtos" |                       |  |           |
|-------------------------|-----------------------|--|-----------|
| Column                  | Type                  |  | Modifiers |
| codigo_produto          | smallint              |  | not null  |
| nome_produto            | character varying(80) |  | not null  |
| descricao_produto       | text                  |  |           |
| preco                   | double precision      |  | not null  |
| peso                    | double precision      |  |           |
| cod_categoria           | smallint              |  | not null  |
| cod_subcategoria        | smallint              |  | not null  |
| adicionais              | text                  |  |           |

Agora você já tem condições de criar seu banco de dados e as tabelas que farão parte dele, além de poder visualizar a estrutura atual dessas tabelas e do próprio banco. O próximo passo é aprender a manipular os dados, fazendo inserções, atualizações e exclusões nas tabelas.

## Inserindo informações em um banco de dados

Cada linha de uma tabela de banco de dados é chamada de registro. Para incluir um ou mais registros em uma tabela, utilizamos o comando SQL *INSERT*. Veremos agora como funciona esse comando, para que posteriormente você possa utilizá-lo em seus programas PHP.

## Comando INSERT

É o comando responsável por incluir dados em uma tabela. Existem duas variações para o comando *INSERT*:

```
INSERT INTO <nome_tabela> VALUES (valor1, valor2, ..., valorn);
```

```
INSERT INTO <nome_tabela> (campo1, campo2, ..., campoX)
VALUES (valor1, valor2, ..., valorx);
```

Na primeira variação, os valores digitados no lugar de *valor1*, *valor2*, ..., *valorn* serão incluídos na mesma ordem em que foram definidos os campos no momento da criação da tabela definida por <*nome\_tabela*>. Portanto, se fôssemos incluir um registro na tabela *produtos*, o *valor1* seria armazenado no campo *codigo\_produto*, o *valor2* seria armazenado no campo *nome\_produto*, o *valor3* seria armazenado no campo *descricao\_produto* e assim por diante, até o *valor8*, que seria armazenado no campo *adicionais*. Os campos criados com a cláusula *NOT NULL* devem ser obrigatoriamente preenchidos, caso contrário ocorrerá um erro na execução do comando *INSERT*. É importante lembrar que os valores numéricos não devem ser delimitados por aspas. Já os dados do tipo *char*, *varchar*, *date* e outros devem ser delimitados por aspas simples.

Na segunda variação do comando *INSERT*, os valores *valor1*, *valor2*, ..., *valorx* serão inseridos nos campos *campo1*, *campo2*, ..., *campox* da tabela definida por *<nome\_tabela>*. Os campos da tabela não listados receberão o valor *NULL* ou o valor-padrão (caso exista um). Se os campos não listados foram criados com a cláusula *NOT NULL*, ocorrerá um erro na execução do comando *INSERT*.

Observe o exemplo a seguir, em que utilizamos o comando *INSERT* para fazer a inclusão de um produto na tabela *produtos*:

```
INSERT INTO produtos VALUES (1, 'Camiseta do Grêmio', 'Camiseta com listras verticais nas cores azul, preto e branco');
```

```

89.95, 'Camiseta do Flamengo', 49.95, 5, 2,
1.5,
5, 'Camiseta do Flamengo', 49.95, 5, 2,
1.5,
'Disponível com as seguintes numerações: 3, 5, 9 e 10';

```

Executando esse comando estaremos incluindo nosso primeiro produto da loja, e ele é válido para os dois SGBDs (MySQL e PostgreSQL).

No PostgreSQL, após a execução desse comando será mostrada uma linha do tipo:

```
INSERT 16990 1
```

Isso significa que o registro foi incluído com sucesso, e que o 16990 é uma identificação única do registro dentro do banco de dados. No MySQL, a mensagem exibida após a execução deve ser do tipo:

```
Query OK, 1 row affected (0.05 sec)
```

Note que na tabela produtos existem três campos (*descricao\_produto*, *peso* e *adicionais*) que não foram declarados com a cláusula NOT NULL. Nesse caso podemos utilizar a segunda variação do comando *INSERT*:

```

INSERT INTO produtos
(codigo_produto, nome_produto, preco, cod_categoria, cod_subcategoria)
VALUES (2, 'Camiseta do Flamengo', 49.95, 5, 2);

```

Dessa forma os valores ocuparão os campos conforme a ordem definida, e os campos não listados ficarão com o valor NULL.

Veja agora alguns exemplos de inclusões de categorias e subcategorias que podemos fazer nas tabelas que criamos anteriormente:

```

INSERT INTO categorias VALUES (1, 'Eletrodomésticos');
INSERT INTO categorias VALUES (2, 'Cama, Mesa e Banho');
INSERT INTO categorias VALUES (3, 'Áudio e Vídeo');
INSERT INTO categorias VALUES (4, 'Informática');
INSERT INTO categorias VALUES (5, 'Artigos Esportivos');
INSERT INTO subcategorias VALUES (1, 'Bolas de Futebol', 5);
INSERT INTO subcategorias VALUES (2, 'Camisetas de Futebol', 5);
INSERT INTO subcategorias VALUES (3, 'Tênis e Chuteiras', 5);
INSERT INTO subcategorias VALUES (4, 'Raquetes de Tênis', 5);
INSERT INTO subcategorias VALUES (1, 'DVDs', 3);
INSERT INTO subcategorias VALUES (2, 'CDs', 3);
INSERT INTO subcategorias VALUES (3, 'Aparelhos de Som', 3);

```

Observação: se tivéssemos um campo do tipo *date*, o formato de entrada dependeria da configuração do SGBD. O formato mais usado é o *aaaa-mm-dd*, delimitado por aspas simples (por exemplo: '2004-11-24').

## Alterando um banco de dados

Existem dois tipos de alterações que podemos fazer em um banco de dados: alterar a estrutura de uma tabela ou alterar valores dos registros de determinada tabela. Vamos ver como podem ser feitos esses dois tipos de alteração, por meio dos comandos SQL *UPDATE* e *ALTER TABLE*.

### Comando UPDATE

O comando *UPDATE* realiza alterações nos valores dos registros de determinada tabela. Pode alterar um ou mais registros simultaneamente. Se for necessário, podemos alterar uma tabela inteira, utilizando apenas um comando *UPDATE*. Sua sintaxe é a seguinte:

```
UPDATE <nome_tabela>
   SET campo1=valor1 [, campo2=valor2, ..., campon=valorn]
      [WHERE <condições>];
```

Se a cláusula *WHERE* não for utilizada, a alteração será efetuada em todos os registros da tabela. Portanto, tome muito cuidado para não esquecer essa cláusula quando ela é necessária, pois isso fará com que sejam alterados diversos dados que não deveriam sofrer modificações.

Se quisermos, por exemplo, alterar o código de uma categoria, podemos utilizar o seguinte comando:

```
UPDATE categorias SET codigo_categoria=6 WHERE codigo_categoria=5;
```

Podemos também alterar o código de todos os produtos que pertenciam a essa categoria:

```
UPDATE produtos SET cod_categoria=6 WHERE cod_categoria=5;
```

Sempre após a execução do comando *UPDATE*, o gerenciador do banco de dados utilizado informará quantos registros foram afetados pelo comando.

## Comando ALTER TABLE

Por meio deste comando podemos incluir campos em uma tabela. É possível ainda alterar o nome de um campo ou de uma tabela. Podem ser usadas as seguintes sintaxes:

```
ALTER TABLE <nome_tabela> ADD <nome_campo> tipo_de_dado;
```

```
ALTER TABLE <nome_tabela> RENAME <nome_campo> TO <novo_nome>;
```

```
ALTER TABLE <nome_tabela> RENAME TO <novo_nome_tabela>;
```

Para adicionar, por exemplo, o campo *fabricante* à nossa tabela de produtos, devemos executar a seguinte linha:

```
ALTER TABLE produtos ADD fabricante varchar(50);
```

O campo *fabricante* será incluído como o último campo da tabela *produtos*.

**Importante:** não é possível excluir campos de uma tabela. Se precisarmos excluir um campo, devemos apagar a tabela e criá-la novamente. No próximo tópico veremos como fazer a exclusão de uma tabela.

## Excluindo informações de um banco de dados

Existem dois tipos de exclusões que podemos fazer em um banco de dados: excluir uma tabela ou excluir registros de determinada tabela. Veremos como podem ser feitos esses dois tipos de exclusão, por meio dos comandos SQL *DELETE* e *DROP TABLE*.

### Comando DELETE

Este comando exclui um ou mais registros de determinada tabela. Sua sintaxe é a seguinte:

```
DELETE FROM <nome_tabela> [WHERE <condições>];
```

Se a cláusula *WHERE* não for utilizada, todos os registros da tabela serão excluídos.

Vamos ver alguns exemplos de utilização do comando *DELETE*. Para excluir a categoria Artigos Esportivos da tabela de categorias, podemos digitar o comando a seguir:

```
DELETE FROM categorias where nome_categoria='Artigos Esportivos';
```

Para excluir todos os produtos da categoria Áudio e Vídeo (que possui o código 3), podemos digitar o comando a seguir:

```
DELETE FROM produtos where cod_categoria=3;
```

Para excluir todos os produtos existentes na tabela, basta digitar:

```
DELETE FROM produtos;
```

Sempre após a execução do comando *DELETE*, o gerenciador do banco de dados utilizado informará quantos registros foram afetados pelo comando.

## Comando *DROP TABLE*

Serve para excluir uma tabela do banco de dados. Sua sintaxe é a seguinte:

```
DROP TABLE <nome_tabela1> [ ,<nome_tabela2>, ... ];
```

Quando excluímos uma tabela estamos excluindo também todo o seu conteúdo. Para excluir as tabelas produtos, categorias e subcategorias basta digitar:

```
DROP TABLE produtos, categorias, subcategorias;
```

Existe também o comando *DROP DATABASE*, que serve para excluir um banco de dados inteiro. Porém, para executar esse comando é necessário que o administrador do sistema lhe dê essa permissão.

**Observação:** quando utilizamos o PostgreSQL, podemos também excluir um banco de dados por meio do comando do Linux *destroydb*. Basta digitar esse comando seguido do nome do banco de dados que deve ser excluído.

## Fazendo consultas em um banco de dados

Já vimos como incluir, alterar e excluir informações em uma tabela do banco de dados. Agora veremos como fazer consultas sobre essas informações que estamos manipulando. O comando *SELECT* é responsável pelos diversos tipos de consultas que podem ser feitas. Por essa razão, o *SELECT* é o comando SQL utilizado com maior freqüência na maioria das aplicações que envolvem banco de dados.

## Comando *SELECT*

Executando o comando *SELECT*, podemos selecionar todas as linhas de uma ou mais tabelas, ou apenas uma parte delas. Sua sintaxe mais básica é a seguinte:

```
SELECT <lista_campos> FROM <lista_tabelas> [WHERE <condições>];
```

Se *<lista\_campos>* for substituída por um asterisco (\*), serão retornados todos os campos existentes na(s) tabela(s) em uso. Se a cláusula *WHERE* for omitida, serão mostrados todos os registros das tabelas determinadas em *<lista\_tabelas>*.

Para listar, por exemplo, todas as categorias existentes em nossa tabela *categorias*, basta digitar:

```
SELECT * FROM categorias;
```

Teremos como resultado:

```
bdteste=# select * from categorias;
  codigo_categoria | nome_categoria
-----+-----
      1 | Eletrodomésticos
      2 | Cama, Mesa e Banho
      3 | Áudio e Vídeo
      4 | Informática
      5 | Artigos Esportivos
(5 rows)
```

Essa tela mostra o resultado obtido por meio da execução do comando *SELECT* no gerenciador do PostgreSQL. No MySQL você também verá uma tela parecida com essa. Todos os comandos que vimos até agora funcionam nos dois SGBDs (MySQL e PostgreSQL), já que ambos utilizam a linguagem SQL para manipulação de dados.

Nesse caso existiam apenas dois campos, e como utilizamos o \*, todos foram mostrados. Veja outro exemplo semelhante, em que selecionamos todos os produtos cadastrados na tabela *produtos*:

```
SELECT * FROM produtos;
```

O resultado será:

```
bdteste=# select * from produtos;
  codigo_produto | nome_produto | cod_categoria | cod_subcategoria | descricao_produto | peso | preco
-----+-----+-----+-----+-----+-----+-----+-----+
      1 | Camiseta do Grêmio | 1 | 1 | Camiseta com listras verticais nas cores azul, preto e branco | 89,95
      1 | 1,5 | 5 | 2 | Disponível com as seguintes numerações: 3, 5, 9 e 10 | 1 | 49,95
      2 | Camiseta do Flamengo | 1 | 2 | Camiseta com listras verticais nas cores azul, preto e branco | 89,95
      2 | 5 | 2 | Disponível com as seguintes numerações: 3, 5, 9 e 10 | 1 | 49,95
(2 rows)
```

Todos os campos da tabela *produtos* foram mostrados, e isso causou uma dificuldade na visualização dos resultados, já que os registros não couberam em apenas uma linha da tela. Para ter uma melhor visualização, podemos escolher apenas os campos que nos interessam para serem exibidos. Se quiséssemos, por exemplo, ver o código, o nome e o preço de todos os produtos, bastaria digitar o seguinte comando:

```
SELECT codigo_produto, nome_produto, preco FROM produtos;
```

O resultado será:

```
bdteste=# SELECT codigo_produto, nome_produto, preco FROM produtos;
            codigo_produto | nome_produto | preco
-----+-----+-----+
           1 | Camiseta do Grêmio | 89.95
           2 | Camiseta do Flamengo | 49.95
(2 rows)
```

Agora vamos ver um exemplo do comando *SELECT*, utilizando a cláusula *WHERE* para determinar quais registros devem ser retornados. Se quisermos, por exemplo, obter o nome e a descrição dos produtos que custam mais de R\$ 50,00, executamos o seguinte comando:

```
bdteste=# SELECT nome_produto FROM produtos WHERE preco>50;
```

O único produto retornado é a camisa do Grêmio, que custa R\$ 89,95:

```
bdteste=# SELECT nome_produto FROM produtos WHERE preco>50;
            nome_produto
-----+
           Camiseta do Grêmio
(1 row)
```

Outro exemplo: listar todas as subcategorias pertencentes a uma categoria da loja. Vamos utilizar a categoria Artigos Esportivos, que possui o código 5, e visualizar todas as suas subcategorias. O comando que deve ser utilizado é o seguinte:

```
bdteste=# SELECT nome_subcategoria FROM subcategorias WHERE codigo_categoria=5;
```

Observe o resultado:

```
bdteste=# SELECT nome_subcategoria FROM subcategorias WHERE codigo_categoria=5;
            nome_subcategoria
-----+
           Bolas de Futebol
           Camisetas de Futebol
           Tênis e Chuteiras
           Raquetes de Tênis
(4 rows)
```

O comando *WHERE* deve ser seguido por uma ou mais condições. Essas condições podem conter os operadores de comparação *>*, *<*, *>=*, *<=*, *=* e *<>*. Se houver mais de uma condição a ser analisada, utilizamos os operadores lógicos *AND* e *OR* entre elas. Quando for necessário, podemos utilizar os parênteses para determinar a ordem de avaliação das expressões.

Veja mais alguns exemplos de consultas que podem ser feitas com a utilização do comando *SELECT*:

```
bdteste=# SELECT nome_subcategoria FROM subcategorias WHERE codigo_categoria=3;
```

Esse comando retorna o nome de todas as subcategorias pertencentes à categoria Áudio e Vídeo, que possui o código 3.

```
SELECT nome_categoria, descricao_categoria
FROM categorias
WHERE cod_categoria=3;
```

Esse comando retorna o nome e o preço de todos os produtos pertencentes à categoria Eletrodomésticos, que possui o código 1.

```
SELECT nome_produto, preco FROM produtos
WHERE cod_categoria=1;
```

O comando apresentado retorna o nome e a descrição de todos os produtos que possuem o número 2 como código da subcategoria.

```
SELECT nome_produto, descricao_produto, preco FROM produtos
WHERE cod_categoria=5 AND cod_subcategoria=2;
```

Esse comando retorna o nome, a descrição e o preço de todos os produtos pertencentes à categoria Artigos Esportivos e à subcategoria Camisetas de Futebol.

```
SELECT nome_produto, descricao_produto, preco FROM produtos
WHERE cod_categoria=5 AND cod_subcategoria=2 OR cod_subcategoria=3;
```

O comando apresentado retorna o nome de todos os produtos que pertencem à categoria Artigos Esportivos e à subcategoria Camisetas de Futebol, ou retorna o nome dos produtos que possuem o código da subcategoria igual a 3, independentemente da categoria. Isso ocorre porque não utilizamos os parênteses para informar a ordem de avaliação das expressões. Veja o mesmo exemplo, mas com a utilização de parênteses:

```
SELECT nome_produto FROM produtos
WHERE cod_categoria=5 AND (cod_subcategoria=2 OR cod_subcategoria=3);
```

Esse comando retornará o nome de todos os produtos que pertencem à categoria Artigos Esportivos, e ao mesmo tempo pertencem à categoria Camisetas de Futebol ou Tênis e Chuteiras. A primeira expressão avaliada será a que está entre os parênteses. Portanto, primeiro será verificado se o código da subcategoria é 2 ou 3, e caso seja, será avaliada a segunda expressão, que recupera todos aqueles registros que possuem o código de categoria 5, ou seja, que são da categoria Artigos Esportivos.

```
SELECT codigo_produto, nome_produto FROM produtos
WHERE cod_categoria=2 AND preco<100;
```

Esse comando retornará o código e o nome de todos os produtos que pertencem à categoria Cama, Mesa e Banho, e custam menos o que R\$ 100,00.

```
SELECT nome_produto, preco FROM produtos  
WHERE cod_categoria=4 AND preco>=50 AND preco<=500;
```

O comando apresentado retornará o nome de todos os produtos que pertencem à categoria Informática, e custam a partir de R\$ 50,00, e no máximo de R\$ 500,00. Comandos desse tipo são interessantes quando o usuário deseja, por exemplo, fazer uma pesquisa por faixa de preços.

A linguagem SQL nos oferece um operador de grande utilidade, que é o *LIKE*. Utilizando esse operador podemos fazer consultas que seriam extremamente complicadas de serem realizadas utilizando apenas os operadores lógicos e os de comparação. Podemos descobrir, por exemplo, todos os produtos que começam com a letra “E”, ou todos que terminam com a letra “A”, ou ainda todos que contenham a letra “O”. Com o uso desse operador podemos também buscar por palavras dentro de um campo dos registros existentes em uma tabela.

Imagine um site que tem um banco de dados de veículos usados, onde existe uma tabela chamada *veículos*, que armazena informações como nome do veículo, preço, cor e um campo chamado de *adicionais*, contendo os acessórios que o veículo possui. Então uma pessoa acessa o site, querendo comprar um veículo com ar-condicionado. Para mostrar a ela todos os veículos com ar-condicionado, podemos utilizar o comando *SELECT* com o operador *LIKE*:

```
SELECT * FROM veiculos WHERE adicionais LIKE '%ar condicionado%';
```

O comando *LIKE* é utilizado com o símbolo %, que representa uma seqüência de caracteres, ou com o símbolo \_, que representa um caractere qualquer. Para selecionar, por exemplo, todos os modelos de Kadett existentes no banco de dados de veículos, digitamos o seguinte comando:

```
SELECT nome_veiculo FROM veiculos WHERE nome_veiculo LIKE 'KADETT%';
```

Veja um exemplo utilizando as tabelas que criamos anteriormente:

```
bdteste=# SELECT * FROM categorias WHERE nome_categoria LIKE "%o";  
codigo_categoria | nome_categoria
```

```
+-----+-----+  
| 2 | Cama, Mesa e Banho  
| 3 | Áudio e Vídeo  
+-----+
```

O comando executado retorna todas as categorias cujo nome terminam com a letra “o”, que nesse caso são “Cama, Mesa e Banho” e “Áudio e Vídeo”.

### Outras formas de uso do operador LIKE

| Operador      | Descrição  |
|---------------|--|
| LIKE '_E%     | A letra E está na segunda posição.   |
| LIKE '%O'     | Termina com a letra O.   |
| LIKE 'A%E%O'  | Começa com a letra A, termina com a letra O e possui a letra E no meio.  |
| NOT LIKE '%@% | Retorna aqueles que NÃO contêm o caractere @. Podemos utilizar, por exemplo, para verificar se existem e-mails digitados incorretamente em um banco de dados, já que o caractere @ deve existir em todos os e-mails. |

**Dica:** é importante lembrar que existe a distinção entre as letras maiúsculas e minúsculas. Por isso é interessante que as informações sejam armazenadas no banco de dados sempre em letras maiúsculas ou sempre em letras minúsculas, para facilitar as pesquisas. Por exemplo: no momento do cadastro de um usuário no site, quando formos incluí-lo no banco de dados, podemos utilizar uma função para transformar seu nome para letras maiúsculas, para depois fazer a inclusão. Já vimos uma função que é capaz de transformar para maiúsculos todos os caracteres de uma string.

Quando um usuário digita sua cidade, também seria interessante transformá-la em letras maiúsculas antes de incluir no banco de dados. Se isso não for feito, pessoas que moram em Porto Alegre poderão digitar o nome da cidade de diversas formas:

Porto Alegre

Se não fizermos a transformação para letras maiúsculas, temos vários tipos de escrita diferentes para uma mesma cidade, e isso causaria uma grande confusão na hora de pesquisar os usuários que existem em determinada cidade. Para visualizar os usuários existentes em Porto Alegre, teríamos de digitar o seguinte comando:

```
SELECT * FROM usuarios WHERE cidade='Porto Alegre' OR cidade='PORTO ALEGRE' OR
cidade='porto alegre' OR cidade='Porto alegre' OR cidade='porto Alegre' OR
cidade='POrto Alegre' OR cidade='Porto alegre';
```

Se transformarmos a string em letras maiúsculas antes de incluí-la no banco de dados, bastaria fazer a seguinte consulta para visualizar os usuários de Porto Alegre:

```
SELECT * FROM usuarios WHERE cidade='PORTO ALEGRE';
```

Suponha que existe um banco de dados que armazena os dados dos usuários do site. Esse banco de dados possui uma tabela chamada usuários, com os campos nome, email, cidade e estado. Uma consulta que poderia ser feita é a visualização do nome de todas as cidades existentes no banco de dados. Veja o seguinte comando:

```
SELECT cidade FROM usuarios;
```

Esse comando retorna o nome de todas as cidades, porém haverá muitas repetições, pois se existirem 500 usuários que moram em São Paulo, será mostrado 500 vezes o nome dessa cidade. Do mesmo modo será feito para as demais cidades. Para solucionar esse problema utilizamos a opção *DISTINCT*, da seguinte forma:

```
SELECT DISTINCT (cidade) FROM usuarios;
```

Dessa forma seria retornado apenas uma vez o nome de cada cidade existente no banco de dados. Portanto, utilizamos a opção *DISTINCT* sempre que queremos excluir valores repetidos do resultado de uma consulta.

Agora vamos ver como realizar outros tipos de consultas, como por exemplo, contar o número de registro existentes em uma tabela, somar os valores de determinado campo, obter o valor máximo de determinado campo em uma tabela; etc.

Para contar quantos registros existem, por exemplo, em nossa tabela de produtos, utilizamos a função *COUNT*:

```
SELECT COUNT(*) FROM produtos;
```

Para contar quantos produtos existem na categoria Eletrodomésticos (codigo 1), executamos o seguinte comando:

```
SELECT COUNT(*) FROM produtos WHERE cod_categoria=1;
```

Para somar os valores de determinado campo de uma tabela, utilizamos a função *SUM*. Vamos somar os preços de todos os produtos de nossa loja, executando o seguinte comando:

```
bdteste=# SELECT SUM(preco) FROM produtos;
          SUM
_____
    139,9
(1 row)
```

O resultado retornado foi R\$ 139,9, que é a soma dos preços dos dois produtos cadastrados na tabela, um custando R\$ 89,95 e o outro custando R\$ 49,95.

Para descobrir quanto custa o produto mais caro da loja, podemos utilizar a função *MAX*, que retorna o registro que possui determinado campo de maior valor:

```
bdteste=# SELECT MAX(preco) FROM produtos;
      max
      -----
     89,95
(1 row)
```

Se ao invés da função *MAX* fosse utilizada a função *MIN*, seria retornado o menor valor, ou seja, o preço do produto mais barato da loja.

Outra função que pode ser usada com o comando *SELECT* é a *AVG*(do inglês *average*), que calcula a média dos valores de determinado campo:

```
bdteste=# SELECT AVG(preco) FROM produtos;
      avg
      -----
     69,95
(1 row)
```

Em nosso caso, temos apenas dois produtos cadastrados no banco de dados. Esse comando calculou a média aritmética dos preços desses dois produtos:  $(89,95+49,95)/2 = 69,95$ .

As funções *SUM* e *AVG* devem ser aplicadas somente sobre dados do tipo numérico, caso contrário ocorrerá um erro na execução do comando. Vamos inserir mais alguns produtos e subcategorias em nossas tabelas para estudar outros tipos de consultas que podem ser feitas:

```
INSERT INTO subcategorias VALUES (1, 'Notebooks' , 4);
INSERT INTO subcategorias VALUES (2 , 'Monitores' , 4);
INSERT INTO subcategorias VALUES (1 , 'Refrigeradores' , 1);
INSERT INTO produtos VALUES (
    3,
    'Notebook Compaq Presário',
    'Computador portátil com processador Pentium III 500 MHz',
    3300,
    3.5,
    4,
    1,
    'Com 32MB de memória RAM e HD de 4.3GB' );
```

```
INSERT INTO produtos VALUES (
    4,
    'Monitor Samsung Syncmaster',
    'Monitor colorido de 14 polegadas',
    450,
    8,
    4,
    2,
    'Controle OSD digital por microprocessador, independente de RGB');

INSERT INTO produtos VALUES (
    5,
    'Refrigerador Brastemp',
    'Refrigerador Side by Side, com prateleiras em vidro temperado',
    7500,
    150,
    1,
    1,
    'Porta em aço revestido com esmalte branco, que não deixa marcas de dedos');
```

A linguagem SQL nos oferece a opção *GROUP BY*, que nos permite agrupar os resultados de uma consulta. Se quisermos, por exemplo, saber quantos produtos existem em cada uma das categorias, em vez de digitar um comando com COUNT para cada uma , basta utilizar a opção GROUP BY, e mostrar todas de uma só vez. Veja o exemplo:

```
bdteste=# SELECT cod_categoria,COUNT(*) FROM produtos GROUP BY cod_categoria;
cod_categoria | count
```

O resultado da execução desse comando nos diz que existe 1 produto na categoria de código 1 (Eletrodomésticos), 2 produtos na categoria 4 (Informática) e 2 produtos na categoria 5 (Artigos Esportivos). O *GROUP BY* também pode ser utilizado com a opção *HAVING*, que seleciona alguns registros retornados pelo *GROUP BY*. Para retornar, por exemplo, somente as categorias que possuem 1 único produto cadastrado, digitamos o seguinte comando:

```
SELECT cod_categoria,COUNT(*) FROM produtos  
GROUP BY cod_categoria HAVING COUNT(*)=1;
```

Vamos ver agora como realizar consultas utilizando mais de uma tabela, e fazendo o relacionamento entre elas. Quando utilizamos mais de uma tabela em uma consulta, e essas tabelas possuem campos com o mesmo nome, devemos colocar antes dos nomes dos campos o nome da tabela a que eles pertencem, e logo após um ponto (.). Para referenciar, por exemplo, o campo *cod\_categoria* da tabela *produtos*, devemos utilizar *produtos.cod\_categoria* como referência.

Faremos uma consulta idêntica ao exemplo anterior, em que utilizamos o *GROUP BY* para mostrar o número de ocorrências de produtos em cada código de categoria. Seria interessante se, ao invés do código da categoria, exibíssemos o nome da categoria. Como o nome das categorias está em uma tabela separada, o comando *SELECT* deverá utilizar as tabelas *produtos* e *categorias*. Veja como ficaria o comando:

```
SELECT nome_categoria,COUNT(*) FROM produtos,categorias  
WHERE codigo_categoria=cod_categoria GROUP BY nome_categoria;
```

Veja o resultado gerado pela execução desse comando:

| nome_categoria     | count |
|--------------------|-------|
| Artigos Esportivos | 2     |
| Eletrodomésticos   | 1     |
| Informática        | 2     |

(3 rows)

O teste realizado na cláusula *WHERE(codigo\_categoria=cod\_categoria)* relaciona as tabelas *produtos* e *categorias* através do código da categoria, e a opção *GROUP BY* faz o agrupamento dos nomes das categorias, mostrando o número de ocorrências de produtos dentro de cada uma delas.

Outro exemplo: selecionar o nome dos produtos e das subcategorias a que cada um pertence. Para isso temos de consultar as tabelas *produtos* e *subcategorias*. Essa é uma consulta um pouco confusa de montar, pois envolve três tabelas: *produtos*, *categorias* e *subcategorias*. Veja a seguir qual o comando que realiza essa consulta, para depois analisar uma explicação mais detalhada de cada parte da construção dessa consulta:

```
SELECT nome_produto,nome_subcategoria FROM produtos,subcategorias,categorias  
WHERE cod_subcategoria=codigo_subcategoria AND  
cod_categoria=categorias.codigo_categoria AND  
subcategorias.codigo_categoria=categorias.codigo_categoria;
```

Executando esse comando teremos como resultado uma tabela contendo o nome dos produtos, ao lado do nome da subcategoria no qual estão inseridos. Veja na tela a seguir o resultado obtido:

| codigo_produto | nome_produto               | name_subcategoria    |
|----------------|----------------------------|----------------------|
|                | Camiseta do Flamengo       | Camisetas de Futebol |
|                | Camiseta do Grêmio         | Camisetas de Futebol |
|                | Notebook Compaq Presário   | Notebooks            |
|                | Monitor Samsung Syncmaster | Monitores            |
|                | Refrigerador Brastemp      | Refrigeradores       |
| (5 rows)       |                            |                      |

Agora vamos analisar cada uma das partes do comando apresentado. Após a cláusula *FROM* aparecem os nomes das três tabelas que serão acessadas. Após a cláusula *WHERE*, começa a nossa lista de condições. A primeira condição (*cod\_subcategoria = codigo\_subcategoria*) relaciona as tabelas *produtos* e *subcategorias* por meio do código da *subcategoria*. A segunda condição (*cod\_categoria=categorias.codigo\_categoria*) relaciona as tabelas *produtos* e *categorias* por meio do código da categoria. Note que antes do campo *codigo\_categoria* colocamos o nome da tabela a que pertence esse campo, seguido por um ponto. Isso foi feito porque na tabela *subcategorias* existe um campo com o mesmo nome. Por fim, a terceira condição (*subcategorias.codigo\_categoria = categorias.codigo\_categoria*) relaciona as tabelas *categorias* e *subcategorias*, por meio do código da categoria. Esse teste deve ser feito porque existem diversas subcategorias com o mesmo número, e sendo satisfeita essa condição estaremos selecionando apenas a subcategoria que nos interessa.

## Ordenando os resultados de uma consulta

Para ordenar os registros retornados por uma consulta a um banco de dados, utilizamos a opção *ORDER BY*. Após a opção *ORDER BY* podemos indicar diversos campos da tabela para serem ordenados. Os campos serão avaliados da esquerda para a direita, ou seja, o primeiro critério para a ordenação será o primeiro campo colocado, o segundo critério será o segundo campo, e assim por diante.

Um exemplo seria mostrar em ordem alfabética o nome de todos os produtos da loja:

```
bdteste=# SELECT codigo_produto, nome_produto FROM produtos ORDER BY nome_produto;
      codigo_produto | nome_produto
---------------------+-----
      2 | Camiseta do Flamengo
      1 | Camiseta do Grêmio
      4 | Monitor Samsung Syncmaster
      3 | Notebook Compaq Presário
      5 | Refrigerador Brastemp
(5 rows)
```

Podemos também fazer uma ordenação decrescente dos resultados, e para isso basta utilizar a opção *DESC* após o nome do campo, conforme mostrado a seguir:

```
bdteste=# SELECT codigo_produto, nome_produto FROM produtos ORDER BY nome_produto DESC;
codigo_produto | nome_produto
-----|-----
5 | Refrigerador Brastemp
3 | Notebook Compaq Presario
4 | Monitor Samsung Syncmaster
1 | Camiseta do Grêmio
2 | Camiseta do Flamengo
(5 rows)
```

Veja agora o exemplo seguinte, em que foi colocado mais de um campo após o *ORDER BY*:

```
bdteste=# SELECT codigo_categoria, nome_subcategoria FROM subcategorias
bdteste=# ORDER BY codigo_categoria, nome_subcategoria;
codigo_categoria | nome_subcategoria
-----|-----
1 | Refrigeradores
3 | Aparelhos de Som
3 | CDs
3 | DVDs
4 | Monitores
4 | Notebooks
5 | Bolas de Futebol
5 | Camisetas de Futebol
5 | Raquetes de Tênis
5 | Tênis e Chuteiras
(10 rows)
```

Esse comando utiliza dois critérios para fazer a ordenação. O primeiro é o código da categoria, e o segundo é o nome da subcategoria. Assim, subcategorias que pertencem à mesma categoria dependerão do seu nome para determinar a ordem que elas serão retornadas no resultado da consulta. Note que os quatro últimos registros retornados nessa consulta possuem o código de categoria número 5, e estão ordenados alfabeticamente pelo nome.

## Determinando o número de linhas retornadas

Utilizando a opção *LIMIT*, podemos determinar o número máximo de registros que uma consulta pode retornar. Se, por exemplo, um usuário fizer uma busca pela palavra “CD”, e houver mais de 500 CDs na loja, obviamente não mostraremos os 500 CDs na mesma tela, pois isso demoraria um bom tempo e a página ficaria muito grande.

Poderíamos, então, determinar que devem ser retornados no máximo 10 CDs, utilizando o seguinte comando:

```
SELECT * FROM produtos WHERE nome_produto LIKE 'CD %' LIMIT 10;
```

Dessa forma exibiríamos apenas os 10 primeiros CDs ao usuário, e colocaríamos um link para a próxima página, em que seriam mostrados os próximos 10. Porém, para pegarmos de 10 em 10, devemos utilizar a opção *ORDER BY* para ordenar os registros utilizando algum campo, pois se isso não for feito, a cada consulta os registros aparecerão em uma ordem diferente. Portanto, para retornar em ordem alfabética os 10 primeiros registros dessa consulta utilizamos o comando:

```
SELECT * FROM produtos WHERE nome_produto LIKE 'CD %'  
ORDER BY nome_produto LIMIT 10;
```

O parâmetro *OFFSET* é utilizado em conjunto com o *LIMIT*, para determinar a partir de qual registro a consulta deve retornar. Portanto, para retornar os próximos 10 registros, utilizamos o seguinte comando:

```
SELECT * FROM produtos WHERE nome_produto LIKE 'CD %'  
ORDER BY nome_produto LIMIT 10 OFFSET 10;
```

Então bastaria incrementar o parâmetro *OFFSET* de 10 em 10 para obter todos os produtos que contêm a palavra "CD".

## Gravando os resultados em uma nova tabela

Essa é uma função que está disponível somente no PostgreSQL. O MySQL não oferece esse suporte. Podemos gravar os resultados de uma consulta em uma nova tabela, e isso pode ser feito utilizando o comando *SELECT* com a opção *INTO*.

Vamos criar, por exemplo, uma tabela de preços chamada *tabela\_precos*, contendo o nome e o preço de todos os produtos da loja. Para isso basta utilizar o seguinte comando:

```
SELECT nome_produto, preco INTO tabela_precos FROM produtos;
```

Para ter certeza, que a tabela foi criada, peça ao psql para verificar sua estrutura:

```
bdteste=# \d tabela_precos  
Table "public.tabela_precos"  
Column | Type | Modifiers  
-----+-----+-----  
nome_produto | character varying(80) |  
preco | double precision |
```

Fazendo uma consulta à tabela você verá que todos os nomes e preços foram transferidos para essa nova tabela:

## Utilizando *INSERT* e *SELECT* para inserir registros

Essa é uma funcionalidade que pode ser explorada tanto no MySQL como no PostgreSQL. No tópico anterior criamos uma tabela chamada *tabela\_precos*, que armazena o nome e o preço de todos os produtos da loja.

Agora imagine que foram adicionados mais 50 produtos tabela *produtos* da loja, e esse produtos possuem códigos que variam de 6 a 55. Como fazer para incluir esses 50 produtos em nossa tabela chamada *tabela\_precos*?

Para isso, faremos uma combinação dos comandos *INSERT* e *SELECT*, conforme é mostrado a seguir:

```
INSERT INTO tabela_precos SELECT nome_produto, preco FROM produtos  
WHERE codigo_produto>=6 AND codigo_produto<=55;
```

Dessa forma estamos selecionando o nome e o preço de todos os produtos que possuem o código na faixa de 6 a 55 e incluindo-os na tabela *tabela\_precos*. Logo após, você pode executar um *SELECT \* FROM tabela\_precos*, e verificar que todos esses produtos foram incluídos.

## Criando e utilizando seqüências

Seqüência é um recurso que utilizamos para fazer uma numeração automática de determinado campo de uma tabela. Se você estiver usando o PostgreSQL, a maneira mais fácil de criar uma seqüência é criando um campo do tipo *SERIAL* em uma tabela. Por exemplo:

```
CREATE TABLE produto (  
    codigo SERIAL,  
    nome varchar(70) NOT NULL  
)
```

Se você estiver usando o MySQL, pode criar um campo com a propriedade de auto-incremento, da seguinte forma:

```
CREATE TABLE produto (  
    codigo int NOT NULL AUTO_INCREMENT,  
    nome varchar(70) NOT NULL,  
    primary key(codigo)
```

Veja que, no MySQL, o campo com numeração automática deve ser declarado como chave primária (*primary key*) da tabela, para evitar a gravação de códigos repetidos.

Em seguida, para inserir registros em qualquer uma dessas tabelas, faríamos do seguinte modo:

```
INSERT INTO produto (nome) VALUES ('Raquete de Tênis');
INSERT INTO produto (nome) VALUES ('Taco de Baseball');
```

Note que incluímos apenas o valor para o campo *nome*, pois o campo *codigo* possui numeração automática. Portanto, o produto Raquete de Tênis ficará com o código 1, e o produto Taco de Baseball ficará com o código 2.

No PostgreSQL existe outra forma de criar seqüências, utilizando o comando *CREATE SEQUENCE*. Vamos criar uma seqüência chamada *seq\_teste*:

```
CREATE SEQUENCE seq_teste;
```

Após a criação da seqüência, podemos utilizar as funções *NEXTVAL*, *CURRVAL* e *SETVAL* para administrá-la. A função *NEXTVAL* retorna o próximo valor da seqüência, e incrementa o contador. Exemplo:

```
SELECT NEXTVAL('seq_teste');
```

A função *CURRVAL* retorna o valor atual da seqüência. Exemplo:

```
SELECT CURRVAL('seq_teste');
```

A função *SETVAL* altera o valor do contador da seqüência. Exemplo:

```
SELECT SETVAL('seq_teste', 5);
```



# Capítulo 12

## PHP com banco de dados

Chegou a hora de aprender como utilizar em seus programas PHP todos os conhecimentos adquiridos no tópico anterior sobre banco de dados e comandos SQL. Agora finalmente você aprenderá a dar um destino para as informações que são recebidas por meio de formulários HTML, que vimos anteriormente.

### Conectando com um banco de dados

Antes de acessar um banco de dados e começar a realizar operações, precisamos abrir uma conexão com ele. Para isso, utilizaremos uma função de conexão e uma variável do PHP como ponteiro, para referenciar essa conexão.

O PHP possui funções que trabalham com o PostgreSQL, e outras que trabalham com o MySQL. Vamos ver como fazer a conexão com o banco de dados, mostrando exemplo dos dois SGBDs.

#### MySQL

Em um programa PHP, para fazer a conexão com um banco de dados MySQL utilizamos a função *mysql\_connect*. Sua sintaxe é a seguinte:

```
mysql_connect ([string servidor [, string usuário [, string senha [, bool novo_link [, int flags_cliente]]]])
```

Para fazer, por exemplo, a conexão com o banco de dados *bdteste*, que possui como nome de usuário *teste*, e como senha *teste2*, executamos a seguinte atribuição:

```
$conexão = mysql_connect ("localhost", "teste", "teste2");
```

Se banco de dados estiver localizado em um servidor diferente, basta substituir localhost pelo nome ou endereço IP desse servidor.

Logo após especificamos através do comando `mysql_select_db` qual será o banco de dados utilizado:

```
mysql_select_db ("bdteste");
```

Feito isso, estará aberta a conexão, e poderemos manipular os dados, através de inclusões, exclusões e alterações. Quando não formos mais utilizar uma conexão aberta dentro de um programa, podemos fechá-la com o comando `mysql_close`. Exemplo:

```
mysql_close ($conexao);
```

Para não ter de repetir todos esses comandos em diversas páginas do site, seria interessante a criação de uma include que faz a conexão com o banco de dados. Vamos criar essa include:

#### conecta\_mysql.inc

```
<?php  
$conexao = mysql_connect ("localhost", "teste", "teste2");  
mysql_select_db ("bdteste");  
?>
```

Agora basta colocar uma chamada para essa include em todas as páginas que precisam acessar o banco de dados:

```
include "conecta_mysql.inc";
```

Quando for necessário fazer alguma alteração no servidor, username, senha ou no nome do banco de dados, precisaremos alterar um único arquivo em vez de ter de alterar cada uma das páginas. Lembre-se de que no final de cada página devemos fechar a conexão que foi aberta dentro da include, por meio do comando `mysql_close($conexao)`.

Se a sua versão do MySQL for 4.1 ou superior, para aproveitar todos os seus recursos você deverá usar os comandos da biblioteca "mysqli" (*Improved MySQL*), disponível na versão 5 do PHP. Por exemplo:

- ao invés de `mysql_connect`, use a função `mysqli_connect`,
- ao invés de `mysql_select_db`, use a função `mysqli_select_db`,

Os programas apresentados neste livro estão disponíveis para download no site da Novatec Editora nas duas versões (`mysql` e `mysqli`).

## PostgreSQL

Em um programa PHP, para fazer a conexão com um banco de dados PostgreSQL utilizamos a função `pg_connect`. Sua sintaxe é a seguinte:

```
pg_connect (string string_conexão)
```

Essa função abre uma conexão considerando a string fornecida como parâmetro. Essa string pode possuir os parâmetros *host*, *port*, *tty*, *options*, *dbname*, *user* e *password*. Para fazer, por exemplo, a conexão com o banco de dados *bdteste*, que possui como nome de usuário *postgres*, e como senha *postgres*, executamos a seguinte atribuição:

```
$conexao = pg_connect ("dbname=bdteste port=5432 user=postgres password=postgres");
```

A porta-padrão para conexão com o PostgreSQL é a 5432, mas se for necessário esse número pode ser alterado no momento da instalação. A função `pg_connect` retorna um valor que é armazenado na variável `$conexao`. Se houve algum problema na conexão, será retornado o valor *False*. Portanto, por meio da variável `$conexao` podemos testar se a conexão foi feita com sucesso. Podemos fazer esse teste na mesma linha em que utilizamos o `pg_connect`. Observe:

```
if (!$con= pg_connect ("dbname=bdteste port=5432 user=postgres password=postgres"))  
{  
    echo "Não foi possível estabelecer uma conexão com o banco de dados.";  
}
```

Quando não formos mais utilizar uma conexão aberta dentro de um programa, podemos fechá-la com o comando `pg_close`. Exemplo:

```
pg_close ($conexao);
```

Seria interessante a criação de uma include que faz a conexão com o banco de dados. Vamos criar essa include:

### conecta\_pg.inc

```
<?php  
$str_conexao = "dbname=bdteste port=5432 user=postgres password=postgres";  
if (!$conexao=pg_connect($str_conexao))  
{  
    echo "Não foi possível estabelecer uma conexão com o banco de dados.";  
    exit;  
}  
?>
```

Agora basta colocar uma chamada para essa include em todas as páginas que precisam acessar o banco de dados:

```
include "conecta_pg.inc";
```

Quando for necessário fazer alguma alteração no servidor, username, senha ou no nome do banco de dados, precisaremos alterar um único arquivo em vez de ter de alterar cada uma das páginas.

**Observação:** lembre-se de que no final de cada página devemos fechar a conexão que foi aberta dentro da include, por meio do comando `pg_close($conexao)`.

A principal diferença do PostgreSQL para o MySQL é que no MySQL precisamos primeiro fazer a conexão com o gerenciador, para depois escolher o banco de dados que será utilizado. No PostgreSQL, no momento da conexão já indicamos o nome do banco de dados que vamos acessar.

## Executando comandos SQL em um programa PHP

Depois que efetuamos a conexão com o banco de dados, podemos utilizar todos os comandos SQL que vimos anteriormente para fazer inclusões, alterações, exclusões e consultas. A partir de agora, a diferença entre o MySQL e o PostgreSQL são apenas os nomes dos comandos de cada um, pois a funcionalidade deles é a mesma. Para executar comandos SQL no MySQL utilizamos a função `mysql_query`, e no PostgreSQL utilizamos a função `pg_query`.

### MySQL

Vimos que devemos selecionar por meio do comando `mysql_select_db` um banco de dados para ser utilizado. Em seguida, para executar comandos em um banco de dados MySQL iremos utilizar a função `mysql_query`, que possui a seguinte sintaxe:

```
recurso mysql_query (string consulta [, recurso conexão])
```

Veja um exemplo em que utilizamos a include criada no tópico anterior para fazer a conexão com o banco de dados, e logo após executamos um comando SQL:

#### exemplo12\_1.php

```
<?php  
include "conecta_mysql.inc";  
$resultado = mysql_query ("SELECT * FROM produtos");  
mysql_close($conexao);  
?>
```

Após a execução do comando *mysql\_query*, a variável *\$resultado* conterá um ponteiro para o conjunto de registros retornados pelo comando *SELECT*. Vemos no próximo tópico como exibir esse resultado na tela.

Se a sua versão do MySQL for 4.1 ou superior, você deverá *mysql\_query* em vez de *mysql\_query*.

## PostgreSQL

Para executar comandos em um banco de dados PostgreSQL, utilizamos a função *pg\_query*. Sua sintaxe é a seguinte:

*pg\_query (recurso conexão, string consulta)*

Podemos atribuir a uma variável uma string contendo um comando SQL, e logo após colocar essa string como parâmetro do comando *pg\_query*. Veja um exemplo em que utilizamos a include criada no tópico anterior para fazer a conexão com o banco de dados, e logo após executamos um comando SQL:

### exemplo12\_2.php

```
<?php  
    include "conecta_pg.inc";  
    $sql = "SELECT * FROM produtos";  
    $resultado = pg_query ($conexao, $sql);  
    pg_close($conexao);  
?>
```

Após a execução do comando *pg\_query*, a variável *\$resultado* irá conter um ponteiro para o conjunto de registros retornados pelo comando *SELECT*. Vemos no próximo tópico como exibir esse resultado na tela.

## Exibindo os resultados de comandos SQL

Para tratar as informações retornadas pelos comandos SQL que executamos, existe uma série de funções que podemos utilizar. Entre o MySQL e o PostgreSQL, o que muda são apenas os nomes das funções. A tabela a seguir mostra as principais funções para tratamento dos dados, mostrando a descrição, os nomes das funções em MySQL e as equivalentes em PostgreSQL:

| MySQL                            | PostgreSQL                    | Descrição  |
|----------------------------------|-------------------------------|--|
| <code>mysql_affected_rows</code> | <code>pg_affected_rows</code> | Retorna o número de linhas afetadas por uma operação.        |
| <code>mysql_fetch_array</code>   | <code>pg_fetch_array</code>   | Armazena a linha atual do resultado em um array associativo. |
| <code>mysql_fetch_object</code>  | <code>pg_fetch_object</code>  | Retorna uma linha como um objeto.                            |
| <code>mysql_fetch_row</code>     | <code>pg_fetch_row</code>     | Armazena a linha atual do resultado em um array.             |
| <code>mysql_result</code>        | <code>pg_result</code>        | Retorna uma coluna do resultado.                             |
| <code>mysql_num_rows</code>      | <code>pg_num_rows</code>      | Retorna o número de linhas de uma consulta.                  |
| <code>mysql_num_fields</code>    | <code>pg_num_fields</code>    | Retorna o número de colunas de uma consulta.                 |
| <code>mysql_field_name</code>    | <code>pg_field_name</code>    | Retorna o nome de uma coluna em uma consulta.                |

Vamos ver exemplos utilizando os comandos do MySQL. Se você estiver usando o PostgreSQL, troque o nome dos comandos com base na tabela apresentada, prestando atenção nos parâmetros de entrada. No site da Novatec Editora, os exemplos que veremos estão disponíveis para download, tanto para o MySQL como para o PostgreSQL.

No caso do MySQL, é importante também verificar o número da versão que você está usando. No PHP 5 foi adicionada a biblioteca “*mysql*” (*Improved MySQL*), que deve ser utilizada quando a versão do MySQL for 4.1 ou superior. Nesse caso você teria que utilizar, por exemplo, `mysqli_num_rows()` ao invés de `mysql_num_rows()`, `mysqli_fetch_array()` ao invés de `mysql_fetch_array()`, e assim por diante. Os exemplos que veremos também estão disponíveis para download na versão “*mysql*”.

Antes de prosseguir, é importante que façamos a distinção entre as funções `mysql_affected_rows` e `mysql_num_rows`. A primeira é utilizada nas operações de inserção (*INSERT*), atualização (*UPDATE*) ou exclusão (*DELETE*). A segunda é utilizada em consultas (*SELECT*) para obtermos o número de linhas que foram retornadas.

Veja o exemplo de um programa que apaga todo o conteúdo de uma tabela no MySQL:

### exemplo12\_3.php

```
<?php
include "conecta_mysql.inc";
$resultado = mysql_query ("DELETE FROM produtos");
$linhas = mysql_affected_rows ();
```

```

    mysql_close($conexao);
    echo "<p align=\"center\">Foram excluidos $linhas produtos!</p>";
}
?>

```

A variável `$linhas` conterá o número de registros afetados pelo *DELETE* executado pelo comando `mysql_query`. Do mesmo modo podemos utilizar os comandos *INSERT* e *UPDATE*.

Já no comando *SELECT*, devemos utilizar a função `mysql_num_rows` para descobrir o número de registros retornados pela consulta. A sintaxe é:

`mysql_num_rows (recurso resultado)`

Veja um exemplo:

### exemplo12\_4.php

```

<?php
    include "conecta_mysql.inc";
    $resultado = mysql_query ("SELECT * FROM produtos");
    $linhas = mysql_num_rows ($resultado);
    mysql_close($conexao);
    echo "<p align=\"center\">A consulta retornou $linhas registros!</p>";
?

```

Se em vez de utilizarmos a função `mysql_num_rows`, utilizássemos a `mysql_num_fields`, teríamos o número de campos retornados na consulta. Como a tabela *produtos* possui 8 campos, teríamos como resultado o número 8.

Para obter os valores dos campos retornados por uma consulta, existem diversas alternativas. Uma delas é utilizar a função `mysql_result`, que possui a seguinte sintaxe:

`mysql_result (recurso resultado, int linha [, misto coluna])`

| Parâmetro              | Descrição   |
|------------------------|---|
| <code>resultado</code> | Nome da variável que recebeu o resultado da função <code>mysql_query</code> . |
| <code>linha</code>     | Número da linha que desejamos recuperar (começando em 0).                     |
| <code>coluna</code>    | Número da coluna (começando em 0) ou nome do campo da tabela.                 |

Veja o exemplo a seguir, em que exibimos alguns valores da primeira linha retornada de uma consulta:

### exemplo12\_5.php

```
<?php
include "conecta_mysql.inc";
$resultado = mysql_query ("SELECT * FROM produtos");
$codigo = mysql_result ($resultado , 0 , 0);
$nome = mysql_result ($resultado , 0 , "nome_produto");
$preco = mysql_result ($resultado , 0 , "preco");
mysql_close($conexao);
echo "Código do produto: $codigo <br>";
echo "Nome do produto: $nome <br>";
echo "Preço do produto: $preco";
?>
```

Note que uma vez utilizamos um número para indicar a coluna que devia ser retornada, e em outra utilizamos como índice o nome do campo a ser retornado. A ordem numérica dos campos é a mesma utilizada no momento da criação da tabela.

Esse programa mostra apenas os campos de um dos registros da tabela. Se quisermos, por exemplo, mostrar o nome e o preço de todos os produtos, devemos construir um laço, e dentro dele imprimir os valores, conforme mostra o programa a seguir:

### exemplo12\_6.php

```
<?php
include "conecta_mysql.inc";
$resultado = mysql_query ("SELECT * FROM produtos");
$linhas = mysql_num_rows ($resultado);
for ($i=0 ; $i<$linhas ; $i++)
{
    $nome = mysql_result ($resultado , $i , "nome_produto");
    $preco = mysql_result ($resultado , $i , "preco");
    echo "Nome do produto: $nome <br>";
    echo "Preço: $preco <br>";
}
mysql_close($conexao);
?>
```

Outra forma (mais recomendada) de obter os valores retornados de uma consulta é por meio das funções *mysql\_fetch\_row* e *mysql\_fetch\_array*. Essas funções colocam as linhas retornadas (uma por vez) em um array. A diferença

entre as duas é que a função *mysql\_fetch\_array* permite o acesso aos índices do array por chave associativa (nome dos campos), enquanto a função *mysql\_fetch\_row* utiliza somente índices numéricos. A sintaxe básica destas funções é a seguinte:

```
mysql_fetch_row (recurso resultado)  
mysql_fetch_array (recurso resultado)
```

O exemplo a seguir imprime os valores de alguns campos da tabela *produtos*, que criamos anteriormente:

#### exemplo12\_7.php

```
<?php  
include "conecta_mysql.inc";  
$resultado = mysql_query ("SELECT * FROM produtos");  
$linhas = mysql_num_rows ($resultado);  
for ($i=0 ; $i<$linhas ; $i++)  
{  
    $registro = mysql_fetch_row($resultado);  
    echo "Código do produto: $registro[0] <br>";  
    echo "Nome do produto: $registro[1] <br>";  
    echo "Descrição: $registro[2] <br>";  
    echo "Preço: $registro[3] <br>";  
    echo "Peso: $registro[4] <br>";  
    echo "Informações adicionais: $registro[7] <br><br>";  
}  
mysql_close($conexao);  
?>
```

A variável *\$linhas* recebe o número de produtos existentes na tabela, e é usada como limite para o fim do laço determinado pelo comando *for*. Quando executamos a função *mysql\_fetch\_row*, é criado o array chamado de registro. Por meio dos índices desse array obtemos os valores dos campos retornados pela consulta.

A função *mysql\_fetch\_row* é executada diversas vezes, e a cada execução o array *registro* armazena os valores da linha seguinte do resultado.

Temos ainda a função *mysql\_fetch\_object*, que retorna um objeto que referencia uma linha do resultado. Sua sintaxe é:

```
mysql_fetch_object (recurso resultado)
```

Devemos utilizar essa função da seguinte maneira:

### exemplo12\_8.php

```
<?php
    include "conecta_mysql.inc";
    $resultado = mysql_query ("SELECT * FROM produtos");
    $objeto = mysql_fetch_object ($resultado);
    echo "Nome do produto: " . $objeto->nome_produto . "<br>";
    echo "Preço: " . $objeto->preco;
    mysql_close($conexao);
?>
```

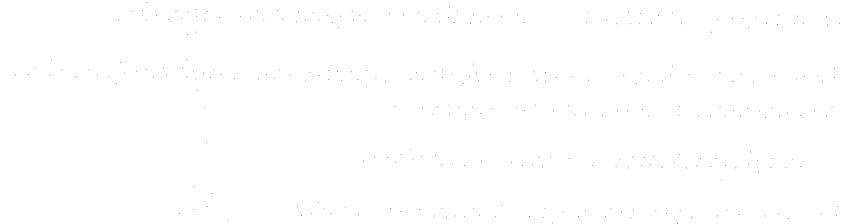
Depois de aprender todas essas funções, você já pode pegar aqueles formulários que vimos em capítulos anteriores, e dar um destino às informações preenchidas neles, como, por exemplo, incluí-las no banco de dados por meio do comando *INSERT*. No próximo tópico veremos maiores detalhes sobre o gerenciamento de um banco de dados, por meio de inclusões, alterações, exclusões e consultas.

## Gerenciando um banco de dados com PHP

Vamos continuar com o exemplo de nossa loja virtual. Em nosso banco de dados existem tabelas de produtos, categorias e subcategorias. Para fazer inclusões, alterações e exclusões seria interessante que existisse uma ferramenta de gerenciamento desse banco de dados. Essa ferramenta poderia utilizar formulários HTML e um programa PHP para receber os dados desses formulários e efetuar as operações solicitadas.

Dessa forma você não precisaria entrar pelo Linux (ou outro sistema operacional) para ativar seu gerenciador do banco de dados e executar comandos SQL sobre ele. Além disso, criando uma ferramenta amigável, qualquer pessoa, sem nenhum conhecimento de comandos SQL poderia manipular as informações de forma facilitada.

Então vamos criar uma tela de administração que permite ao administrador incluir, excluir ou consultar os produtos cadastrados. Utilizaremos um formulário para cada uma dessas funções, conforme mostra a tela a seguir:



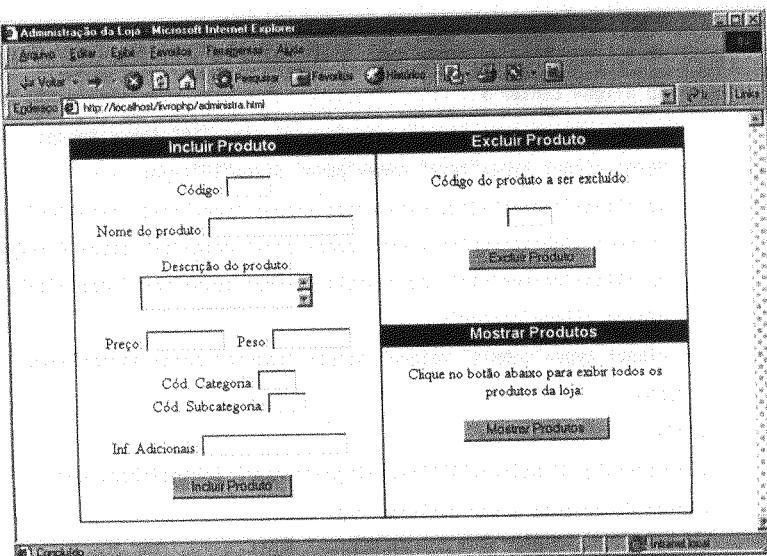


Figura 12.1 – Formulários para o gerenciamento da loja.

O código dessa página, que será chamada de *administra.html*, é apresentado a seguir.

```
<html>
<title>Administração da Loja</title>
<body>
<div align="center"> <center>
<table border="1" cellpadding="0" cellspacing="0" width="85%">
    bordercolor="#008000" height="348">
        <tr>
            <td width="33%" bgcolor="#000080" height="19"><p align="center">
                <font color="#FFFFFF" face="Arial" size="3"><b>Incluir Produto</b></font></p>
            <td width="33%" bgcolor="#000080" height="19"><p align="center">
                <font color="#FFFFFF" face="Arial" size="3"><b>Excluir Produto</b></font></p>
            </td>
        </tr>
        <tr>
            <td width="33%" rowspan="3" valign="top" height="325">
                <form method="POST" action="administra.php">
                    <input type="hidden" name="operacao" value="incluir">
                    <p align="center">&ampnbsp&ampnbsp <br>
                    Código: <input type="text" name="codigo" size="5"></p><p align="center">
```

```
Nome do produto: <input type="text" name="nome" size="20"></p>
<p align="center">Descrição do produto:<br>
<textarea rows="2" name="descricao" cols="20"></textarea></p>
<p align="center">
Preço: <input type="text" name="preco" size="10">&ampnbsp&ampnbsp
Peso: <input type="text" name="peso" size="10"></p>
<p align="center">Cód. Categoria: <input type="text" name="cc" size="4">
<br>Cód. Subcategoria: <input type="text" name="cs" size="4"></p>
<p align="center">Inf. Adicionais: <input type="text" name="ad" size="20">
</p><p align="center">
<input type="submit" value="Incluir Produto" name="enviar"></p>
</form>
</td>
<td width="33%" height="175"><p align="center">&ampnbsp&ampnbsp <br>
Código do produto a ser excluído:</p>
<form method="POST" action="administra.php">
<input type="hidden" name="operacao" value="excluir">
<p align="center"><input type="text" name="codigo" size="5"></p>
<p align="center">
<input type="submit" value="Excluir Produto" name="enviar"></p>
</form>
<p align="center"><br>
&ampnbsp&ampnbsp
</td></tr>
<tr>
<td width="33%" bgcolor="#000080" height="22">
<p align="center"><font color="#FFFFFF" face="Arial" size="3">
<b>Mostrar Produtos</b></font>
</p></td></tr>
<tr>
<td width="33%" height="124">
<p align="center">&ampnbsp&ampnbsp <br>
Clique no botão abaixo para exibir todos os produtos da loja:</p>
<form method="POST" action="administra.php">
<input type="hidden" name="operacao" value="mostrar">
<p align="center">
<input type="submit" value="Mostrar Produtos" name="enviar"></p>
</form>
<p align="center">&ampnbsp</p>
</td></tr>
```

```
</table>
</center>
</div>
</body>
</html>
```

Agora vamos construir o programa *administra.php*, que é responsável por realizar todas as operações solicitadas por meio dos formulários descritos anteriormente: Nesse exemplo, vamos utilizar as funções do MySQL.

```
administra.php
```

```
<?php
$operacao = $_POST["operacao"];
include "conecta_mysql.inc";
if ($operacao=="incluir")
{
$codigo = $_POST["codigo"];
$nome = $_POST["nome"];
$descricao = $_POST["descricao"];
$preco = $_POST["preco"];
$peso = $_POST["peso"];
$cc = $_POST["cc"];
$cs = $_POST["cs"];
$ad = $_POST["ad"];
$sql = "INSERT INTO produtos VALUES ";
$sql .= "('$codigo','$nome','$descricao',$preco,$peso,$cc,$cs,'$ad')";
$resultado = mysql_query ($sql);
echo "Produto incluído com sucesso!";
}
elseif ($operacao=="excluir")
{
$codigo = $_POST[«codigo»];
$sql = «DELETE FROM produtos WHERE codigo_produto=$codigo»;
$resultado = mysql_query ($sql);
$linhas = mysql_affected_rows();
if($linhas==1)
{ echo "Produto excluído com sucesso!"; }
else
{ echo "Produto não encontrado!"; }
}
```

```

elseif ($operacao=="mostrar")
{
    $resultado = mysql_query ("SELECT * FROM produtos");
    $linhas = mysql_num_rows ($resultado);
    echo "<p><b>Lista de produtos da loja</b></p>";
    for ($i=0 ; $i<$linhas ; $i++)
    {
        $reg = mysql_fetch_row($resultado);
        echo "$reg[0] <br>$reg[1] <br>$reg[2] <br>$reg[3] <br>";
        echo "$reg[4] <br>$reg[5] <br>$reg[6] <br>$reg[7] <br><br>";
    }
}
mysql_close($conexao);
?>

```

Veja que o programa *administra.php* está dividido em três partes: inclusão, exclusão e visualização de registros.

Já que os três formulários ativam o mesmo programa, utilizamos o campo *hidden* dos mesmos para indicar ao programa qual operação deve ser realizada. Em cada formulário existe um campo escondido chamado *operacao*, que pode possuir os valores incluir, excluir ou mostrar, conforme o formulário ativado.

O objetivo deste tópico foi mostrar uma simples tela de administração, para facilitar o entendimento da relação entre os formulários e o banco de dados. Agora você pode incrementar o programa adaptando-o às suas necessidades, fazendo verificação dos campos dos formulários e adicionando novas funcionalidades, como, por exemplo, a inclusão e exclusão de categorias e subcategorias.

## Exemplo utilizando a biblioteca SQLite

Conforme citado no capítulo anterior, além do MySQL e do PostgreSQL, na versão 5 do PHP podemos optar por uma forma mais simples e limitada para o armazenamento de dados. Trata-se do suporte à ferramenta SQL embutida chamada *SQLite*, que lê e grava dados diretamente em arquivos de bancos de dados no disco.

Se você precisa de velocidade e não necessita de todos os recursos oferecidos pelos SGBDs mais completos, o *SQLite* é uma boa opção. Todas as informações sobre essa ferramenta podem ser obtidas no site:

<http://sqlite.org>

Veremos agora um exemplo de utilização dessa ferramenta, envolvendo desde a criação do banco de dados até a realização de operações (gravação e leitura) sobre ele.

Primeiramente, é recomendável que você acesse o site <http://sqlite.org> e faça o download do programa *sqlite*, que serve para administrar bancos de dados *SQLite* pela linha de comando. É um programa semelhante ao gerenciadores *mysql* e *psql*, que vimos no capítulo anterior, e será de grande utilidade na criação das tabelas dos seus banco de dados. Esse programa está disponível tanto para Linux como para Windows.

Logo após, iremos criar um banco de dados chamado *bdteste*, digitando na linha de comando do sistema operacional:

```
sqlite bdteste
```

Executando esse comando, será criado um arquivo chamado *bdteste*, que irá armazenar todos as informações (estrutura e conteúdo das tabelas) referentes a esse banco de dados, e será exibido um *prompt* para a digitação de comandos SQL. Exemplo:

```
Enter ".help" for instructions
sqlite>
```

A partir daí, você já pode executar os comandos SQL que estudamos no capítulo anterior. Digitando *.help*, será apresentada uma tela de ajuda, contendo a lista de opções do programa, como por exemplo listar os bancos de dados (*.databases*), tabelas (*.tables*) etc.

O exemplo que veremos consiste na criação de um simples livro de visitas (*guestbook*), onde os visitantes poderão postar suas mensagens contendo comentários sobre o seu site. Para armazenar essas mensagens, crie a seguinte tabela no *sqlite*:

```
CREATE TABLE mensagens (
    nome varchar(80),
    cidade varchar(50),
    estado char(2),
    mensagem text
);
```

Veja que, além da mensagem, serão armazenados nome, cidade e Estado do visitante. O formulário de entrada de dados será o seguinte:

Nome: \_\_\_\_\_  
Cidade: \_\_\_\_\_  
Estado: \_\_\_\_\_  
Mensagem: \_\_\_\_\_

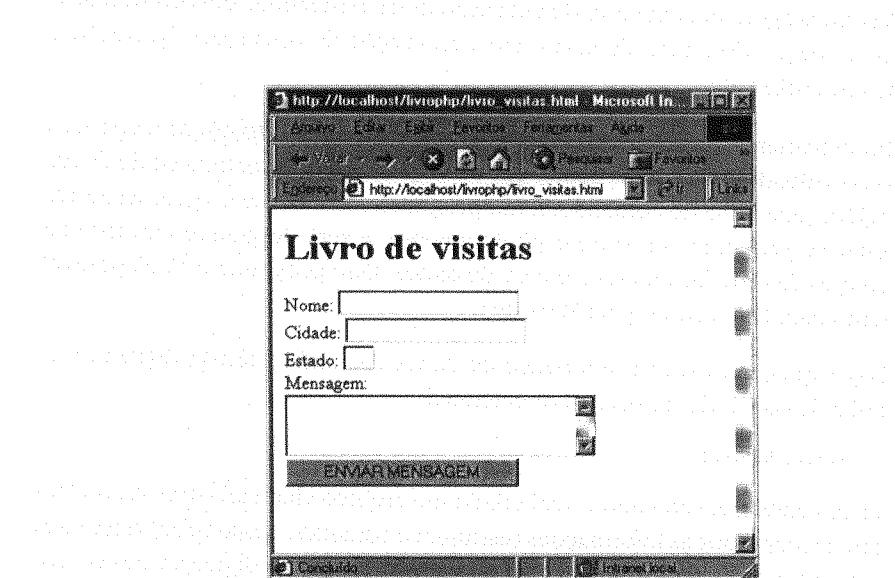


Figura 12.2 – Formulário para inclusão de mensagens.

O código HTML para esse formulário é apresentado a seguir.

```
livro_visitas.html
<html>
<body>
<h1>Livro de visitas</h1>
<form method="POST" action="livro_visitas.php">
    <p>Nome: <input type="text" name="nome" size="20"><br>
    Cidade: <input type="text" name="cidade" size="20"><br>
    Estado: <input type="text" name="estado" size="20"><br>
    Mensagem: <br>
    <textarea rows="3" name="mensagem" cols="30"></textarea><br>
    <input type="submit" value="ENVIAR MENSAGEM" name="enviar"></p>
</form>
<p>&nbsp;</p>
</body>
</html>
```

Conforme especificado na opção *action* da tag *<form>*, os dados desse formulário serão enviados para o programa *livro\_visitas.php*. Esse programa deverá utilizar as funções do PHP/SQLite para incluir a mensagem no banco de dados, e posteriormente mostrar a lista das mensagens já incluídas. As principais funções do PHP disponíveis para manipulação de bancos de dados SQLite são apresentadas a seguir.

| Função                          | Descrição   |
|---------------------------------|---|
| <code>sqlite_open</code>        | Abre um banco de dados SQLite. Se o banco de dados não existir, ele será criado.              |
| <code>sqlite_close</code>       | Fecham um banco de dados SQLite.  |
| <code>sqlite_query</code>       | Executa um comando SQL no banco de dados especificado e retorna um ponteiro para o resultado. |
| <code>sqlite_num_rows</code>    | Retorna o número de linhas resultantes de uma consulta.                                       |
| <code>sqlite_num_fields</code>  | Retorna o número de campos existentes em um resultado.  |
| <code>sqlite_fetch_array</code> | Obtém a próxima linha do resultado, armazenando-a em um array.                                |
| <code>sqlite_changes</code>     | Retorna o número de linhas que foram alteradas pelo último comando SQL executado.             |

A função `sqlite_open`, responsável pela abertura/criação de bancos de dados *SQLite*, possui a seguinte sintaxe:

```
recurso sqlite_open (string nome_arquivo [, int modo [, string
&msg_erro]])
```

O valor padrão para o segundo parâmetro (modo) é 0666. Se o terceiro parâmetro for usado, ele irá armazenar a mensagem referente ao erro ocorrido (se houver falha na abertura). Veja que essa função retorna um ponteiro para o banco de dados aberto, e portanto seu retorno deve ser atribuído a uma variável, que será usada posteriormente nas demais funções. Exemplo:

```
$bd = sqlite_open('bdteste', 0666, $erro);
```

A função `sqlite_query`, utilizada para execução de comandos SQL no banco de dados, pode ser usada de duas formas:

```
recurso sqlite_query (recurso handle_bd, string comandosSQL)
recurso sqlite_query (string comandosSQL, recurso handle_bd)
```

No parâmetro `handle_bd` devemos passar o ponteiro retornado pela função `sqlite_open`. Se o comando SQL retornar linhas, essa função retorna um ponteiro para o resultado (*result set*), que posteriormente pode ser utilizado pela função `sqlite_fetch_array` para leitura dos valores de cada linha. Por exemplo, para retornar todos os registros da tabela *mensagens* e armazenar o primeiro deles em um array, teríamos:

```
$res = sqlite_query($db, 'SELECT * FROM mensagens');
$linha = sqlite_fetch_array($res);
```

Vamos ver então como poderia ser implementado o programa *livro\_visitas.php*, responsável pela gravação de mensagens no banco de dados e exibição de todas as mensagens já gravadas.

O primeiro teste a ser feito no programa é se o usuário forneceu os seus dados (nome, cidade e Estado) e sua mensagem. Se ele forneceu, será realizada a gravação desses dados na tabela, para posteriormente as mensagens serem exibidas. Se ele não forneceu, o programa irá apenas exibir as mensagens já existentes. Será utilizada uma variável de controle (*\$incluir*) para indicar se há uma mensagem a ser incluída. Observe o código apresentado a seguir.

### livro\_visitas.php

```
<html>
<body>
<h1>Livro de visitas</h1>
<?php
$incluir = 0;
if(isset($_POST["nome"]) && isset($_POST["cidade"])
&& isset($_POST["estado"]) && isset($_POST["mensagem"]))
{
    $nome = $_POST["nome"];
    $cidade = $_POST["cidade"];
    $estado = $_POST["estado"];
    $mensagem = $_POST["mensagem"];
    $incluir = 1;
}

if ($db = sqlite_open('bdteste', 0666, $erro))
{
    // grava a mensagem recebida
    if($incluir)
        sqlite_query($db,"INSERT INTO mensagens
VALUES ('$nome','$cidade','$estado','$mensagem')");
    // exibe todas as mensagens
    $res = sqlite_query($db,'SELECT * FROM mensagens');
    $linhas = sqlite_num_rows($res);
    for($i=0; $i<$linhas; $i++)
    {
        $linha = sqlite_fetch_array($res);
        echo "<b>Nome</b>: ".$linha["nome"]. "<br>";
        echo "<b>Cidade</b>: ".$linha["cidade"]. "<br>";
        echo "<b>Estado</b>: ".$linha["estado"]. "<br>";
        echo "<b>Mensagem</b>: ".$linha["mensagem"]. "<hr>";
    }
}
```

```
    sqlite_close ($db);
}
else
die ($erro);
?>
</body>
</html>
```

Se você criou o banco de dados *bdteste* em outro diretório, não esqueça de passar o caminho completo no primeiro parâmetro da função *sqlite\_open*.

Note que o número total de mensagens foi obtido pela função *sqlite\_num\_rows*, e em seguida foi criado um laço para leitura dessas mensagens através da função *sqlite\_fetch\_array*. A figura 12.3 mostra um exemplo de exibição das mensagens gravadas no banco de dados.

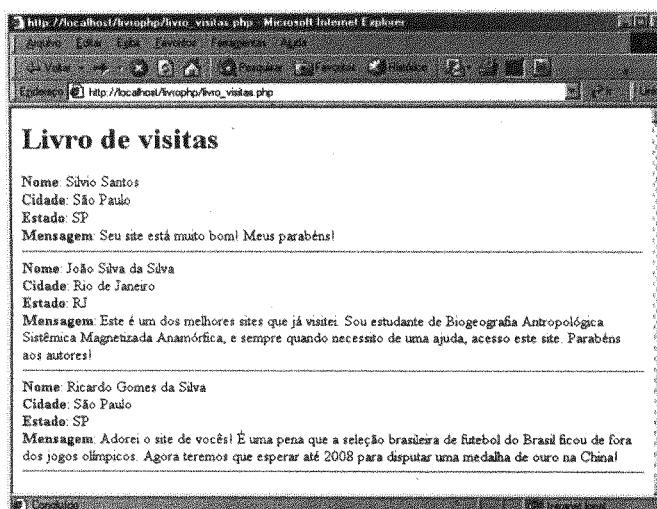


Figura 12.3 – Exibindo as mensagens do livro de visitas.

Na referência das funções, ao final deste livro, você irá encontrar a descrição das demais funções da biblioteca *SQLite*.



# Capítulo 13

## Cookies e sessões

Se você precisa manter informações sobre seus usuários enquanto eles navegam pelo seu site, ou até quando eles saem do mesmo, é importante que você saiba lidar com cookies ou com sessões. Cookies são arquivos-texto que podem ser armazenados no computador do usuário, para serem recuperados posteriormente pelo servidor no qual seu site está hospedado. Sessões são recursos que podemos utilizar para manter uma conexão com o usuário, enquanto ele estiver navegando no site. Veremos com detalhes esses dois mecanismos nos próximos tópicos.

### Algumas utilidades de cookies e sessões

Quando você acessa uma página Web, a comunicação entre o seu programa navegador e o servidor Web é feita através de um protocolo chamado HTTP (Hypertext Transfer Protocol). O problema é que esse protocolo não armazena informações de estado, ou seja, ele trata de forma independente cada requisição de página que recebe.

Por exemplo, se você acessar as páginas chamadas *teste1.html* e *teste2.html*, e outra pessoa acessar essas mesmas páginas, o servidor irá tratar esses acessos como sendo 4 requisições independentes. Ele não irá saber que foram 2 pessoas que acessaram 2 páginas cada. Por essa razão é que os mecanismos de cookies e sessões são importantes, pois eles nos permitem armazenar informações que poderão ser utilizadas enquanto o usuário estiver navegando entre as páginas do nosso site.

Entre algumas utilidades de cookies e sessões, podem ser citadas:

- Autenticação de usuários: criação de um sistema envolvendo login, autenticação e logout, o que garante o acesso do conteúdo somente aos usuários autorizados.

- **Carrinho de compras:** utilizado nos sites de comércio eletrônico para armazenar todos os produtos já selecionados pelo cliente para compra, enquanto ele navega pelo site da loja.
- **Exibição de anúncios ou imagens:** para não exibir mais de uma vez um mesmo anúncio ou imagem para o usuário, é necessário manter informações sobre as que já foram exibidas.
- **Personalização de páginas:** por exemplo, uma livraria virtual poderia exibir o anúncio de um livro de culinária, caso o usuário tivesse feito uma pesquisa pela palavra “culinária” na última vez que acessou o site.

Essas são apenas algumas das aplicações que podem ser implementadas com o uso de cookies ou de sessões.

## Utilizando cookies

Cookie é um arquivo-texto que podemos armazenar no computador do usuário, para ser recuperado posteriormente pelo servidor. Um cookie é formado por um par nome/valor, ou seja, possui um nome pelo qual ele é referenciado e um valor associado a esse nome. Podem ser utilizados em qualquer aplicação que necessite compartilhar dados entre diferentes páginas, ou até entre diferentes acessos (em dias e horários distintos).

Os cookies podem ser mantidos na máquina do usuário por vários dias, ao contrário das sessões, que mantêm os dados somente enquanto o usuário permanecer no seu site.

### Enviando cookies pelo PHP

O PHP nos oferece a função *setcookie*, que envia cookies para o computador do usuário. Essa função é usada tanto para definir um cookie, como também para excluí-lo. Sua sintaxe é a seguinte:

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

| Parâmetro       | Descrição  |
|-----------------|--|
| <i>nome</i>     | Indica o nome do cookie que está sendo enviado, e é o único parâmetro obrigatório para a função.   |
| <i>valor</i>    | É o valor do cookie. Se não for fornecido, o servidor tentará excluir o cookie com o nome especificado.  |
| <i>validade</i> | Define o tempo de validade do cookie. Deve ser expresso no formato-padrão de tempo do Unix (número de segundos após 1º de janeiro de 1970, às 0h). |

| Parâmetro      | Descrição (cont.)   |
|----------------|---|
| <i>caminho</i> | Caminho no servidor para o qual o cookie estará disponível. Se for definido o valor "/", ele estará disponível para todo domínio especificado no parâmetro <i>domínio</i> . O valor padrão é o diretório corrente a partir do qual o cookie foi definido. |
| <i>domínio</i> | Domínio para o qual o cookie estará disponível.   |
| <i>seguro</i>  | É um valor inteiro (0 ou 1), que indica se o cookie é seguro. Se for utilizado o valor 1, o cookie só será transmitido se a conexão for segura (HTTPS).   |

Se for utilizado somente o parâmetro *nome*, o servidor tentará excluir o cookie do computador do usuário. Portanto, para definir um cookie devemos utilizar no mínimo os parâmetros *nome* e *valor*. Exemplo:

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

Para excluir o cookie criado anteriormente basta executar o comando:

```
setcookie ("nome");
```

Para criar, por exemplo, um cookie válido por 2 dias (48 horas) podemos utilizar como auxílio a função *time* do PHP. Exemplo:

```
setcookie ("nome", "Juliano", time()+172800);
```

Esse cookie é válido por 2 dias, pois utilizamos a função *time* para obter o tempo atual, e somamos 172.800 segundos, que equivalem a 48 horas. Esse resultado foi passado como o parâmetro *validade* para a função *setcookie*.

Caso você não queira passar algum parâmetro intermediário, utilize uma string vazia ("") para substituí-lo, com exceção dos parâmetros *validade* e *seguro*, pois estes devem conter valores numéricos. Nesse caso utilize zero (0) em vez de "".

---

Importante: o envio de cookies deve ser a primeira coisa a ser feita na execução da página, ou seja, a função *setcookie* deve ser utilizada antes de qualquer tag HTML, como, por exemplo, as tags <HTML> e <BODY>. Se for utilizada após as tags do HTML, o PHP irá exibir uma mensagem de erro, dizendo que a função foi chamada após o envio dos cabeçalhos.

---

## O array superglobal **\$\_COOKIE**

Agora que você já aprendeu a enviar cookies para o computador do usuário, vamos ver como recuperar esses valores em seus programas PHP. Após definidos, os cookies estarão disponíveis para toda a requisição de páginas feita pelo usuário ao servidor. É importante lembrar que os cookies não poderão ser utilizados dentro da própria página que os criou. Poderemos utilizá-los somente a partir da próxima solicitação de página vindas do browser do usuário.

Existem duas formas de acessar, por meio do PHP, os cookies enviados para a máquina do usuário. Uma delas, e a mais recomendada, é através do array superglobal `$_COOKIE` (em versões mais antigas era chamado de `$HTTP_COOKIE_VARS`). Devemos utilizar o nome do cookie como chave associativa desse array. Por exemplo, se você definiu um cookie chamado *nome*:

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

Na próxima página acessada pelo usuário, esse valor poderia ser acessado da seguinte maneira:

```
$_COOKIE["nome"]
```

A segunda forma de acessar o valor de um cookie funciona apenas se a diretiva `register_globals` estiver habilitada no `php.ini`. Nesse caso, você pode utilizar o próprio nome do cookie como uma variável. Por exemplo, o cookie *nome* seria referenciado pela variável `$nome`.

## Criando um sistema de *username/senha* para seu site

Esses sistemas são utilizados em diversos tipos de sites. Alguns utilizam para proteger um conteúdo que não é gratuito, permitindo o acesso somente de usuários autorizados. Atualmente até alguns sites que possuem um conteúdo gratuito estão exigindo um cadastramento de usuários, a fim de conhecer o perfil das pessoas que os acessam.

Veremos agora como criar um sistema de *username/senha* para seu site, utilizando um banco de dados para armazenar as informações dos usuários e cookies para implementar a autenticação. O sistema será dividido em três partes:

- 1) Página de login, que recebe os dados do usuário, verifica se ele está cadastrado e cria os cookies.
- 2) Rotina de validação para ser utilizada nas páginas que fazem parte das áreas restritas.
- 3) Página de logout para os usuários que estão autenticados (realizar exclusão dos cookies).

A primeira coisa a ser feita é criar uma tabela no banco de dados para armazenar os dados dos usuários. Vamos utilizar como exemplo uma tabela que armazena nome, e-mail, cidade, Estado, nome de usuário (*username*) e senha do usuário. Essa tabela será criada por meio do utilitário *mysql* (ou *psql*, no caso do PostgreSQL), através da digitação do seguinte comando SQL:

```
CREATE TABLE usuarios
(
    username varchar(10) NOT NULL,
    senha varchar(10) NOT NULL,
    nome varchar(80) NOT NULL,
    email varchar(80) NOT NULL,
    cidade varchar(40) NOT NULL,
    estado char(2) NOT NULL,
    primary key(username)
);
```

Repare que utilizamos o campo *username* como chave primária (Primary Key), pois isso nos garante que não poderá haver nenhum *username* repetido em nosso banco de dados. Portanto, cada usuário possuirá uma identificação única, que será armazenada no campo *username*.

Após a criação da tabela, você pode utilizar os conhecimentos adquiridos nos capítulos 8 e 12, para criar um formulário de cadastramento de usuários. Basta-ria criar um formulário com os campos listados anteriormente, e enviar os re-sultados para um programa PHP que realizasse a inclusão na tabela de usuári-os por meio do comando SQL *INSERT*.

Como o objetivo desse tópico é ensinar a autenticação, vamos considerar que o banco de dados já possui alguns usuários cadastrados. Se você ainda não criou um formulário de cadastro e quiser testar a autenticação, pode incluir um registro apenas para fins de teste, digitando diretamente no *mysql* (ou *psql*):

```
INSERT INTO usuarios VALUES
('juliano','teste','JULIANO','juliano@niederauer.com.br','Porto Alegre','RS');
```

## Login

Após a criação da tabela e inclusão de alguns usuários, o próximo passo é fazer um programa de login, que recebe o *username* e a senha de um usuário, verifica se essas informações estão corretas, cria os *cookies* no computador do usuário e o direciona para a página inicial para usuários do site.

Vamos supor que na página inicial de seu site há um formulário com dois cam-pos (*username* e *senha*) para o usuário fazer seu login, conforme mostra a figura 13.1.

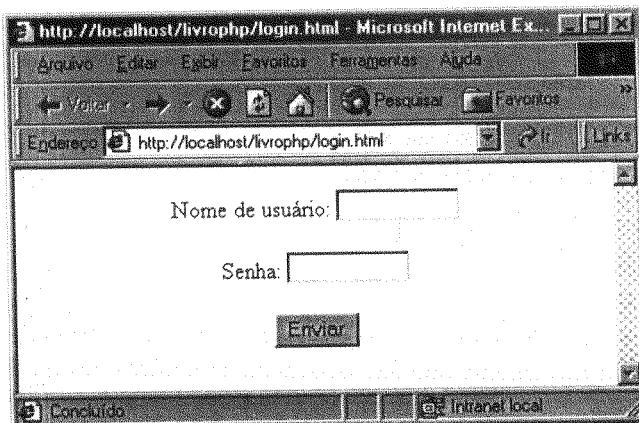


Figura 13.1 – Formulário de login.

O código dessa página, nomeada como *login.html*, é o seguinte:

### login.html

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

Ao clicar no botão “Enviar”, será ativado o programa *login.php*, que deve analisar as informações enviadas e retornar uma resposta para o usuário ou direcioná-lo para outra página.

Vamos agora construir o programa *login.php*, que primeiro deve verificar se o usuário realmente está cadastrado e se a senha digitada está correta, para posteriormente criar os cookies. Para acessar o banco de dados utilizaremos a include *conecta\_mysql.inc*, criada no capítulo anterior. Se você desejar, no site da Novatec Editora (no endereço indicado no início deste livro), você poderá obter esse mesmo programa utilizando as funções do PostgreSQL ao invés do MySQL.

 login.php

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

Veja que primeiro foi feita uma consulta ao banco de dados para verificar se o usuário existe. Se não existir, é exibida a mensagem “*Usuário não encontrado*”.

Se existir, a senha digitada é comparada com a senha gravada no banco de dados e, se elas forem iguais, o login é realizado através da criação dos cookies e direcionamento do usuário à página principal.

Nesse exemplo, foi considerado que a página principal do seu site (depois do login) se chama *pagina\_inicial.php*. Se você for utilizar páginas com outros nomes, apenas faça a substituição desse nome.

### Autenticação

Depois que o usuário fez o login, os cookies já foram armazenados na máquina dele. Agora a cada página que ele for acessar, devemos verificar se os cookies realmente existem e se contêm as informações corretas. Devemos fazer isso para evitar que o usuário digite diretamente o endereço de uma página, sem passar pelo login.

Para fazer essa verificação e validação dos cookies, vamos criar uma *include* chamada *valida\_cookies.inc*, que deve ser chamada por todas as páginas que devem ser protegidas.

Acompanhe então o código apresentado a seguir. Note que nas primeiras linhas, antes de obter o valor dos cookies, testamos se eles realmente estão definidos através da função *IsSet* do PHP. É recomendável fazer esse teste para evitar que o PHP exiba alguma mensagem de erro ao tentarmos acessar um elemento que não existe no array *\$\_COOKIE*.

#### valida\_cookies.inc

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

```

if($senha_usuario != mysql_result($resultado,0,"senha"))
{
    setcookie("nome_usuario");
    setcookie("senha_usuario");
    echo "Você não efetuou o LOGIN!";
    exit;
}
else
{
    setcookie("nome_usuario");
    setcookie("senha_usuario");
    echo "Você não efetuou o LOGIN!";
    exit;
}
else
{
    echo "Você não efetuou o LOGIN!";
    exit;
}
mysql_close($conexao);
?>
```

Esse programa verifica a existência e a validade dos cookies. Caso eles não existam, é mostrada a mensagem “*Você não efetuou o LOGIN!*”. Se os *cookies* existirem, mas forem inválidos, eles serão excluídos pela função *setcookie*, e será mostrada a mesma mensagem, informando que o login deve ser feito.

Agora basta colocar uma chamada para essa include no topo de todas as páginas que devem ser protegidas:

```
include "valida_cookies.inc";
```

**Dica:** Como os cookies são arquivos-texto, qualquer pessoa pode descobrir onde eles estão armazenados e visualizar seu conteúdo. Por isso seria interessante armazenar a senha do usuário de forma criptografada em um cookie. Para criptografar um dado devemos utilizar a função *crypt* do PHP. A *crypt* é uma função que não tem volta, ou seja, não podemos decodificar um dado que foi criptografado. Portanto, no momento de fazer uma comparação com a senha do banco de dados, esta também deve ser criptografada antes de testarmos se as duas são iguais.

O próximo tópico mostra como proceder quando o usuário vai sair do site, ou seja, quando ele vai efetuar um logout.

## Logout

Para o sistema de username/senha ficar completo só está faltando a construção de um programa de logout do usuário. Esse programa é importante porque o usuário pode querer continuar navegando por outros sites, ou seja, o browser dele continua aberto, mas ele continua “logado” em seu site, embora não o esteja mais utilizando. Em muitos lugares um computador é utilizado por mais de uma pessoa, e se não houver um programa de logout, pode ocorrer de uma pessoa pegar uma conexão que a outra deixou aberta, e acessar uma área restrita de seu site.

O programa de logout nada mais é do que a exclusão dos cookies criados no momento do login, que armazenam a senha e o nome de usuário:

### logout.php

```
<?php  
    setcookie("nome_usuario");  
    setcookie("senha_usuario");  
    header ("Location: login.html");  
?>
```

Esse programa exclui os cookies *nome\_usuario* e *senha\_usuario*, e redireciona o browser para a página de login. A partir desse momento o usuário só poderá acessar o site se fornecer sua identificação novamente.

Podemos colocar em todas as páginas do site um link para que o usuário possa fazer o logout a qualquer momento:

```
<a href="logout.php">LOGOUT</a>
```

Por exemplo, sua página inicial poderia ter a seguinte estrutura, incluindo a chamada para a include de validação e um link para realizar o logout.

### pagina\_inicial.php

```
<?php  
    include "valida_cookies.inc";  
?>  
<html>  
    <body>  
        Seja bem-vindo ao meu site!!!<br>  
        Coloque o conteúdo do seu site aqui neste espaço.<br>  
        <p><a href="logout.php">LOGOUT</a></p>  
    </body>  
</html>
```

Dessa forma, se o usuário tentasse acessar pelo browser diretamente o arquivo *pagina\_inicial.php*, sem passar pelo login, o sistema de autenticação não iria permitir a visualização da página.

## Utilizando sessões

Uma sessão é um período de tempo durante o qual uma pessoa navega pelas páginas de um site. Quando um usuário entrar no site, podemos abrir uma sessão e nela registrar diversas variáveis, que ficarão gravadas em arquivos no servidor e poderão ser acessadas em qualquer página do site, enquanto a sessão estiver aberta.

Cada sessão terá um número identificador único, chamado de *session id*. Para uma página ter acesso aos dados da sessão, ela deve conhecer esse identificador. Para transmitir o identificador da sessão entre as páginas, existem duas formas:

- Cookies
- Propagação de variáveis na URL (endereço destino)

A primeira forma consiste em armazenar o identificador da sessão em um *cookie* na máquina do usuário, e utilizá-lo como referência para acessar os dados da sessão no servidor. Esse é o melhor método, e por padrão está habilitado pela diretiva *session.use\_cookies* do arquivo *php.ini*. Porém, como ele nem sempre está disponível (o usuário pode desabilitar os *cookies*), pode ser utilizado o segundo método como alternativa.

O segundo método consiste em propagar o identificador da sessão por meio da URL, o que pode ser feito de duas formas. Se no momento da compilação do PHP foi ativado o parâmetro *--enable-trans-sid*, a identificação da sessão será enviada transparentemente entre as páginas. Caso contrário, devemos acrescentar ao final da URL a constante *SID*, que o PHP cria automaticamente no início de uma sessão. Por exemplo:

```
echo '<a href="pagina.php?'. SID .'">Clique aqui</a>';
```

Sempre que possível utilize o primeiro método (cookies), pois além de oferecer um melhor gerenciamento da sessão, também é muito mais recomendado em termos de segurança.

## Criando uma sessão no PHP

No PHP uma sessão pode ser criada de forma manual ou automática. A criação manual pode ser feita de forma explícita (função *session\_start*) ou de forma implícita (ao registrar uma variável com a função *session\_register*). A forma automática consiste em habilitar a diretiva *session.auto\_start* do arquivo *php.ini*. Assim, sempre que um usuário entrar em seu site será automaticamente criada uma sessão.

A função *session\_start* serve tanto para criar uma sessão como para restaurar os dados uma sessão com base no identificador corrente (passado pelo método *GET*, *POST* ou por cookie). Essa função não possui parâmetros.

```
bool session_start (vazio)
```

Se você estiver usando o método dos cookies para armazenar o identificador da sessão, deve chamar a função *session\_start* antes de qualquer saída produzida pelo browser.

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

## Registrando variáveis em uma sessão

Ao registrar uma variável em uma sessão, estamos tornando-a disponível para todas as páginas que serão acessadas até o término dessa sessão. Existem duas formas de registrar uma variável:

- utilizando a função *session\_register*, que permite registrar uma ou mais variáveis na sessão corrente.  

```
bool session_register (misto nome [, misto...])
```
- adicionando diretamente entradas ao array superglobal *\$\_SESSION*.

A segunda alternativa é mais recomendada, pois a primeira pode apresentar problemas caso a diretiva *register\_globals* do *php.ini* esteja desabilitada. O array *\$\_SESSION* está disponível desde a versão 4.1.0 do PHP. Para versões anteriores pode-se utilizar o *\$HTTP\_SESSION\_VARS*.

Se você utilizar o array *\$\_SESSION*, não será necessário usar funções como *session\_register*, *session\_unregister* e *session\_is\_registered*. Vamos ver alguns exemplos de registro de variáveis. Acompanhe o código a seguir.

```
<?php
session_start();
if (!isset($_SESSION['contador'])) {
    $_SESSION['contador'] = 1;
} else {
    $_SESSION['contador']++;
}
?>
```

Nesse código, uma sessão é inicializada com a função *session\_start()*, e logo após é utilizado o comando *isset* (*Is Set*, ou seja, está definido) do PHP para ver se a variável de sessão chamada *contador* já existe. Se não existir, ela é criada com o valor 1, caso contrário seu valor atual é incrementado.

Vejamos agora outro exemplo. Para testar o registro de variáveis em uma sessão e o uso desses valores em outra página, vamos criar duas páginas, chamadas *pagina1.php* e *pagina2.php*. Na primeira, serão definidas três variáveis de sessão e será exibido um link para a segunda página. Na segunda, serão exibidos os valores dessas três variáveis por meio do comando *echo* do PHP. O código da primeira é o seguinte.

#### pagina1.php

```
<?php
session_start();
echo 'Esta é a primeira página';
$_SESSION['nome'] = 'Juliano';
$_SESSION['sobrenome'] = 'Niederauer';
$_SESSION['data'] = date('d/m/Y', time());
echo '<br><a href="pagina2.php">Página 2</a>';
?>
```

Após iniciar a sessão, as variáveis *nome*, *sobrenome* e *data* foram registradas por meio de uma atribuição direta aos seus respectivos elementos no array *\$\_SESSION*. Logo após foi exibido um link para a segunda página, que irá mostrar esses valores na tela. Nesse exemplo, estamos supondo que está em uso o mecanismo dos cookies para propagação do identificador da sessão. Porém, se você estiver propagando o identificador através da URL, pode exibir o link usando a constante *SID*, da seguinte forma.

```
echo '<br><a href="pagina2.php?' . SID . '">Página 2</a>';
```

Na segunda página, basta restaurar os dados da sessão com o comando `session_start()` e exibir os valores das variáveis, acessando novamente o array `$_SESSION`. Acompanhe o código apresentado a seguir.

### pagina2.php

```
<?php  
session_start();  
echo 'Esta é a segunda página <br>';  
echo $_SESSION['nome'] . "<br>";  
echo $_SESSION['sobrenome'] . "<br>";  
echo $_SESSION['data'] . "<br>";  
echo '<br><a href="pagina1.php">Página 1</a>';  
?>
```

Caso você não vá mais utilizar alguma variável da sessão nos próximos acessos, é possível eliminá-la. Se a diretiva `register_globals` do `php.ini` estiver desabilitada, isso pode ser feito pelo comando `unset` do PHP. Exemplo:

```
unset($_SESSION['nome']);
```

Se a diretiva `register_globals` do `php.ini` estiver habilitada, então você pode eliminar uma variável de sessão por meio da função `session_unregister`. Exemplo:

```
session_unregister('nome');
```

## Parâmetros de configuração

O gerenciamento das sessões pode ser configurado por meio do arquivo `php.ini`, que é um arquivo de configuração do PHP. Veja na tabela a seguir os principais parâmetros disponíveis:

| Parâmetro                              | Descrição   |
|--|---|
| <code>session.save_handler</code>      | Nome do manipulador para armazenar e recuperar dados de uma sessão. O padrão é <code>files</code> .           |
| <code>session.save_path</code>         | Parâmetro que será passado para a rotina de armazenamento (ex: diretório onde serão criados os arquivos).     |
| <code>session.name</code>              | Nome da sessão. O valor-padrão é <code>PHPSESSID</code> .   |
| <code>session.auto_start</code>        | Define se uma sessão deve ser iniciada automaticamente quando houver uma requisição do usuário. O padrão é 0. |
| <code>session.serialize_handler</code> | Define como os dados da sessão serão serializados. O padrão é <code>php</code> , um formato interno do PHP.   |

| Parâmetro                             | Descrição (cont.)  |
|---------------------------------------|--|
| <code>session.gc_probability</code>   | Define a probabilidade (em percentual) de execução da rotina gc ( <i>garbage collection</i> ) a cada requisição.   |
| <code>session.gc_maxlifetime</code>   | Número de segundos que deve ser esperado para que os dados armazenados sejam descartados.  |
| <code>session.referer_check</code>    | Permite definir uma substring para verificar se ela está contida na requisição HTTP.   |
| <code>session.entropy_file</code>     | Caminho para um recurso externo (arquivo) que será usado no processo de criação do identificador da sessão. Exemplo: <code>/dev/random</code> .  |
| <code>session.entropy_length</code>   | Número de bytes que devem ser lidos a partir do recurso especificado no parâmetro <code>session.entropy_file</code> .  |
| <code>session.use_cookies</code>      | Indica se serão utilizados cookies para armazenar o identificador da sessão na máquina do usuário. O padrão é 1 (habilitado).  |
| <code>session.use_only_cookies</code> | Indica se devem ser usados somente cookies para armazenar o identificador da sessão. É usado para prevenir possíveis ataques de passagem do identificador através da URL. O padrão é 0 (desabilitado).                                 |
| <code>session.cookie_lifetime</code>  | Define em quanto tempo (em segundos) os cookies enviados expiram. O padrão é 0, ou seja, até que o browser seja encerrado.   |
| <code>session.cookie_path</code>      | Define o caminho no qual o cookie é válido. O padrão é .   |
| <code>session.cookie_domain</code>    | Define o domínio no qual o cookie é válido.  |
| <code>session.cookie_secure</code>    | Se habilitado, os cookies serão enviados apenas se a conexão for segura (HTTPS).   |
| <code>session.cache_limiter</code>    | Define o método de controle de cache usado para as páginas da sessão. O padrão é <code>nocache</code> . Outros valores possíveis são <code>none</code> , <code>private</code> , <code>private_no_expire</code> e <code>public</code> . |
| <code>session.cache_expire</code>     | Especifica o tempo (em minutos) de validade para as páginas de sessão armazenadas na cache. Só tem efeito se o método utilizado não for o <code>nocache</code> . O padrão é 180.   |
| <code>session.use_trans_sid</code>    | Indica se a propagação transparente do identificador da sessão está habilitada. O padrão é 0 (desabilitada).   |
| <code>session.bug_compat_42</code>    | Permite desabilitar uma característica da versão 4.2 do PHP, que possibilita inicializar variáveis de sessão no escopo global.   |
| <code>session.bug_compat_warn</code>  | É complementar ao parâmetro anterior. Indica se deve ser exibido um aviso caso o usuário tente usar essa característica.   |
| <code>url_rewriter.tags</code>        | Define as tags HTML que devem ser reescritas para inclusão do identificador de sessão, quando o modo transparente estiver habilitado.  |

## Usando sessões no sistema de username/senha

Se você desejar, pode alterar o sistema de username/senha apresentado no tópico sobre cookies, para que ele passe a utilizar sessões. Para isso, são necessárias apenas algumas modificações nos arquivos de login, autenticação e logout.

Por exemplo, ao invés de criar dois cookies no momento do login (arquivo *login.php*), deveriam ser registradas duas variáveis de sessão, ou seja, ao invés de:

```
setcookie("nome_usuario", $username);
setcookie("senha_usuario", $senha);
```

teríamos o seguinte:

```
session_start();
$_SESSION['nome_usuario'] = $username;
$_SESSION['senha_usuario'] = $senha;
```

Na etapa de autenticação (arquivo *valida\_cookies.inc*), deveríamos verificar o valor das variáveis de sessão ao invés dos cookies, ou seja, bastaria substituir o array *\$\_COOKIE* pelo *\$\_SESSION*. Porém, lembre-se que sempre é necessário restaurar os dados da sessão atual utilizando a função *session\_start()*. Portanto, alterando o nome do array nas primeiras linhas do nosso script de autenticação, teríamos o seguinte código:

```
session_start();
if(isset($_SESSION["nome_usuario"]))
    $nome_usuario = $_SESSION["nome_usuario"];
if(isset($_SESSION["senha_usuario"]))
    $senha_usuario = $_SESSION["senha_usuario"];
```

Outra parte da rotina de autenticação que deveria ser alterada é a exclusão dos cookies. No caso das sessões, isso equivale a eliminar as variáveis de sessão *nome\_usuario* e *senha\_usuario*. Portanto, as linhas:

```
setcookie("nome_usuario");
setcookie("senha_usuario");
```

deveriam ser substituídas por:

```
unset($_SESSION['nome_usuario']);
unset($_SESSION['senha_usuario']);
```

Por fim, deveríamos alterar a rotina de saída do usuário do site (*logout.php*), fazendo-a utilizar a função *session\_destroy()* do PHP, que elimina todos os dados de uma sessão. É importante destacar que essa função apenas encerra a sessão, mas não libera o espaço alocado para as variáveis registradas. Para eliminar essas variáveis, se você estiver usando o array *\$\_SESSION*, basta reiniá-lo atribuindo-lhe um array vazio:

```
$_SESSION = array();
```

Caso contrário, utilize a função *session\_unset()*, que tem a mesma finalidade. Veja então no código a seguir como poderia ser o programa de logout de usuários com o uso de sessões.

```
<?php  
session_start();  
$_SESSION = array();  
session_destroy();  
header ("Location: login.html");  
?>
```

Esse programa encerra a sessão atual, elimina todas suas variáveis e redireciona o usuário de volta para a página principal de seu site. A partir daí, ele só poderá acessar o conteúdo do site se fizer o login novamente. Note que, antes de destruir a sessão, foi necessário chamar a função *session\_start* para restaurar o ambiente da sessão atual.



# Capítulo 14

## Manipulando arquivos em PHP

Além de utilizar um SGBD (Sistema Gerenciador de Banco de Dados), como o MySQL e o PostgreSQL, existe outra forma de armazenar dados para recuperá-los posteriormente. O PHP oferece uma série de funções para trabalharmos com o sistema de arquivos do sistema operacional. Podemos abrir, fechar, ler, escrever e realizar diversas outras operações sobre um arquivo de formato conhecido, não necessitando que haja um SGBD instalado no servidor. Veremos neste capítulo em quais situações é interessante realizar a manipulação de arquivos, e conheceremos os principais comandos para essa tarefa.

### Quando utilizar arquivos no PHP

Em aplicações que exigem o armazenamento de poucos dados, às vezes não há a necessidade de utilizar um SGBD para armazená-los, pois isso torna o processo mais lento, visto que deve haver uma conexão com o banco de dados para realizar uma consulta. Se o servidor de banco de dados estiver localizado em um outro host (outro servidor da rede), a situação se agrava, pois a rede pode estar congestionada. Nesses casos podemos trabalhar, por exemplo, diretamente com arquivos no formato texto, o que torna mais rápido o processo de armazenamento e recuperação de dados.

Entre as situações em que podemos utilizar a manipulação de arquivos em vez de um SGBD, podem ser citadas: criação de contadores de acessos para uma ou mais páginas do site, criação de um programa gerenciador de publicidade online (anúncios por meio de banners) e criação de fóruns de discussão ou livro de visitas (guestbook).

Em todos os exemplos citados os dados podem ser armazenados em simples arquivos no formato texto. É claro que não vamos utilizar esse tipo de arquivos para armazenar os dados dos usuários do site, pois além disso não oferecem a segurança necessária e a manipulação desses dados (inclusão, alteração e ex-

clusão) fica extremamente difícil. É recomendado o uso de arquivos somente quando o volume de dados for pequeno, evitando assim que seja perdido tempo com a utilização de um SGBD para realizar uma tarefa simples.

## Funções para manipulação de arquivos

Essas funções são responsáveis pelas diversas operações que podemos realizar sobre os arquivos. O PHP nos oferece dezenas de funções capazes de trabalhar com o filesystem (sistema de arquivos) do sistema operacional. As principais operações que podemos realizar sobre um arquivo são: abertura, leitura, escrita e fechamento. Portanto, veremos com maiores detalhes as funções que realizam essas tarefas.

### fopen

Para abrir um arquivo, utilizamos a função *fopen*, que possui a seguinte sintaxe:

```
recurso fopen (string nome_arquivo, string modo [, int  
    usar_include_path [, recurso contexto]])
```

| Parâmetro                | Descrição  |
|--------------------------|--|
| <i>nome_arquivo</i>      | Nome do arquivo a ser aberto, que pode ser local ou remoto. Se for fornecido um URL, a abertura só irá funcionar se a diretiva <i>allow_url_fopen</i> estiver habilitada no <i>php.ini</i> . |
| <i>modo</i>              | Modo de acesso ao arquivo. A lista dos modos disponíveis será apresentada logo a seguir.   |
| <i>usar_include_path</i> | Indica se o arquivo deve ser procurado nos diretórios especificados na diretiva <i>include_path</i> do <i>php.ini</i> .  |
| <i>contexto</i>          | Permite a definição de um contexto, que consiste em um conjunto de parâmetros que modificam o comportamento do arquivo. O suporte a contextos foi adicionado no PHP 5.                       |

O parâmetro *nome\_arquivo* pode referenciar um arquivo que está no mesmo computador ou em um computador remoto. Se esse parâmetro iniciar com “*http://*”, será aberta uma conexão HTTP, que retornará um ponteiro para o arquivo que foi aberto. Se o nome do arquivo iniciar com “*ftp://*”, será aberta uma conexão FTP antes da abertura do arquivo.

Se a abertura não necessitar do estabelecimento de uma conexão HTTP ou FTP, o arquivo será aberto do filesystem da própria máquina, e será retornado um ponteiro para esse arquivo. A função *fopen* retorna o valor falso se a abertura do arquivo falhar.

O segundo parâmetro da função *fopen* é o *modo*, que pode possuir os seguintes valores:

| Modo | Descrição   |
|------|---|
| 'r'  | Abre somente para leitura, posicionando o ponteiro no inicio do arquivo.  |
| 'r+' | Abre para leitura e escrita, posicionando o ponteiro no inicio do arquivo.  |
| 'w'  | Abre somente para escrita, posicionando o ponteiro no inicio do arquivo e deixando-o com tamanho zero. Se o arquivo não existir, tenta criá-lo.   |
| 'w+' | Abre para leitura e escrita, posicionando o ponteiro no inicio do arquivo e deixando-o com tamanho zero. Se o arquivo não existir, tenta criá-lo.   |
| 'a'  | Abre somente para escrita, posicionando o ponteiro no final do arquivo. Se o arquivo não existir, tenta criá-lo.  |
| 'a+' | Abre para leitura e escrita, posicionando o ponteiro no final do arquivo. Se o arquivo não existir, tenta criá-lo.  |
| 'x'  | Cria e abre um arquivo somente para escrita, posicionando o ponteiro no inicio do arquivo. Se o arquivo já existir, retorna falso ( <i>FALSE</i> ) e gera um erro do tipo <i>E_WARNING</i> . Disponível desde a versão 4.3.2 do PHP, esse modo é usado apenas em arquivos locais.   |
| 'x+' | Cria e abre um arquivo para leitura e escrita, posicionando o ponteiro no inicio do arquivo. Se o arquivo já existir, retorna falso ( <i>FALSE</i> ) e gera um erro do tipo <i>E_WARNING</i> . Disponível desde a versão 4.3.2 do PHP, esse modo é usado apenas em arquivos locais. |

Existe ainda a possibilidade de usar as letras 'b' ou 't' como último caractere do parâmetro *modo*. A letra 'b' força o uso do modo binário, devendo ser utilizada em sistemas que diferenciam arquivos nos formatos binário e texto. A letra 't' é usada em sistemas Windows para traduzir os caracteres de quebra de linha (\n) para \r\n. Veja alguns exemplos de uso da função *fopen*:

```
<?php
$ponteiro = fopen ("/home/juliano/teste.txt", "r");
$ponteiro = fopen ("/home/juliano/teste2.txt", "wb");
$ponteiro = fopen ("/home/juliano/teste2.txt", "a+");
$ponteiro = fopen ("http://www.teste.com.br", "r");
?>
```

Se você estiver com problemas para a criação ou abertura de um arquivo, verifique se você tem as devidas permissões para acessar o diretório no qual a gravação deveria ser feita. Outro detalhe importante diz respeito aos sistemas Windows, onde devem ser acrescentados caracteres de escape ao utilizarmos as barras invertidas. Por exemplo:

```
<?php
$ponteiro = fopen ("c:\\teste\\\\arquivo.txt", "r");
?>
```

### **fclose**

Para fechar um arquivo, utilizamos a função *fclose*, que possui a seguinte sintaxe:

```
bool fclose (recurso ponteiro_arquivo)
```

Essa função retorna *true* se o arquivo foi fechado com sucesso e retorna *false* se houver alguma falha. O parâmetro passado para a função *fclose* deve conter a variável para a qual foi atribuído o resultado da função *fopen*, ou seja, o ponteiro (handle) para o arquivo que foi aberto. Exemplo:

```
<?php  
$ponteiro = fopen('arquivo.txt', 'r');  
***  
fclose($ponteiro);  
?>
```

### **fread**

Uma das formas de ler dados de um arquivo é utilizar a função *fread*, que permite especificar a quantidade de informações a serem lidas. Sua sintaxe é a seguinte:

```
string fread (recurso ponteiro_arquivo, int tamanho)
```

Essa função lê o número de bytes especificado no parâmetro *tamanho* a partir da posição atual do ponteiro definido pelo primeiro parâmetro. A leitura termina quando o número de bytes especificado é lido ou o fim do arquivo (*EOF* - *End Of File*) é alcançado. Por exemplo:

#### **leitura.php**

```
<?php  
$ponteiro = fopen ("teste.txt", "r");  
$conteudo = fread ($ponteiro, 30);  
echo $conteudo;  
fclose ($ponteiro);  
?>
```

Esse programa abre um arquivo chamado *teste.txt* e lê os seus 30 primeiros bytes, armazenando-os na variável *\$conteudo*. Em seguida, o valor obtido é exibido na tela com o comando *echo* e o arquivo é fechado com *fclose*.

**fgets**

Também realiza a leitura de um arquivo, mas lê no máximo uma linha. É bastante utilizado quando queremos trabalhar individualmente com cada linha do arquivo. Sua sintaxe é a seguinte:

```
string fgets (recurso ponteiro_arquivo [, int tamanho])
```

A leitura é feita até que seja lido o número de bytes especificados em *tamanho*, ou quando terminar a linha atual do arquivo (caráter \n), ou quando o arquivo chegar ao seu final. Se não for especificado o parâmetro *tamanho*, será utilizado o valor padrão, que é 1024 bytes (1k). Como exemplo, considere o programa *linha.php* apresentado a seguir.

**琳ha.php**

```
<?php  
$ponteiro = fopen ("teste.txt", "r");  
$linha = fgets($ponteiro, 4096);  
echo $linha;  
fclose ($ponteiro);  
?>
```

Esse programa lê a primeira linha de um arquivo, exibindo-a na tela. Note que foi utilizado um valor bem alto (4096) para garantir a leitura da linha inteira. Se a função *fgets* fosse chamada novamente, seria retornada a segunda linha do arquivo, e assim por diante.

**fwrite**

Para escrever dados em um arquivo com o PHP, utilizamos a função *fwrite*, que possui a seguinte sintaxe:

```
int fwrite (recurso ponteiro_arquivo, string string [, int tamanho])
```

Esta função escreve o conteúdo do parâmetro *string* no arquivo referenciado pelo parâmetro *ponteiro\_arquivo*. O parâmetro *tamanho* é opcional. Se for informado, a escrita irá parar após serem atingidos o número de bytes especificado. Veja um exemplo de utilização dessa função:

**escrita.php**

```
<?php  
$conteudo = "Este texto será escrito no arquivo!";  
$ponteiro = fopen ("arquivo.txt", "w");  
fwrite($ponteiro, $conteudo);  
fclose ($ponteiro);  
?>
```

Esse programa tentará abrir um arquivo chamado *arquivo.txt* no modo de escrita, eliminando o seu conteúdo e posicionando o ponteiro no início do arquivo. Caso o arquivo não exista, ele tentará criá-lo. Lembre-se que para isso é necessário que o diretório possua permissão de escrita.

Em seguida, é utilizada a função *fwrite* para escrever a string armazenada na variável *\$conteudo* no arquivo. Por fim, o arquivo é fechado com *fclose*.

## Exemplo: contador de acessos

Utilizando arquivos você pode criar um contador de acessos para as páginas do seu site. Você pode criar um único contador para o site inteiro, ou então criar scripts individuais para contar os acessos de cada página. Como exemplo, vamos criar um programa que armazena o valor atual em um arquivo chamado *contador.txt* e o incrementa a cada acesso. Esse programa será uma *include* nomeada como *contador.inc*, que posteriormente poderá ser incluída nas páginas que desejarmos realizar a contagem. Acompanhe o código apresentado a seguir.

### contador.inc

```
<?php
$arquivo = "contador.txt"; // arquivo do contador
if(file_exists($arquivo)) // se existe, lê o valor atual e o incrementa
{
    $fd = fopen($arquivo,"r");
    $valor_atual = chop(fgets($fd));
    fclose($fd);
    $valor_atual++;
}
else
    $valor_atual = 1;
// grava o novo valor no arquivo
$ponteiro = fopen ($arquivo, "w");
fwrite($ponteiro, $valor_atual);
fclose ($ponteiro);
?>
```

Primeiramente, é utilizada a função *file\_exists* para testar se o arquivo *contador.txt* já existe. Caso ele exista, obtemos seu valor atual utilizando a função *fgets* e logo após o incrementamos. Note que nesse exemplo utilizamos a função *chop* do PHP no momento da leitura do conteúdo do arquivo. Essa

função tem por finalidade remover os espaços e quebras de linha existentes no fim de uma string. Se não utilizarmos essa função, poderão ocorrer problemas no momento de realizar operações aritméticas sobre o dado lido.

Caso o arquivo do contador ainda não tenha sido criado, atribuímos o valor 1 à variável que armazena o valor atual. Em seguida, o arquivo do contador é aberto (ou criado) no modo de escrita e o valor atual é gravado por meio da função *fwrite*.

Lembre-se que o diretório onde o programa está sendo executado deve possuir permissão de escrita (isso pode ser configurado no Linux por meio do comando *chmod*). Então, bastaria colocar uma chamada para o contador nas páginas que desejarmos:

```
include "contador.inc";
```

Dessa forma o arquivo *contador.txt* sempre irá armazenar o número total de acessos que determinadas páginas tiveram. Uma alternativa seria você usar arquivos texto separados para contar de forma individual o número de visitas a diferentes páginas do site.

## Outras funções para o sistema de arquivos

Além das funções principais que acabamos de ver, existem dezenas de funções adicionais para o tratamento de arquivos. A seguir são apresentadas as funções que podemos utilizar, juntamente com a descrição e sintaxe de cada uma delas:

### basename

Retorna o nome do arquivo que faz parte do caminho (*path*) fornecido. Permite especificar um sufixo a ser eliminado caso ele exista no final do nome do arquivo.

```
string basename (string caminho [, string sufixo])
```

### chgrp

Tenta alterar o grupo a que pertence o arquivo.

```
bool chgrp (string nome_arquivo, misto grupo)
```

### chmod

Tenta alterar as permissões do arquivo, de acordo com o modo fornecido.

```
bool chmod (string nome_arquivo, int modo)
```

**chown**

Tenta alterar o proprietário do arquivo.

```
bool chown (string nome_arquivo, misto usuário)
```

**clearstatcache**

Limpa a memória cache que armazena informações de estado do arquivo. O PHP utiliza essa memória para melhorar o desempenho na execução de algumas funções, como *stat*, *lstat*, *file\_exists*, *is\_writable* etc.

```
void clearstatcache (vazio)
```

**copy**

Copia um arquivo.

```
bool copy (string origem, string destino)
```

**delete**

Exclui um arquivo, assim como a função *unlink*.

```
void delete (string arquivo)
```

**dirname**

Retorna o diretório que faz parte do caminho (*path*) fornecido.

```
string dirname (string caminho)
```

**disk\_free\_space**

Retorna o espaço (em bytes) disponível no diretório.

```
float disk_free_space (string diretório)
```

**disk\_total\_space**

Retorna o tamanho (em bytes) total de um diretório.

```
float disk_total_space (string diretório)
```

**diskfreespace**

É um apelido (alias) para a função *disk\_free\_space*.

**fclose**

Fecha um arquivo.

```
bool fclose (recurso ponteiro_arquivo)
```

**feof**

Testa se o ponteiro já alcançou o final do arquivo.

*bool feof (recurso ponteiro\_arquivo)*

**fflush**

Força a escrita de todos os dados de saída armazenados em buffer para um arquivo.

*bool fflush (recurso ponteiro\_arquivo)*

**fgetc**

Obtém um caractere a partir da posição atual do ponteiro do arquivo.

*string fgetc (recurso ponteiro\_arquivo)*

**fgetcsv**

Lê uma linha do arquivo e analisa os campos como se estivessem no formato CSV.

*array fgetcsv (recurso ponteiro\_arquivo [, int tamanho [, string delimitador [, string delimitador2]]])*

**fgets**

Lê uma linha do arquivo.

*string fgets (recurso ponteiro\_arquivo [, int tamanho])*

**fgetss**

Lê uma linha do arquivo e retira as tags HTML e PHP.

*string fgetss (recurso ponteiro\_arquivo, int tamanho [, string tags\_permitidas])*

**file\_exists**

Testa se um arquivo ou diretório existe.

*bool file\_exists (string nome\_arquivo)*

**file\_get\_contents**

Lê o conteúdo inteiro de um arquivo, retornando-o em uma string.

*string file\_get\_contents (string nome\_arquivo [, int usar\_include\_path [, recurso contexto]])*

**file\_put\_contents**

Escreve uma string em um arquivo. É equivalente a chamar as funções *fopen*, *fwrite* e *fclose* sucessivamente.

```
int file_put_contents (string nome_arquivo, string dados [, int flags  
[, recurso contexto]])
```

**file**

Lê o conteúdo inteiro de um arquivo, retornando-o em array onde cada elemento corresponde a uma linha do arquivo.

```
array file (string nome_arquivo [, int usar_include_path [, recurso  
contexto]])
```

**fileatime**

Retorna a hora do último acesso ao arquivo, utilizando formato timestamp do UNIX.

```
int fileatime (string nome_arquivo)
```

**filectime**

Retorna a hora da última modificação no descritor do arquivo, utilizando formato timestamp do UNIX.

```
int filectime (string nome_arquivo)
```

**filegroup**

Retorna o identificador do grupo ao qual pertence o arquivo.

```
int filegroup (string nome_arquivo)
```

**fileinode**

Retorna o descritor (*inode*) do arquivo.

```
int fileinode (string nome_arquivo)
```

**filemtime**

Retorna a hora de alteração do arquivo, utilizando formato timestamp do UNIX.

```
int filemtime (string nome_arquivo)
```

**fileowner**

Retorna o número identificador do proprietário do arquivo.

```
int fileowner (string nome_arquivo)
```

**fileperms**

Retorna as permissões de acesso do arquivo.

```
int fileperms (string nome_arquivo)
```

**filesize**

Retorna o tamanho do arquivo, em bytes.

```
int filesize (string nome_arquivo)
```

**filetype**

Retorna o tipo do arquivo. Os valores possíveis são *fifo*, *char*, *dir*, *block*, *link*, *file* e *unknown*.

```
string filetype (string nome_arquivo)
```

**flock**

Bloqueia ou desbloqueia um arquivo para realizar determinadas operações. Pode ser usado, por exemplo, para evitar escritas concorrentes.

```
bool flock (recurso ponteiro_arquivo, int operação [, int &bloquear])
```

**fnmatch**

Busca por um padrão em um nome de arquivo.

```
array fnmatch (string padrão, string string [, int flags])
```

**fopen**

Abre um arquivo ou URL. Os modos de abertura mais usados são “r” (leitura), “r+” (leitura e escrita), “w” (escrita, zerando o arquivo), “w+” (leitura e escrita, zerando o arquivo), “a” (escrita, posicionando no final do arquivo) e “a+” (leitura e escrita, posicionando no final do arquivo).

```
recurso fopen (string nome_arquivo, string modo [, int  
    usar_include_path [, 'recurso contexto']])
```

**fpassthru**

Exibe os dados restantes a partir da posição atual do ponteiro do arquivo.

```
int fpassthru (recurso ponteiro_arquivo)
```

**fputs**

É um apelido (alias) para a função *fwrite*.

**fread**

Lê dados de um arquivo. A leitura termina quando o número de bytes especificado é lido ou o fim do arquivo é alcançado.

```
string fread (recurso ponteiro_arquivo, int tamanho)
```

**fscanf**

Interpreta o conteúdo de um arquivo de acordo com um determinado formato.

```
misto fscanf (recurso ponteiro_arquivo, string formato [, string vari])
```

**fseek**

Posiciona o ponteiro em um arquivo. A nova posição é obtida adicionando-se o parâmetro *offset* à posição especificada pelo parâmetro *local*, que pode assumir os valores *SEEK\_SET*(padrão), *SEEK\_CUR* e *SEEK\_END*.

```
int fseek (recurso ponteiro_arquivo, int offset [, int local])
```

**fstat**

Retorna informações sobre um arquivo usando um ponteiro aberto.

```
array fstat (recurso ponteiro_arquivo)
```

**f.tell**

Retorna a posição atual de leitura/escrita do ponteiro do arquivo.

```
int ftell (recurso ponteiro_arquivo)
```

**ftruncate**

Trunca um arquivo, ou seja, elimina seu conteúdo deixando-o com tamanho zero.

```
bool ftruncate (recurso ponteiro_arquivo, int tamanho)
```

**fwrite**

Escreve dados em um arquivo. Se for informado o parâmetro *tamanho*, a escrita irá parar após serem atingidos o número de bytes especificado.

```
int fwrite (recurso ponteiro_arquivo, string string [, int tamanho])
```

**glob**

Retorna um array contendo todos os caminhos (arquivos/diretórios) que satisfazem um determinado padrão.

```
array glob (string padrão [, int flags])
```

**is\_dir**

Testa se o nome fornecido é um diretório.

```
bool is_dir (string nome_arquivo)
```

**is\_executable**

Testa se um arquivo é executável.

```
bool is_executable (string nome_arquivo)
```

**is\_file**

Testa se o nome fornecido é um arquivo válido.

```
bool is_file (string nome_arquivo)
```

**is\_link**

Testa se o arquivo é um link simbólico.

```
bool is_link (string nome_arquivo)
```

**is\_readable**

Testa se o arquivo pode ser lido.

```
bool is_readable (string nome_arquivo)
```

**is\_uploaded\_file**

Indica se o arquivo foi enviado através de um upload via HTTP pelo método POST.

```
bool is_uploaded_file (string nome_arquivo)
```

**is\_writable**

Testa se o arquivo pode ser escrito.

```
bool is_writable (string nome_arquivo)
```

**is\_writeable**

Apelido (*alias*) para a função *is\_writable*.

**link**

Cria um link físico.

```
bool link (string alvo, string link)
```

**linkinfo**

Retorna informações sobre um link.

```
int linkinfo (string caminho)
```

**lstat**

Retorna informações sobre um arquivo ou um link simbólico.

```
array lstat (string nome_arquivo)
```

**mkdir**

Cria um diretório.

```
bool mkdir (string caminho [, int modo])
```

**move\_uploaded\_file**

Move um arquivo enviado por upload para um novo local.

```
bool move_uploaded_file (string nome_arquivo, string destino)
```

**parse\_ini\_file**

Analisa um arquivo de configuração *.ini*, retornando suas informações em um array associativo.

```
array parse_ini_file (string nome_arquivo [, bool seções_processos])
```

**pathinfo**

Retorna um array associativo contendo informações sobre o caminho para um arquivo. Os elementos retornados são *dirname*, *basename* e *extension*.

```
array pathinfo (string caminho)
```

**pclose**

Fecha o ponteiro para um processo, aberto com *popen*.

```
int pclose (recurso ponteiro_arquivo)
```

**popen**

Abre um ponteiro para um processo. Difere do *fopen* por ser unidirecional (usado somente para leitura ou para escrita). O ponteiro pode ser utilizado em funções como *fgets*, *fgetss* e *fputs*.

```
recurso popen (string comando, string modo)
```

**readfile**

Lê um arquivo e imprime seu conteúdo na saída-padrão.

```
int readfile (string nome_arquivo [, bool usar_include_path [, recurso contexto]])
```

**readlink**

Retorna o destino de um link simbólico.

```
string readlink (string caminho)
```

**realpath**

Retorna um caminho em sua forma canônica, expandindo links simbólicos e resolvendo referências como './' e '../'.  
**string realpath (string caminho)**

**rename**

Renomeia um arquivo.

**bool rename (string nome\_antigo, string novo\_nome)****rewind**

Retorna o ponteiro ao início do arquivo.

**bool rewind (recurso ponteiro\_arquivo)****rmdir**

Exclui um diretório.

**bool rmdir (string diretório)****set\_file\_buffer**

Apelido (*alias*) para a função *stream\_set\_write\_buffer*. Permite ajustar a quantidade de dados a serem armazenados em buffer nas operações de escrita.

**int stream\_set\_write\_buffer (recurso arquivo, int buffer)****stat**

Retorna um array contendo informações sobre um arquivo, como número do descritor (*inode*), identificador do proprietário, tamanho, último acesso, entre outras.

**array stat (string nome\_arquivo)****symlink**

Cria um link simbólico.

**bool symlink (string alvo, string link)****tempnam**

Cria um nome de arquivo único dentro do diretório especificado.

**string tempnam (string dir, string prefixo)**

**tmpfile**

Cria um arquivo temporário com um nome único, no modo de escrita. O arquivo é automaticamente excluído ao final da execução ou se ele for fechado de forma explícita com *fclose*.

*recurso tmpfile (vazio)*

**touch**

Tenta configurar a hora de acesso e de modificação de um determinado arquivo.

*bool touch (string nome\_arquivo [, int tempo [, int tempo\_acesso]])*

**umask**

Troca a máscara (*umask*) corrente e retorna a máscara antiga.

*int umask ([int mask])*

**unlink**

Exclui um arquivo. Retorna *TRUE* em caso de sucesso e *FALSE* se falhar.

*bool unlink (string nome\_arquivo)*

# Capítulo 15

## Enviando e-mails com o PHP

O envio de e-mails com o PHP é um recurso utilizado com freqüência pelos desenvolvedores de páginas Web. Neste capítulo veremos qual é a utilidade de enviar e-mails por meio de seus programas e como utilizar a função que executa essa tarefa.

### Por que enviar e-mails com o PHP?

Existem diversas situações em que o envio de e-mails pelo PHP é de grande utilidade. Quando, por exemplo, um usuário preenche o formulário de cadastramento de um site, escolhendo um username (nome de usuário) e uma senha, é comum que o sistema lhe envie automaticamente um e-mail confirmando seu cadastro, enviando também os dados que ele forneceu por meio do formulário. Isso é feito para que o usuário possa guardar essas informações, podendo recuperá-las em caso de esquecimento da senha ou do username.

Outro exemplo que pode ser citado é o de uma loja que vende os seus produtos pela Internet, e cada pedido de compra é armazenado no banco de dados. Como a loja vai saber se algum cliente realizou alguma compra? Uma alternativa seria fazer um programa que consulte o banco de dados e mostre os pedidos ocorridos em determinado período de tempo. Mas nesse caso a melhor alternativa seria fazer com que a loja recebesse um e-mail informando que houve uma compra. Esse e-mail poderia conter, por exemplo, nome e endereço do cliente e os produtos que ele comprou.

Seria interessante que o cliente também recebesse um e-mail informando que seu pedido foi recebido, com outras instruções que ele deve seguir para receber os produtos comprados. Nesse caso o programa PHP que recebe os dados do cliente e da compra deveria realizar três tarefas:

- 1) Gravar os dados do cliente e da compra no banco de dados.

- 2) Enviar um e-mail para o cliente confirmando o recebimento de seu pedido.
- 3) Enviar um e-mail para a própria loja, notificando-a de que houve uma compra, para que o pedido seja processado.

Alguns desenvolvedores, para garantir que o usuário digite o e-mail correto no momento do cadastro, utilizam o PHP para enviar um e-mail para o endereço digitado, enviando nesse e-mail uma senha especial. Então, o usuário deve acessar uma outra página e digitar essa senha para validar seu cadastro.

Outra situação que pode ser citada é o envio de sugestões ou comentários pelos usuários de um site. Muitos sites apresentam um formulário pelo qual o usuário pode enviar suas opiniões. No momento em que o usuário termina de digitar e clica no botão “Enviar”, o programa PHP que recebe os dados envia dois e-mails: um para o próprio site, contendo os dados do formulário, e outro para o usuário, agradecendo e confirmando o recebimento de sua opinião.

Enfim, essas são apenas algumas das inúmeras aplicações desse recurso que o PHP nos oferece.

## Utilizando a função *mail*

A função *mail* é a única responsável pelo envio de e-mails por meio de programas PHP. Sua sintaxe é a seguinte:

```
bool mail (string destinatário, string assunto, string mensagem [,  
          string cabeçalhos_adicionais [, string parâmetros_adicionais]])
```

Esta função envia automaticamente um e-mail para o endereço especificado em *destinatário*, contendo o texto especificado em *mensagem*. O título do e-mail deve ser especificado no parâmetro *assunto*. As informações mostradas entre colchetes são opcionais.

Veja um simples exemplo de uso da função *mail*:

```
mail("joao@dominio.com.br", "Bem-vindo", "Olá João\nSeja bem-vindo!");
```

Podemos também enviar um e-mail para múltiplos destinatários. Para isso, basta separar os endereços pelo caractere ; (ponto-e-vírgula), conforme mostra o exemplo a seguir:

```
mail("joao@dominio.com.br;pedro@outrodominio.com.br", "Teste", "Olá João e  
Pedro!\nIsto é um teste");
```

O caractere especial \n representa uma quebra de linha. É o equivalente a apertar a tecla ENTER nos editores de texto. Portanto, quando esse comando for executado, será enviado um e-mail com o título “Teste” para os dois endereços.

reços especificados no primeiro parâmetro da função, e esse e-mail conterá a seguinte mensagem:

Olá João e Pedro!

Isto é um teste

Como exemplo, veremos um programa que recebe sugestões enviadas por visitantes de um site e as envia por e-mail ao webmaster. Esse programa será ativado por um formulário HTML, onde o usuário vai digitar seu nome, e-mail e sua sugestão. O código para esse formulário será o seguinte:

### **exemplo15\_1.html**

```
<HTML>
<BODY>
<form action="exemplo15_1.php" method="POST">
    Nome: <input type="text" name="nome"><br>
    E-mail: <input type="text" name="email"><br>
    Sugestão: <input type="text" name="sugestao"><br>
    <input type="submit" name="enviar" value="Enviar!">
</form>
</BODY>
</HTML>
```

Ao acessar essa página em seu navegador, você verá a seguinte tela.

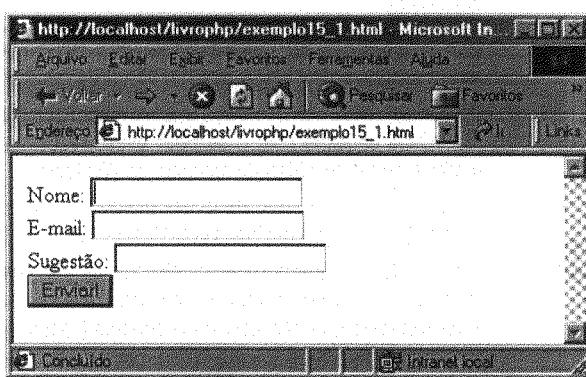


Figura 15.1 – Formulário para o envio de sugestões.

Quando o usuário clicar no botão “Enviar!”, os dados serão submetidos ao programa *exemplo15\_1.php*, apresentado a seguir. Esse programa obtém os dados que o usuário digitou e coloca-os em forma de uma mensagem, para depois enviá-los utilizando a função *mail*.

### exemplo15\_1.php

```
<?php
    $nome = $_POST["nome"];
    $email = $_POST["email"];
    $sugestao = $_POST["sugestao"];
    $mensagem = "Sugestão enviada por um visitante:\n\n";
    $mensagem .= "Nome: $nome\n";
    $mensagem .= "E-mail: $email\n";
    $mensagem .= "Sugestão: $sugestao";
    mail("webmaster@umsite.com.br", "Sugestao", $mensagem);
    echo "Obrigado por enviar sua sugestão!";
?>
```

Note que foi utilizado o operador de concatenação (`.`) para formar a mensagem que foi enviada pela função `mail`.

## Configurações no arquivo `php.ini`

O arquivo de configuração `php.ini` possui algumas diretivas de configuração que estão relacionadas com o envio de e-mails pelo PHP. A seguir são apresentadas essas diretivas, juntamente com uma descrição de cada uma delas.

| Diretiva                   | Descrição  |
|----------------------------|--|
| <code>SMTP</code>          | Contém o nome ou o IP do servidor SMTP que o PHP deve utilizar para enviar e-mails com a função <code>mail</code> no Windows ( <code>default=localhost</code> ).   |
| <code>smtp_port</code>     | Disponível desde o PHP 4.3.0, contém o número da porta na qual deve ser feita a conexão com o servidor especificado na diretiva <code>SMTP</code> ( <code>default=porta 25</code> ). Utilizada somente no Windows.   |
| <code>sendmail_from</code> | Endereço de e-mail que deve ser usado no campo do remetente ("From:") nos e-mails enviados pelo PHP no Windows ( <code>default=NULL</code> ).  |
| <code>sendmail_path</code> | Indica onde o programa do sendmail pode ser encontrado no sistema. Normalmente contém o valor <code>/usr/sbin/sendmail</code> ou <code>/usr/lib/sendmail</code> . Para sistemas que não utilizam o <code>sendmail</code> , essa diretiva deve ser configurada de acordo com o software utilizado. Por exemplo, usuários do Qmail podem alterar essa diretiva para <code>/var/qmail/bin/sendmail</code> ou <code>/var/qmail/bin/qmail-inject</code> . |

## Adicionando informações ao cabeçalho do e-mail

A função `mail` ainda possui um quarto e um quinto parâmetro, que são opcionais. Se o quarto parâmetro for passado, ele será acrescentado ao final do ca-

beçalho do e-mail. Esse parâmetro é utilizado para enviar informações extras ao servidor que receberá a mensagem. Nele podemos incluir uma ou mais informações extras, separando-as por uma quebra de linha (caractere \n). Veja um exemplo simples:

### exemplo15\_2.php

```
<?php  
    $de = "webmaster@algumdominio.com.br";  
    $para = "alguem@dominio.com.br";  
    $mensagem .= "Isto é um teste!";  
    mail($para, "Teste", $mensagem, "From: $de");  
?>
```

Esse comando envia, no quarto parâmetro, a informação *From*, que indica, ao servidor que receberá a mensagem, qual o endereço eletrônico do remetente. Agora observe o comando a seguir, que envia mais de uma informação no quarto parâmetro da função *mail*:

```
mail("alguem@dominio.com", "Teste", $mensagem, "From:  
webmaster@dominio.com\nReply-To: info@dominio.com");
```

A informação *Reply-To* indica para qual endereço eletrônico deve ser enviada a resposta para este e-mail. Portanto, quando o destinatário receber o e-mail em seu programa de correio eletrônico, ao clicar em “Responder”, nem sempre a resposta será enviada para o remetente da mensagem. A resposta será enviada para o endereço especificado na opção *Reply-To*. Veremos no final deste capítulo outras informações extras que podemos enviar.

Apenas para seu conhecimento, existe ainda um quinto parâmetro da função *mail*, pelo qual podemos enviar informações extras à chamada do programa que envia os e-mails. Essas informações são as mesmas que poderíamos passar se o programa fosse chamado pela linha de comando do sistema operacional.

## Enviando e-mails em formato HTML

O envio de e-mails em formato HTML está sendo cada vez mais utilizado por diversos sites que necessitam dos recursos de correio eletrônico. Em vez de enviar um e-mail em formato texto, sem cores, figuras ou qualquer outra formatação, os desenvolvedores estão optando por enviar e-mails com todos os recursos que o HTML nos oferece.

Muitos sites costumam enviar um clipping (informativo sobre as novidades da semana, por exemplo) aos seus usuários. Com certeza será muito mais atrativo

se os usuários receberem um e-mail colorido e com imagens que receber um e-mail com texto puro. Provavelmente muitos deles nem sequer leriam um e-mail que possuísse uma enorme quantidade de texto sem formatação nenhuma.

No entanto, o formato texto é o padrão para o envio de e-mails por meio da função *mail* do PHP. Se tentarmos simplesmente colocar tags HTML em uma mensagem e enviá-la a algum e-mail, essas tags não serão reconhecidas pelo gerenciador de e-mails do destinatário. Veja o seguinte exemplo:

### exemplo15\_3.php

```
<?php  
    $mensagem = "<font color=\"#0000FF\">Teste</font>";  
    mail("joao@dominio.com.br", "Teste", $mensagem);  
?>
```

O objetivo seria enviar um e-mail escrito “Teste” em letras azuis, porém o dono do endereço joao@dominio.com.br receberá um e-mail escrito:

```
<font color="#0000FF">Teste</font>
```

Portanto para enviar e-mails em formato HTML não basta simplesmente colocar as tags. Além de colocá-las na mensagem, devemos informar o tipo de e-mail que está sendo enviado. Isso é feito por meio do quarto parâmetro da função *mail*, com o uso da opção *Content-Type*. Essa opção deve conter o valor *text/html* quando quisermos enviar e-mails em formato HTML.

Veja o exemplo:

### exemplo15\_4.php

```
<?php  
    $para = "joao@dominio.com.br";  
    $mensagem = "<font color=\"#0000FF\">Teste</font>";  
    mail($para, "Teste", $mensagem, "Content-Type: text/html");  
?>
```

Por meio do quarto parâmetro da função *mail* foi enviada uma informação extra (*Content-Type*) informando que este e-mail contém texto e tags HTML. Quando o destinatário receber este e-mail, essa informação será analisada por seu gerenciador, que fará a leitura adequada.

Para evitar a tarefa de ter de digitar os comandos HTML, é possível a utilização de um editor de páginas, como, por exemplo, o FrontPage ou o Dreamweaver. Após construir a página, basta copiar o código HTML que o editor gerou e trazê-lo para o programa PHP, atribuindo-o para a variável que deverá conter o conteúdo do e-mail a ser enviado.

É importante lembrar que nem todos os programas gerenciadores de e-mails lêem em formato HTML. Muitas pessoas podem utilizar um gerenciador que só lê em formato-texto, e dessa forma elas verão o texto em códigos, mesmo que a opção Content-Type contenha o valor text/html.

## **Lista dos cabeçalhos de e-mail (mail headers)**

Veremos agora uma lista das informações extras que podem ser enviadas em um e-mail. Essas informações são conhecidas como mail headers (cabeçalhos de e-mail). Logo sem seguida temos a descrição das principais delas:

- **Bcc:** é uma abreviatura para Blind Carbon Copy. É utilizado para enviar cópias da mensagem para outros endereços, sem que os destinatários conheçam os endereços para quais a mensagem foi enviada. Quando utilizado, o Bcc não aparece no cabeçalho do e-mail recebido pelo destinatário.
- **Cc:** é uma abreviatura para Carbon Copy, e serve para especificarmos destinatários adicionais. A diferença entre To e Cc é apenas conotativa, pois alguns programas tratam essas informações de maneiras diferentes no momento de enviar respostas.
- **Comments:** normalmente este header é adicionado por alguns programas para especificar quem enviou a mensagem. Algumas vezes os spammers (aqueles que enviam e-mails não solicitados) o adicionam manualmente com informações falsas.
- **Content-Transfer-Encoding:** é um header relacionado com MIME (Multi-purpose Internet Mail Extentions). Define como o conteúdo da mensagem deve ser interpretado. Normalmente é utilizado para conteúdos que não são do tipo texto.
- **Content-Type:** outro header relacionado com MIME. Informa como o programa de e-mails deve tratar a mensagem. Veja alguns tipos de MIME que são aceitos:

*text/plain*: documento em formato texto (ASCII).

*text/html*: documento em HTML.

*application/postscript*: documento PostScript.

*image/gif*: imagem codificada em formato GIF.

*image/jpeg*: imagem codificada em formato JPEG.

*audio/basic*: um arquivo de som.

*video/mpeg*: um vídeo codificado em mpeg.

*x-world/x-vrml*; arquivo do tipo VRML.

- Date: especifica a data em que a mensagem foi enviada. Alguns computadores podem ter seu relógio desconfigurado, portanto nem sempre podemos garantir que essa informação seja correta.
- Errors-To: especifica um endereço para que sejam informados os erros no envio da mensagem.
- From: contém o endereço do remetente da mensagem.
- Message-Id: é um identificador atribuído a uma mensagem. E-mails em que esta informação é vazia provavelmente foram forjados.
- Mime-Version: especifica a versão do protocolo MIME que está sendo utilizada pelo programa de e-mails.
- Priority: atribui uma prioridade a um e-mail. Alguns spammers enviam e-mails com prioridade “Urgente”, para tentar fazer com que seus e-mails sejam lidos. Muitos programas gerenciadores de e-mails não leem este tipo de header.
- Received: fornece um log detalhado do histórico da mensagem, tornando possível o conhecimento de sua origem.
- Reply-To: especifica o endereço para o qual a mensagem deve ser respondida. Quando a pessoa clicar em “Responder” em seu gerenciador de e-mails, a resposta será enviada para o e-mail especificado nesse header.
- Sender: este header é pouco comum em um e-mail. Normalmente X-Sender é utilizado em seu lugar para especificar que está enviando a mensagem. O Sender é um header mais confiável que o From.
- Subject: contém o título da mensagem.
- To: contém o endereço do(s) destinatário(s) da mensagem.
- X-Confirm-Reading-To: este header solicita uma confirmação de recebimento por parte do destinatário do e-mail, porém ele pode enviar ou não essa confirmação.
- X-Mailer: identifica o software que enviou o e-mail.

# Apêndice A

## Comandos gerais do PHP

Neste apêndice veremos uma rápida descrição de cada um dos principais comandos do PHP. Não serão mostrados todos os comandos da linguagem, mas veremos principalmente aqueles que estão relacionados com os conteúdos apresentados neste livro.

Os comandos foram divididos em diversas categorias, como: arrays, strings, variáveis, funções, classes e objetos, matemática, data e hora, sistema de arquivos (filesystem), diretórios, HTTP, FTP, URL, imagens, opções e informações do PHP e sessões.

### Arrays

| Função                | Descrição   |
|-----------------------|---|
| array                 | Cria um array.  |
| array_change_key_case | Retorna um array com todas as chaves em letras maiúsculas ou minúsculas.  |
| array_chunk           | Divide um array em diversos arrays com um determinado tamanho.  |
| array_count_values    | Conta todos os valores de um array.   |
| array_diff_assoc      | Retorna os valores do primeiro array que não existem nos demais arrays fornecidos. As chaves (índices) também são utilizadas na comparação. |
| array_diff            | Calcula a diferença entre arrays.   |
| array_fill            | Preenche um array com um determinado valor.   |
| array_filter          | Filtre elementos do array utilizando uma função de callback.  |
| array_flip            | Retorna um array com chaves e valores trocados.   |
| array_intersect       | Calcula a intersecção entre arrays.   |
| array_intersect_assoc | Calcula a intersecção entre arrays, comparando também as chaves dos elementos.  |
| array_key_exists      | Verifica se uma determinada chave existe em um array.   |

| Função                | Descrição (cont.)  |
|-----------------------|--|
| array_keys            | Retorna todas as chaves de um array.   |
| array_map             | Aplica o callback sobre os elementos dos arrays dados.   |
| array_merge           | Une dois ou mais arrays.   |
| array_merge_recursive | Une dois ou mais arrays recursivamente.  |
| array_multisort       | Ordena arrays multidimensionais.   |
| array_pad             | Preenche determinado tamanho de um array.  |
| array_pop             | Retira um elemento do fim do array.  |
| array_push            | Insere um elemento no fim do array.  |
| array_rand            | Retorna chaves para entradas randômicas de um array.   |
| array_reverse         | Retorna um array com seus elementos em ordem reversa.  |
| array_reduce          | Reduz um array a um valor simples.   |
| array_shift           | Retira um elemento do início do array.   |
| array_slice           | Retorna uma parte do array.  |
| array_splice          | Substitui uma parte de um array.   |
| array_sum             | Soma os elementos de um array.   |
| array_unique          | Remove valores duplicados de um array.   |
| array_unshift         | Coloca um ou mais elementos no início do array.  |
| array_values          | Retorna todos os valores de um array.  |
| array_walk            | Aplica determinada função sobre cada elemento do array.  |
| arsort                | Ordena um array em ordem reversa.  |
| asort                 | Ordena um array.   |
| compact               | Cria um array contendo variáveis e seus valores.   |
| count                 | Conta o número de elementos em uma variável ou array.  |
| current               | Retorna o elemento corrente de um array.   |
| each                  | Retorna o próximo par chave/valor de um array.   |
| end                   | Aponta o ponteiro interno do array para seu último elemento.   |
| extract               | Importa variáveis de um array para a tabela de símbolos.   |
| in_array              | Retorna TRUE se determinado valor existe no array.   |
| array_search          | Procura um valor dado em um array e retorna sua chave em caso de sucesso.                            |
| key                   | Busca uma chave de um array associativo.   |
| krsort                | Ordena um array por chave em ordem reversa.  |
| ksort                 | Ordena um array por chave.   |
| list                  | Atribui valores de um arrays a variáveis especificadas.  |
| natsort               | Ordena um array usando o algoritmo "natural order".  |
| natcasesort           | Ordena um array usando o algoritmo "natural order" não diferenciando letras maiúsculas e minúsculas. |
| next                  | Avança o ponteiro interno de um array.   |
| pos                   | Obtém o elemento corrente de um array.   |
| prev                  | Retrocede o ponteiro interno de um array.  |

| Função  | Descrição (cont.)  |
|---------|--|
| range   | Cria um array contendo determinada faixa de inteiros.                      |
| reset   | Coloca o ponteiro interno do array em seu primeiro elemento.               |
| rsort   | Ordena um array em ordem reversa.  |
| shuffle | Embaralha um array trocando a ordem de seus elementos de forma aleatória.  |
| sizeof  | Obtém o número de elementos de um array.                                   |
| sort    | Ordena um array.   |
| uasort  | Ordena um array com uma função definida pelo usuário e mantém a indexação. |
| uksort  | Ordena um array pelas chaves com uma função definida pelo usuário.         |
| usort   | Ordena um array por valores com uma função definida pelo usuário.          |

## Classes e objetos

| Função                 | Descrição   |
|------------------------|---|
| call_user_method_array | Chama um método do usuário com um array de parâmetros.                  |
| call_user_method       | Chama um método do usuário em um objeto específico.                     |
| class_exists           | Testa se a classe foi definida.   |
| get_class              | Retorna o nome da classe de um objeto.                                  |
| get_class_methods      | Retorna um array com o nome dos métodos da classe.                      |
| get_class_vars         | Retorna um array com as propriedades-padrão da classe.                  |
| get_declared_classes   | Retorna um array com o nome das classes definidas.                      |
| get_object_vars        | Retorna um array associativo das propriedades do objeto.                |
| get_parent_class       | Retorna o nome da classe-pai (parent) de um objeto.                     |
| is_subclass_of         | Determina se um objeto pertence a uma subclasse da classe especificada. |
| method_exists          | Testa se um método existe em uma classe.                                |

## Data e hora

| Função       | Descrição  |
|--------------|--|
| checkdate    | Valida uma data/hora.                              |
| date         | Formata uma data/hora.                             |
| getdate      | Retorna informações sobre data/hora.               |
| gettimeofday | Retorna a hora atual.                              |
| gmdate       | Formata uma data/hora GMT/CUT.                     |
| gmmktime     | Obtém uma data/hora GMT no formato UNIX timestamp. |

| Função     | Descrição (cont.)   |
|------------|---|
| gmstrftime | Formata uma data/hora GMT/CUT conforme as configurações locais. |
| localtime  | Obtém a hora local.   |
| microtime  | Retorna o UNIX timestamp corrente em microsegundos.             |
| mktime     | Obtém o UNIX timestamp a partir de uma data.                    |
| strftime   | Formata uma data/hora local conforme as configurações locais.   |
| time       | Retorna a data atual no formato UNIX timestamp.                 |
| strtotime  | Tenta traduzir uma string para o formato UNIX timestamp.        |

## Diretórios

| Função    | Descrição                                       |
|-----------|---|
| chroot    | Altera o diretório do root.                     |
| chdir     | Altera o diretório corrente.                    |
| dir       | Classe diretório.                               |
| closedir  | Fechá o handle do diretório.                    |
| getcwd    | Obtém o diretório corrente de trabalho.         |
| opendir   | Abre um handle de diretório.                    |
| readdir   | Lê uma entrada a partir do handle do diretório. |
| rewinddir | Retrocede o handle do diretório.                |

## FTP

| Função         | Descrição  |
|----------------|--|
| ftp_cdup       | Altera para o diretório que está um nível acima (parent).                    |
| ftp_chdir      | Altera o diretório em um servidor FTP.                                       |
| ftp_chmod      | Define as permissões de um arquivo via FTP.                                  |
| ftp_close      | Fechá uma conexão FTP.   |
| ftp_connect    | Abre uma conexão FTP.  |
| ftp_delete     | Exclui arquivos em um servidor FTP.  |
| ftp_exec       | Solicita a execução de um programa no servidor FTP.                          |
| ftp_fget       | Faz o download de um arquivo do servidor FTP e salva em um arquivo aberto.   |
| ftp_fput       | Envia de um arquivo aberto para um servidor FTP.                             |
| ftp_get_option | Retorna diversas configurações em tempo de execução da conexão FTP corrente. |
| ftp_get        | Faz o download de um arquivo do servidor FTP.                                |
| ftp_login      | Realiza o login em uma conexão FTP.  |
| ftp_mdtm       | Retorna a hora da última modificação de determinado arquivo.                 |
| ftp_mkdir      | Cria um diretório.   |

| Função          | Descrição (cont.)  |
|-----------------|--|
| ftp_nb_continue | Continua enviando ou recebendo um arquivo (não-bloqueante).                            |
| ftp_nb_fget     | Retorna um arquivo de um servidor FTP e o grava em um arquivo aberto (não-bloqueante). |
| ftp_nb_fput     | Armazena um arquivo aberto em um servidor FTP (não-bloqueante).                        |
| ftp_nb_get      | Retorna um arquivo de um servidor FTP e o grava em um arquivo local (não-bloqueante).  |
| ftp_nb_put      | Armazena um arquivo no servidor FTP (não-bloqueante).                                  |
| ftp_nlist       | Retorna uma lista dos arquivos em determinado diretório.                               |
| ftp_pasv        | Ativa ou desativa o modo passivo.  |
| ftp_put         | Envia (upload) um arquivo para um servidor FTP.  |
| ftp_pwd         | Retorna o nome do diretório corrente.  |
| ftp_quit        | Fechá uma conexão FTP.   |
| ftp_raw         | Envia um comando arbitrário para um servidor FTP.                                      |
| ftp_rawlist     | Retorna uma lista detalhada dos arquivos em determinado diretório.                     |
| ftp_rename      | Renomeia arquivos em um servidor FTP.  |
| ftp_rmdir       | Remove um diretório.   |
| ftp_set_option  | Define diversas opções do FTP em tempo de execução.                                    |
| ftp_site        | Envia um comando do tipo SITE ao servidor.   |
| ftp_size        | Retorna o tamanho de determinado arquivo.  |
| ftp_ssl_connect | Abre uma conexão segura SSL-FTP.   |
| ftp_systype     | Retorna o identificador do tipo de sistema do servidor remoto de FTP.                  |

## Funções

| Função                     | Descrição  |
|----------------------------|--|
| call_user_func_array       | Chama uma função do usuário passando um array de parâmetros.     |
| call_user_func             | Chama uma função do usuário passando o primeiro parâmetro.       |
| create_function            | Cria uma função anônima (estilo lambda).                         |
| func_get_arg               | Retorna um item da lista de argumentos.                          |
| func_get_args              | Retorna um array contendo a lista de argumentos da função.       |
| func_num_args              | Retorna o número de argumentos passados para uma função.         |
| function_exists            | Testa se uma função foi definida.                                |
| get_defined_functions      | Retorna um array contendo a lista de todas as funções definidas. |
| register_shutdown_function | Registra uma função para ser executada ao término do script.     |

## HTTP

| Função       | Descrição                              |
|--------------|--|
| header       | Envia um cabeçalho HTTP.               |
| headers_sent | Testa se os headers já foram enviados. |
| setcookie    | Envia um cookie.                       |

## Imagens

| Função                  | Descrição  |
|-------------------------|--|
| exif_imagetype          | Determina o tipo de uma imagem através da leitura de seus primeiros bytes.   |
| exif_read_data          | Lê os cabeçalhos EXIF de arquivos do tipo JPEG ou TIFF.  |
| exif_thumbnail          | Retorna a miniatura (thumbnail) embutida em uma imagem TIFF ou JPEG.   |
| gd_info                 | Retorna informações sobre a biblioteca GD atualmente instalada.  |
| image_type_to_mime_type | Obtém o tipo MIME de um tipo de imagem retornado pelas funções <i>getimagesize</i> , <i>exif_read_data</i> , <i>exif_thumbnail</i> e <i>exif_imagetype</i> . |
| image2wbmp              | Envia uma imagem do tipo WBMP para o browser ou para um arquivo.   |
| GetImageSize            | Retorna o tamanho de uma imagem GIF, JPEG, PNG ou SWF.   |
| ImageAlphaBlending      | Define o modo de transparência de uma imagem.  |
| ImageArc                | Desenha uma elipse parcial.  |
| ImageFilledArc          | Desenha uma elipse parcial preenchida.   |
| ImageEllipse            | Desenha uma elipse.  |
| ImageFilledEllipse      | Desenha uma elipse preenchida.   |
| ImageChar               | Desenha um caractere horizontalmente.  |
| ImageCharUp             | Desenha um caractere verticalmente.  |
| ImageColorAllocate      | Aloca uma cor para uma imagem.   |
| ImageColorAllocateAlpha | Aloca uma cor para uma imagem. É semelhante à função <i>ImageColorAllocate</i> , possuindo um parâmetro adicional para definir a transparência.              |
| ImageColorDeAllocate    | Desaloca uma cor para uma imagem.  |
| ImageColorAt            | Retorna o índice da cor de um pixel.   |
| ImageColorClosest       | Retorna o índice da cor mais próxima à especificada.   |
| ImageColorClosestAlpha  | Obtém o índice da cor mais próxima à cor especificada, considerando o parâmetro de transparência.  |
| ImageColorClosestHWB    | Obtém o índice da cor que possui o tom, branco e preto mais próximo da cor especificada.   |
| ImageColorExact         | Retorna o índice de uma cor especificada.  |

| Função                  | Descrição (cont.)   |
|-------------------------|---|
| ImageColorExactAlpha    | Retorna o índice de uma cor especificada + alpha.                             |
| ImageColorResolve       | Retorna o índice de uma cor ou sua alternativa mais próxima.                  |
| ImageColorResolveAlpha  | Retorna o índice de uma cor+alpha ou sua alternativa mais próxima.            |
| ImageGammaCorrect       | Aplica uma correção gamma em uma imagem GD.                                   |
| ImageColorSet           | Define a cor para o índice da paleta especificada.                            |
| ImageColorsForIndex     | Obtém as cores para um índice.  |
| ImageColorsTotal        | Retorna o número de cores de uma paleta de imagens.                           |
| ImageColorTransparent   | Define uma cor como transparente.   |
| ImageCopy               | Copia parte de uma imagem.  |
| ImageCopyMerge          | Copia e une parte de uma imagem.  |
| ImageCopyMergeGray      | Copia e une parte de uma imagem com tons de cinza.                            |
| ImageCopyResized        | Copia e redimensiona parte de uma imagem.                                     |
| ImageCopyResampled      | Copia e redimensiona parte de uma imagem com resampling.                      |
| ImageCreate             | Cria uma nova imagem.   |
| ImageCreateTrueColor    | Cria uma nova imagem true color.  |
| ImageTrueColorToPalette | Converte uma imagem de true color para palette.                               |
| ImageCreateFromGD2      | Cria uma nova imagem a partir de um arquivo GD2 ou de um URL.                 |
| ImageCreateFromGD2part  | Cria uma nova imagem baseada em uma parte de um arquivo GD2 ou de um URL.     |
| ImageCreateFromGD       | Cria uma nova imagem a partir de um arquivo GD ou de um URL.                  |
| ImageCreateFromGIF      | Cria uma nova imagem a partir de um arquivo GIF ou de um URL.                 |
| ImageCreateFromJPEG     | Cria uma nova imagem a partir de um arquivo JPEG ou de um URL.                |
| ImageCreateFromPNG      | Cria uma nova imagem a partir de um arquivo PNG ou de um URL.                 |
| ImageCreateFromWBMP     | Cria uma nova imagem a partir de um arquivo WBMP ou de um URL.                |
| ImageCreateFromString   | Retorna um identificador representando a imagem obtida de determinada string. |
| ImageCreateFromXBM      | Cria uma nova imagem a partir de um arquivo XBM ou de um URL.                 |
| ImageCreateFromXPM      | Cria uma nova imagem a partir de um arquivo XPM ou de um URL.                 |
| ImageDashedLine         | Desenha uma linha pontilhada.   |
| ImageDestroy            | Destrói uma imagem.   |
| ImageEllipse            | Desenha uma elipse.   |
| ImageFill               | Preenche uma imagem ou parte dela.  |

| Função               | Descrição (cont.)   |
|----------------------|---|
| ImageFilledArc       | Desenha um elipse parcial preenchida com uma determinada cor.   |
| ImageFilledEllipse   | Desenha um elipse preenchida com uma determinada cor.   |
| ImageFilledPolygon   | Desenha um polígono preenchido.   |
| ImageFilledRectangle | Desenha um retângulo preenchido.  |
| ImageFillToBorder    | Preenche uma imagem com determinada cor.  |
| ImageFontHeight      | Obtém a altura da fonte.  |
| ImageFontWidth       | Obtém a largura da fonte.   |
| ImageFtBox           | Usa fontes FreeType 2 para definir uma caixa ( <i>bounding box</i> ) para o texto.  |
| ImageFtText          | Escreve um texto em uma imagem usando fontes FreeType 2.  |
| ImageGD2             | Envia uma imagem GD2 para o browser ou para um arquivo.   |
| ImageGD              | Envia uma imagem GD para o browser ou para um arquivo.  |
| ImageGIF             | Envia uma imagem GIF para o browser ou para um arquivo.   |
| ImagePNG             | Envia uma imagem PNG para o browser ou para um arquivo.   |
| ImageJPEG            | Envia uma imagem JPEG para o browser ou para um arquivo.  |
| ImageWBMP            | Envia uma imagem WBMP para o browser ou para um arquivo.  |
| ImageInterlace       | Habilita ou desabilita o entrelaçamento.  |
| ImageIsTrueColor     | Verifica se uma imagem é <i>true color</i> (24 bits).   |
| ImageLine            | Desenha uma linha.  |
| ImageLoadFont        | Carrega uma nova fonte.   |
| ImagePaletteCopy     | Copia a paleta de cores de uma imagem para outra.   |
| ImagePolygon         | Desenha um polígono.  |
| ImagePSBBox          | Usa uma fonte Postscript e define uma caixa para o texto.   |
| ImagePSCopyFont      | Faz a cópia de uma fonte já carregada, para posterior modificação.  |
| ImagePSEncodeFont    | Altera o vetor de codificação de caracteres para uma fonte.   |
| ImagePSExtendFont    | Estende ou condensa uma fonte.  |
| ImagePSFreeFont      | Libera a memória usada por uma fonte PostScript do tipo 1.  |
| ImagePSLoadFont      | Carrega a fonte PostScript tipo 1 de um arquivo.  |
| ImagePsExtendFont    | Estende ou condensa uma fonte.  |
| ImagePsSlantFont     | Inclina uma fonte.  |
| ImagePSText          | Desenha um texto sobre uma imagem.  |
| ImageRectangle       | Desenha um retângulo.   |
| ImageRotate          | Realiza uma rotação em uma imagem, considerando o ângulo especificado.  |
| ImageSaveAlpha       | Usada em imagens PNG para configurar a flag que indica se as informações sobre o canal alfa (alpha channel) devem ser salvas. |

| Função            | Descrição (cont.)  |
|-------------------|--|
| ImageSetPixel     | Define um pixel simples.   |
| ImageSetBrush     | Define uma imagem para desenho de linhas.                            |
| ImageSetStyle     | Define o estilo para o desenho de linhas.                            |
| ImageSetTile      | Define uma imagem (tile) para o preenchimento de uma área.           |
| ImageSetThickness | Define a espessura de uma linha.                                     |
| ImageString       | Desenha uma string horizontalmente.                                  |
| ImageStringUp     | Desenha uma string verticalmente.                                    |
| ImageSX           | Retorna a largura da imagem.   |
| ImageSY           | Retorna a altura da imagem.  |
| ImageTTFBox       | Utiliza fontes TypeType e define uma caixa para o texto.             |
| ImageTTFFText     | Escreve o texto na imagem usando fontes TrueType.                    |
| ImageTypes        | Retorna o tipo das imagens suportadas pela compilação atual do PHP.  |
| iptcembed         | Embute dados binários IPTC em uma imagem JPEG.                       |
| iptcparse         | Analisa um bloco binário IPTC em tags simples.                       |
| jpeg2wbmp         | Converte uma imagem JPEG para o formato WBMP.                        |
| png2wbmp          | Converte uma imagem PNG para o formato WBMP.                         |
| read_exif_data    | Apelido ( <i>alias</i> ) para a função <code>exif_read_data</code> . |

## Matemática

| Função       | Descrição  |
|--------------|--|
| abs          | Valor absoluto.                                  |
| acos         | Arco cosseno.                                    |
| acosh        | Arco cosseno hiperbólico.                        |
| asin         | Arco seno.                                       |
| asinh        | Arco seno hiperbólico.                           |
| atan         | Arco tangente.                                   |
| atan2        | Arco tangente de duas variáveis.                 |
| atanh        | Arco tangente hiperbólico.                       |
| base_convert | Converte um número entre duas bases arbitrárias. |
| bindec       | Converte de binário para decimal.                |
| ceil         | Arredonda frações para cima.                     |
| cos          | Cosseno.   |
| cosh         | Cosseno hiperbólico.                             |
| decbin       | Converte de decimal para binário.                |
| dechex       | Converte de decimal para hexadecimal.            |
| decoct       | Converte de decimal para octal.                  |

| Função        | Descrição (cont.)   |
|---------------|---|
| deg2rad       | Converte um número de graus para radianos.  |
| exp           | Eleva "e" a determinada potência.   |
| expm1         | Retorna $\exp(número) - 1$ , calculado de forma precisa mesmo quando o número for próximo de zero.        |
| floor         | Arredonda frações para baixo.   |
| getrandmax    | Mostra o maior valor randômico possível.  |
| hexdec        | Converte de hexadecimal para decimal.   |
| hypot         | Aplica a fórmula da hipotenusa aos valores fornecidos, ou seja, $\sqrt{\text{num1}^2 + \text{num2}^2}$ .  |
| is_finite     | Verifica se um valor em ponto flutuante é finito dentro dos limites da plataforma.                        |
| is_infinite   | Verifica se um valor em ponto flutuante é infinito dentro dos limites da plataforma. Exemplo: $\log(0)$ . |
| is_nan        | Retorna <i>true</i> se o valor fornecido não for um número ( <i>Not a Number</i> ).                       |
| lcg_value     | Retorna um número pseudo-aleatório entre 0 e 1.   |
| log           | Logaritmo natural.  |
| log10         | Logaritmo de base 10.   |
| log1p         | Retorna $\log(1 + \text{número})$ , calculado de forma precisa mesmo quando o número for próximo de zero. |
| max           | Retorna o maior valor.  |
| min           | Retorna o menor valor.  |
| mt_rand       | Gera um valor randômico melhor.   |
| mt_srand      | Define o melhor gerador de números randômicos.  |
| mt_getrandmax | Mostra o maior valor randômico possível.  |
| number_format | Formata um número.  |
| octdec        | Converte de octal para decimal.   |
| pi            | Retorna o valor do número pi.   |
| pow           | Eleva um número a outro.  |
| rad2deg       | Converte um número de radianos para graus.  |
| rand          | Gera um valor aleatório.  |
| round         | Arredonda um número em ponto flutuante.   |
| sin           | Seno.   |
| sinh          | Seno hiperbólico.   |
| sqrt          | Raiz quadrada.  |
| srand         | Define um gerador de números aleatórios.  |
| tan           | Tangente.   |
| tanh          | Tangente hiperbólica.   |

## Opções e informações do PHP

| Função                   | Descrição   |
|--------------------------|---|
| assert                   | Checa se uma declaração é falsa.  |
| assert_options           | Define/obtém diversas flags sobre o assert.   |
| extension_loaded         | Verifica se uma extensão foi carregada.   |
| dl                       | Carrega uma extensão do PHP em tempo de execução.   |
| getenv                   | Obtém o valor de uma variável de ambiente.  |
| get_cfg_var              | Obtém o valor de uma opção de configuração do PHP.  |
| get_current_user         | Obtém o nome do dono do script PHP corrente.  |
| get_defined_constants    | Retorna um array associativo com os nomes e valores das constantes definidas.                                 |
| get_loaded_extensions    | Retorna um array com o nome dos módulos carregados.   |
| get_extension_funcs      | Retorna um array com os nomes das funções de um módulo.   |
| get_required_files       | Retorna um array com os nomes dos arquivos requeridos em um script.   |
| get_included_files       | Retorna um array com os nomes dos arquivos incluídos em um script.  |
| get_magic_quotes_gpc     | Retorna a configuração ativa.   |
| get_magic_quotes_runtime | Retorna a configuração ativa de <i>magic_quotes_runtime</i> .   |
| getlastmod               | Retorna a hora de modificação da última página.   |
| getmygid                 | Retorna o identificador do grupo (GID) do proprietário do script corrente.                                    |
| getmyinode               | Retorna o inode do script corrente.   |
| getmypid                 | Retorna o ID do processo PHP.   |
| getmyuid                 | Retorna o UID do dono do script.  |
| getopt                   | Retorna um array associativo com pares opções/argumentos, baseado na lista de argumentos da linha de comando. |
| getrusage                | Obtém os recursos que estão sendo utilizados.   |
| ini_alter                | Altera o valor de uma opção de configuração.  |
| ini_get_all              | Obtém o valor de todas opções de configuração.  |
| ini_get                  | Obtém o valor de uma determinada opção de configuração.   |
| ini_restore              | Restaura o valor de uma opção de configuração.  |
| ini_set                  | Define o valor de uma opção de configuração.  |
| php_ini_scanned_files    | Retorna uma lista de arquivos .ini analisados após o php.ini. Os elementos são separados por vírgulas.        |
| phpcredits               | Imprime os créditos do PHP.   |
| phpinfo                  | Imprime informações sobre o PHP.  |
| phpversion               | Retorna a versão do PHP.  |
| php_logo_guid            | Obtém o logo guid.  |
| php_sapi_name            | Retorna o tipo de interface entre o servidor Web e o PHP.   |

| Função                   | Descrição (cont.)  |
|--------------------------|--|
| php_uname                | Retorna informações sobre o sistema operacional.                       |
| putenv                   | Define o valor de uma variável de ambiente.                            |
| set_magic_quotes_runtime | Define a configuração ativa de <i>magic_quotes_runtime</i> .           |
| set_time_limit           | Limita o tempo máximo de execução.                                     |
| version_compare          | Compara duas strings padronizadas, contendo o número da versão do PHP. |
| zend_logo_guid           | Obtém o zend guid.   |
| zend_version             | Obtém a versão da ferramenta Zend.                                     |

## PDF

| Função                    | Descrição   |
|---------------------------|---|
| pdf_add_bookmark          | Adiciona um bookmark à página corrente.   |
| pdf_add_launchlink        | Cria um link para um arquivo, URL ou outro recurso.   |
| pdf_add_locallink         | Adiciona um link local.   |
| pdf_add_note              | Adiciona notas em um documento PDF.   |
| pdf_add_pdflink           | Adiciona à página corrente um link para um arquivo PDF externo.   |
| pdf_add_thumbnail         | Adiciona à página corrente uma miniatura ( <i>thumbnail</i> ).  |
| pdf_add_weblink           | Adiciona um weblink à página corrente através da definição de uma área retangular.  |
| pdf_arc                   | Desenha um arco em sentido anti-horário.  |
| pdf_arcn                  | Desenha um arco em sentido horário.   |
| pdf_attach_file           | Anexa um arquivo a um documento PDF na forma de um ícone.   |
| pdf_begin_page            | Inicia uma nova página no documento.  |
| pdf_begin_pattern         | Inicia a definição de um novo padrão (pattern), que poderá ser utilizado para o preenchimento de objetos.                   |
| pdf_begin_template        | Inicia a definição de um novo modelo (template).  |
| pdf_circle                | Desenha um círculo.   |
| pdf_clip                  | Seleciona o caminho atual, definindo-o como clip path.  |
| pdf_close_image           | Fecha uma imagem.   |
| pdf_close_pdi_page        | Fecha uma página aberta com a função <i>pdf_open_pdi_page</i> .   |
| pdf_close_pdi             | Fecha um documento PDI aberto com a função <i>pdf_open_pdi</i> .  |
| pdf_close                 | Fecha um objeto PDF e libera todos os recursos associados a ele.  |
| pdf_closepath_fill_stroke | Completa o caminho corrente (adicionando uma linha do último ao primeiro ponto), preenchendo-o e desenhando-o no documento. |
| pdf_closepath_stroke      | Completa o caminho corrente (adicionando uma linha do último ao primeiro ponto) e o desenha no documento.                   |

| Função                | Descrição (cont.)   |
|-----------------------|---|
| pdf_closepath         | Completa o caminho corrente adicionando uma linha do último ao primeiro ponto.                  |
| pdf_concat            | Concatena uma matriz à atual matriz utilizada para transformação de textos e gráficos (CTM).    |
| pdf_continue_text     | Escreve um texto na próxima linha do documento.   |
| pdf_curveto           | Desenha uma curva Bezier a partir do ponto corrente, utilizando outros três pontos de controle. |
| pdf_delete            | Exclui um objeto PDF e libera todos os recursos associados a ele.                               |
| pdf_end_page          | Encerra uma página do documento, iniciada pela função <i>pdf_begin_page</i> .                   |
| pdf_end_pattern       | Enderra a definição de um padrão, iniciada pela função <i>pdf_begin_pattern</i> .               |
| pdf_end_template      | Encerra a definição de um modelo (template), iniciada pela função <i>pdf_begin_template</i> .   |
| pdf_fill_stroke       | Preenche o caminho atual e o desenha no documento.  |
| pdf_fill              | Preenche o caminho atual.   |
| pdf_findfont          | Prepara uma fonte para uso posterior.   |
| pdf_get_buffer        | Obtém o conteúdo do buffer onde estão localizados os dados referentes ao documento PDF gerado.  |
| pdf_get_majorversion  | Retorna o número da maior versão disponível da biblioteca PDFlib.                               |
| pdf_get_minorversion  | Retorna o número da menor versão disponível da biblioteca PDFlib.                               |
| pdf_get_parameter     | Obtém o valor de um determinado parâmetro do documento.   |
| pdf_get_pdi_parameter | Obtém o valor de um determinado parâmetro de um documento PDI.                                  |
| pdf_get_pdi_value     | Obtém o valor de um parâmetro numérico de um documento PDI.                                     |
| pdf_get_value         | Obtém o valor de um determinado parâmetro numérico do documento.                                |
| pdf_initgraphics      | Reinicia todos os parâmetros gráficos e de cores, fazendo-os assumir os seus valores padrão.    |
| pdf_linetoo           | Desenha uma linha.  |
| pdf_makespotcolor     | Define a cor atual como sendo uma cor especial chamada spot.                                    |
| pdf_moveto            | Define o ponto corrente.  |
| pdf_new               | Cria um novo objeto PDF.  |
| pdf_open_CCITT        | Abre um novo arquivo de imagem contendo dados comprimidos do tipo CCITT.                        |
| pdf_open_file         | Cria um novo arquivo PDF.   |
| pdf_open_image_file   | Lê uma imagem de um arquivo.  |
| pdf_open_image        | Usa dados de uma imagem a partir de diversas origens.   |

| Função                | Descrição (cont.)  |
|-----------------------|--|
| pdf_open_memory_image | Abre uma imagem criada com as funções de imagem do PHP, tornando-a disponível para o objeto PDF.                     |
| pdf_open_pdi_page     | Abre uma página no documento PDI especificado.   |
| pdf_open_pdi          | Abre o documento PDF especificado e retorna o identificador do objeto PDI.   |
| pdf_place_image       | Insere uma imagem na página corrente.  |
| pdf_place_pdi_page    | Insere uma página de um documento PDI em um documento PDF.   |
| pdf_rect              | Desenha um retângulo.  |
| pdf_restore           | Restaura o último estado dos gráficos salvo pela função <i>pdf_save</i> .  |
| pdf_rotate            | Realiza uma rotação no sistema de coordenadas.   |
| pdf_save              | Salva o estado gráfico corrente do documento.  |
| pdf_scale             | Define a escala do sistema de coordenadas.   |
| pdf_set_border_color  | Define a cor da borda a ser utilizada em torno de links e anotações.   |
| pdf_set_border_dash   | Define o estilo de pontilhado a ser utilizado em torno de links e anotações.   |
| pdf_set_border_style  | Define o estilo de borda a ser utilizado em torno de links e anotações.  |
| pdf_set_info          | Define os campos de informação sobre um documento, que serão exibidos quando o usuário visualizar suas propriedades. |
| pdf_set_parameter     | Altera o valor de um determinado parâmetro do documento.   |
| pdf_set_text_pos      | Define a posição corrente para inserir textos no documento.  |
| pdf_set_value         | Altera o valor de um determinado parâmetro numérico do documento.  |
| pdf_setcolor          | Define a cor de preenchimento ou de contorno.  |
| pdf_setdash           | Define o padrão de pontilhado.   |
| pdf_setflat           | Define a distância máxima ( <i>flatness</i> ) entre o caminho de um objeto e uma aproximação criada com linhas.      |
| pdfSetFont            | Define a fonte corrente, assim como o seu tamanho.   |
| pdf_setlinecap        | Altera o valor do parâmetro <i>linecap</i> , que faz com que a linha seja desenhada de forma diferente.              |
| pdf_setlinejoin       | Configura o tipo de junção entre duas linhas.  |
| pdf_setlinewidth      | Define a espessura da linha que será utilizada para desenhar as figuras.   |
| pdf_setmatrix         | Define a matriz corrente para transformações.  |
| pdf_setmiterlimit     | Configura o limite <i>miter</i> para um valor maior ou igual a 1.  |
| pdf_setpolydash       | Define um padrão de pontilhado mais complexo, baseado em um array.   |
| pdf_show_boxed        | Cria uma caixa de texto.   |

| Função          | Descrição (cont.)   |
|-----------------|---|
| pdf_show_xy     | Escreve um texto em um determinado ponto do documento, utilizando a fonte corrente. |
| pdf_show        | Escreve um texto no ponto corrente do documento, utilizando a fonte corrente.       |
| pdf_skew        | Inclina o sistema de coordenadas do documento nos eixos x e y.                      |
| pdf_stringwidth | Retorna a largura de um texto utilizando uma determinada fonte.                     |
| pdf_stroke      | Desenha uma linha em torno do caminho corrente.                                     |
| pdf_translate   | Define a origem do sistema de coordenadas.  |

## Sessões

| Função                    | Descrição  |
|---------------------------|--|
| session_start             | Inicializa uma sessão.   |
| session_destroy           | Destroi todos os dados registrados em uma sessão.                                  |
| session_name              | Obtém e/ou define o nome da sessão corrente.                                       |
| session_module_name       | Obtém e/ou define o módulo da sessão corrente.                                     |
| session_save_path         | Obtém e/ou define o caminho de onde serão armazenados os dados da sessão corrente. |
| session_id                | Obtém e/ou define o ID da sessão corrente.   |
| session_regenerate_id     | Gera um novo identificador para a sessão corrente.                                 |
| session_register          | Registra uma ou mais variáveis na sessão corrente.                                 |
| session_unregister        | Exclui uma variável da sessão corrente.  |
| session_unset             | Limpa todas as variáveis da sessão.  |
| session_is_registered     | Verifica se uma variável está registrada em uma sessão.                            |
| session_get_cookie_params | Obtém parâmetros dos cookies da sessão.  |
| session_set_cookie_params | Define parâmetros dos cookies da sessão.   |
| session_decode            | Decodifica dados da sessão em uma string.  |
| session_encode            | Codifica dados da sessão corrente em uma string.                                   |
| session_set_save_handler  | Define funções de armazenamento da sessão em nível de usuário.                     |
| session_cache_limiter     | Obtém e/ou define o limitador atual da cache.                                      |
| session_cache_expire      | Retorna o tempo (em minutos) de expiração da cache.                                |
| session_write_close       | Encerra a sessão corrente e armazena os seus dados.                                |

## Sistema de arquivos (Filesystem)

| Função            | Descrição   |
|-------------------|---|
| basename          | Retorna o nome do arquivo em um caminho.                                    |
| chgrp             | Altera o grupo de um arquivo.   |
| chmod             | Altera as permissões de um arquivo.   |
| chown             | Altera o dono de um arquivo.  |
| clearstatcache    | Limpa a cache do arquivo.   |
| copy              | Copia arquivos.   |
| delete            | Exclui um arquivo.  |
| dirname           | Retorna o nome do diretório contido em um caminho.                          |
| diskfreespace     | Retorna o espaço disponível em um diretório.                                |
| disk_total_space  | Retorna o tamanho total de um diretório.                                    |
| diskfreespace     | Apelido ( <i>alias</i> ) para a função <i>disk_free_space()</i> .           |
| fclose            | Fechá um ponteiro de arquivo.   |
| feof              | Testa se o ponteiro está no final do arquivo.                               |
| fflush            | Grava os dados da memória em um arquivo.                                    |
| fgetc             | Obtém caracteres a partir da posição do ponteiro.                           |
| fgetcsv           | Obtém uma linha do arquivo e converte em campos CSV.                        |
| fgets             | Obtém uma linha do arquivo.   |
| fgetss            | Obtém uma linha do arquivo e retira as tags HTML.                           |
| file              | Lê um arquivo inteiro e o armazena em um array.                             |
| file_exists       | Testa se determinado arquivo existe.  |
| file_get_contents | Lê o conteúdo inteiro de um arquivo para uma string.                        |
| file_put_contents | Escreve uma string em um arquivo.   |
| fileatime         | Obtém a hora do último acesso ao arquivo.                                   |
| filectime         | Obtém a hora de alteração do descritor do arquivo.                          |
| filegroup         | Retorna o grupo do arquivo.   |
| fileinode         | Retorna o inode do arquivo.   |
| filemtime         | Retorna a hora que o arquivo foi modificado.                                |
| fileowner         | Retorna o dono do arquivo.  |
| fileperms         | Retorna as permissões do arquivo.   |
| filesize          | Retorna o tamanho do arquivo.   |
| filetype          | Obtém o tipo do arquivo.  |
| flock             | Bloqueia um arquivo.  |
| fnmatch           | Testa se o nome do arquivo possui um determinado padrão ( <i>pattern</i> ). |
| fopen             | Abre um arquivo ou um URL.  |
| fpassthru         | Imprime os dados restantes de um arquivo.                                   |
| fputs             | Escreve em um arquivo.  |

| Função             | Descrição (cont.)   |
|--------------------|---|
| fread              | Lê um arquivo (binário).  |
| fscanf             | Interpreta a entrada de um arquivo conforme determinado formato.                      |
| fseek              | Posiciona o ponteiro do arquivo em um local determinado.                              |
| fstat              | Obtém informações sobre um arquivo.   |
| ftell              | Retorna a posição do ponteiro do arquivo.   |
| truncate           | Trunca um arquivo em determinado tamanho.   |
| fwrite             | Escreve em um arquivo.  |
| glob               | Busca por diretórios e arquivos que possuem um determinado padrão ( <i>pattern</i> ). |
| is_dir             | Testa se o arquivo é um diretório.  |
| is_executable      | Testa se o arquivo é executável.  |
| is_file            | Testa se o arquivo é regular.   |
| is_link            | Testa se o arquivo é um link simbólico.   |
| is_readable        | Testa se o arquivo tem permissão de leitura.  |
| is_writable        | Testa se o arquivo tem permissão de escrita.  |
| is_writeable       | Tem a mesma função do comando <code>is_writable</code> .                              |
| is_uploaded_file   | Testa se o arquivo foi enviado por upload via HTTP POST.                              |
| link               | Cria um link físico.  |
| linkinfo           | Obtém informações sobre um link.  |
| mkdir              | Cria um diretório.  |
| move_uploaded_file | Move um arquivo enviado por upload.   |
| parse_ini_file     | Analisa um arquivo de configuração (.ini).  |
| pathinfo           | Retorna informações sobre o caminho (path).   |
| pclose             | Fecha o ponteiro para um processo.  |
| popen              | Abre o ponteiro para um processo.   |
| readfile           | Lê um arquivo e o imprime na saída-padrão.  |
| readlink           | Retorna o alvo de um link simbólico.  |
| rename             | Renomeia um arquivo.  |
| rewind             | Retrocede a posição de um ponteiro de arquivo.  |
| rmdir              | Remove um diretório.  |
| stat               | Obtém informações sobre um arquivo.   |
| lstat              | Obtém informações sobre um link simbólico.  |
| realpath           | Retorna o nome absoluto do caminho.   |
| set_file_buffer    | Configura o buffer de arquivo.  |
| symlink            | Cria um link simbólico.   |
| tmpnam             | Cria um nome único de arquivo.  |
| tmpfile            | Cria um arquivo temporário.   |
| touch              | Define a hora de modificação de um arquivo.   |
| umask              | Altera a máscara atual.   |
| unlink             | Exclui um arquivo.  |

## Strings

| Função                     | Descrição  |
|----------------------------|--|
| addCSlashes                | Coloca uma barra invertida (\) antes de determinados caracteres.   |
| addSlashes                 | Coloca \ antes dos caracteres ', " e \.  |
| bin2hex                    | Converte um dado de binário para hexadecimal.  |
| chop                       | Remove espaços e quebras de linha do final da string.  |
| chr                        | Retorna um caractere específico.   |
| chunk_split                | Divide uma string em pequenos pedaços.   |
| convert_cyr_string         | Converte um caractere do tipo Cyrillic para outro.   |
| count_chars                | Retorna informações sobre caracteres usados em uma string.   |
| crc32                      | Calcula o crc32 polinomial de uma string.  |
| crypt                      | Codifica uma string.   |
| echo                       | Imprime uma string em determinada saída.   |
| explode                    | Forma um array através de uma string.  |
| get_html_translation_table | Retorna a tabela de tradução usada pelas funções <code>htmlentities()</code> e <code>htmlspecialchars()</code> . |
| hebrev                     | Converte texto lógico Hebrew para texto visual.  |
| hebrevc                    | Converte texto lógico Hebrew para texto visual convertendo o caractere \n para "<br>\n".                         |
| html_entity_decode         | Converte todas entidades HTML para os seus respectivos caracteres.   |
| htmlentities               | Converte todos os caracteres aplicáveis para entidades HTML.   |
| htmlspecialchars           | Converte caracteres especiais para entidades HTML.   |
| implode                    | Forma uma string por meio da união dos elementos de um array.  |
| join                       | Similar à função <code>implode</code> .  |
| levenshtein                | Calcula a distância Levenshtein entre duas strings.  |
| localeconv                 | Retorna um array contendo informações numéricas e monetárias.  |
| ltrim                      | Remove espaços em branco do inicio de uma string.  |
| md5_file                   | Calcula o hash MD5 de um determinado nome de arquivo.  |
| md5                        | Calcula o hash MD5 de uma string.  |
| metaphone                  | Calcula a chave metaphone de uma string.   |
| money_format               | Aplica um determinado formato de moeda a um número.  |
| nl_langinfo                | Retorna informações sobre algumas configurações do sistema, como formatos e localidade.                          |
| nl2br                      | Insere quebras de linha HTML antes de todas as quebras de linha de uma string.                                   |
| number_format              | Formato um número, permitindo definir o separar decimal e de milhar.   |

| Função                  | Descrição (cont.)   |
|-------------------------|---|
| ord                     | Retorna o valor ASCII de um caractere.  |
| parse_str               | Divide uma string em variáveis.   |
| print                   | Imprime o valor de uma string.  |
| printf                  | Imprime o valor de uma string com formatação.   |
| quoted_printable_decode | Converte uma string para 8 bit.   |
| quotemeta               | Coloca o caractere \ antes de determinados caracteres.                                    |
| rtrim                   | Remove espaços no final da string, incluindo quebras de linha.                            |
| sscanf                  | Divide uma string de acordo com um formato.   |
| setlocale               | Ajusta informações do local.  |
| sha1_file               | Calcula o hash sha1 (Secure Hash Algorithm 1) de um determinado arquivo.                  |
| sha1                    | Calcula o hash sha1 (Secure Hash Algorithm 1) de uma string.                              |
| similar_text            | Calcula a similaridade entre duas strings.  |
| soundex                 | Calcula a chave soundex de uma string.  |
| sprintf                 | Retorna uma string formatada.   |
| str_replace             | Versão case-insensitive (não diferencia maiúsculas e minúsculas) da função str_replace(). |
| str_rot13               | Realiza uma transformação rot13 em uma string.  |
| str_shuffle             | Retorna uma string com a posição de seus caracteres alterada de forma aleatória.          |
| str_word_count          | Conta o número de palavras existentes em uma string.                                      |
| strncasecmp             | Comparação binária (case-insensitive) dos n primeiros caracteres.                         |
| strcasecmp              | Comparação binária (case-insensitive).  |
| strchr                  | Encontra a primeira ocorrência de um caractere.   |
| strcmp                  | Comparação binária.   |
| strcoll                 | Compara duas strings no local correto.  |
| strcspn                 | Encontra o tamanho do segmento inicial da string que não contém a máscara de comparação.  |
| strip_tags              | Retira tags HTML e PHP de uma string.   |
| stripcslashes           | Retira o caractere \ colocado com a função addslashes().                                  |
| stripslashes            | Retira o caractere \ colocado com a função addslashes().                                  |
| stristr                 | Versão case-insensitive da função strstr().   |
| strlen                  | Obtém o tamanho de uma string.  |
| strnatcasecmp           | Compara strings usando o algoritmo "natural order".                                       |
| strnatcasecmp           | Faz uma comparação case-insensitive de strings usando o algoritmo "natural order".        |
| strncmp                 | Comparação binária dos n primeiros caracteres.  |
| str_pad                 | Preenche um certo tamanho da string com outra string.                                     |
| strpos                  | Encontra a posição da primeira ocorrência de uma string.                                  |

| Função         | Descrição (cont.)   |
|----------------|---|
| strrchr        | Encontra a última ocorrência de um caractere.   |
| str_repeat     | Repete uma string.  |
| strrev         | Reverte uma string.   |
| strrpos        | Encontra a posição da última ocorrência de um caractere em uma string.                          |
| strspn         | Encontra o tamanho do segmento inicial da string que contém a máscara de comparação.            |
| strstr         | Encontra a primeira ocorrência de uma string.   |
| strtok         | Divide uma string em partes conforme um parâmetro.  |
| strtolower     | Transforma uma string em letras minúsculas.   |
| strtoupper     | Transforma uma string em letras maiúsculas.   |
| str_replace    | Substitui todas as ocorrências de uma string por outra.   |
| strtr          | Substitui alguns caracteres por outros.   |
| substr         | Retorna uma parte da string.  |
| substr_count   | Conta o número de ocorrências de uma substring.   |
| substr_replace | Substitui textos dentro de uma substring.   |
| trim           | Remove os espaços em branco do início e do fim da string.                                       |
| ucfirst        | Transforma o primeiro caractere de uma string em maiúsculo.                                     |
| ucwords        | Transforma em maiúsculo o primeiro caractere de cada palavra contida em uma string.             |
| vprintf        | Exibe os valores de um array de acordo com o formato especificado. Não possui valor de retorno. |
| vsprintf       | Retorna os valores de um array de acordo com o formato especificado.                            |
| wordwrap       | Coloca quebras de linha em uma string após determinado número de caracteres.                    |

## URL

| Função        | Descrição                                     |
|---------------|---|
| base64_decode | Decodifica dados codificados com MIME base64. |
| base64_encode | Codifica dados com MIME base64.               |
| parse_url     | Separa uma URL e retorna seus componentes.    |
| urlencode     | Decodifica uma URL codificada.                |
| urldecode     | Codifica uma URL.                             |

## Variáveis

| Função                   | Descrição  |
|--------------------------|--|
| doubleval                | Retorna o valor de uma variável em ponto flutuante.                      |
| empty                    | Testa se o conteúdo de uma variável é nulo.                              |
| floatval                 | Obtém o valor de uma variável em ponto flutuante.                        |
| gettype                  | Obtém o tipo da variável.  |
| get_defined_vars         | Retorna um array com todas as variáveis definidas.                       |
| get_resource_type        | Retorna o tipo do recurso.   |
| import_request_variables | Importa as variáveis GET/POST/Cookie em um escopo global.                |
| intval                   | Retorna o valor inteiro de uma variável.                                 |
| is_array                 | Testa se uma variável é um array.  |
| is_bool                  | Testa se uma variável é do tipo bool.                                    |
| is_callable              | Verifica se o conteúdo de uma variável pode ser chamado como uma função. |
| is_double                | Testa se uma variável é do tipo double.                                  |
| is_float                 | Testa se uma variável é do tipo ponto flutuante.                         |
| is_int                   | Testa se uma variável é do tipo inteiro.                                 |
| is_integer               | Testa se uma variável é do tipo inteiro.                                 |
| is_long                  | Testa se uma variável é do tipo inteiro.                                 |
| is_null                  | Testa se uma variável é nula.  |
| is_numeric               | Verifica se uma variável é um número ou uma string numérica.             |
| is_object                | Testa se uma variável é um objeto.                                       |
| is_real                  | Testa se uma variável é do tipo real.                                    |
| is_resource              | Testa se uma variável é um recurso.                                      |
| is_scalar                | Testa se uma variável é escalar.   |
| is_string                | Testa se uma variável é uma string.                                      |
| isset                    | Verifica se uma variável está definida.                                  |
| print_r                  | Imprime informações sobre uma variável.                                  |
| serialize                | Gera uma representação armazenável de um valor.                          |
| settype                  | Define o tipo de uma variável.   |
| strval                   | Retorna a variável como uma string.                                      |
| unserialize              | Cria um valor em PHP para uma representação armazenada.                  |
| unset                    | Exclui uma variável do programa.   |
| var_dump                 | Retorna informações sobre uma variável.                                  |
| var_export               | Retorna uma representação estruturada de uma determinada variável.       |



# Apêndice B

## Funções PHP/bancos de dados

Neste apêndice veremos uma lista de funções do PHP que podem ser utilizadas para realizar a integração com diversos SGBDs, como *MySQL*, *PostgreSQL*, *InterBase*, *Oracle*, *SQL Server* etc.

### MySQL

| Função                             | Descrição  |
|------------------------------------|--|
| <code>mysql_affected_rows</code>   | Obtém o número de linhas afetadas em uma operação.   |
| <code>mysql_change_user</code>     | Altera o usuário na conexão ativa.   |
| <code>mysql_client_encoding</code> | Retorna o nome do conjunto de caracteres utilizado pelo cliente.   |
| <code>mysql_close</code>           | Fechá uma conexão MySQL.   |
| <code>mysql_connect</code>         | Abre uma conexão com o servidor MySQL.   |
| <code>mysql_data_seek</code>       | Move o ponteiro interno de resultado.  |
| <code>mysql_db_name</code>         | Obtém dados do resultado.  |
| <code>mysql_errno</code>           | Retorna o número do erro de uma operação executada.  |
| <code>mysql_error</code>           | Retorna a mensagem de erro de uma operação executada.  |
| <code>mysql_escape_string</code>   | Adiciona caracteres de escape a uma string, para que ela possa ser usada em um comando SQL sem gerar erros na sintaxe. |
| <code>mysql_fetch_array</code>     | Coloca uma linha do resultado em um array associativo, numérico ou ambos.  |
| <code>mysql_fetch_assoc</code>     | Coloca uma linha do resultado em um array associativo.   |
| <code>mysql_fetch_field</code>     | Retorna um objeto com informações sobre um campo da consulta.  |
| <code>mysql_fetch_lengths</code>   | Retorna o tamanho de cada campo em uma consulta.   |
| <code>mysql_fetch_object</code>    | Busca o resultado como um objeto.  |
| <code>mysql_fetch_row</code>       | Obtém uma linha do resultado como um array enumerado.  |

| Função                                | Descrição (cont.)   |
|---------------------------------------|---|
| <code>mysql_field_flags</code>        | Retorna os flags associados a um campo de uma consulta.   |
| <code>mysql_field_name</code>         | Retorna o nome de determinado campo.  |
| <code>mysql_field_len</code>          | Retorna o tamanho de determinado campo.   |
| <code>mysql_field_seek</code>         | Posiciona o ponteiro para um campo específico da consulta.  |
| <code>mysql_field_table</code>        | Retorna o nome da tabela à qual o campo pertence.   |
| <code>mysql_field_type</code>         | Retorna o tipo de determinado campo.  |
| <code>mysql_free_result</code>        | Libera a memória reservada para uma consulta.   |
| <code>mysql_get_client_info</code>    | Retorna informações sobre a versão do cliente MySQL.  |
| <code>mysql_get_host_info</code>      | Retorna o tipo de conexão estabelecida com o servidor.  |
| <code>mysql_get_proto_info</code>     | Retorna informações do protocolo utilizado na conexão.  |
| <code>mysql_get_server_info</code>    | Retorna a versão do servidor MySQL.   |
| <code>mysql_info</code>               | Retorna informações detalhadas sobre a última consulta realizada.   |
| <code>mysql_insert_id</code>          | Obtém o ID gerado por uma operação INSERT executada anteriormente.  |
| <code>mysql_list_dbs</code>           | Lista os bancos de dados existentes no servidor MySQL.  |
| <code>mysql_list_fields</code>        | Lista os campos do resultado.   |
| <code>mysql_list_processes</code>     | Lista os processos do MySQL. Retorna um ponteiro para os resultados, que descrevem as threads atuais.   |
| <code>mysql_list_tables</code>        | Lista tabelas em um banco de dados MySQL.   |
| <code>mysql_num_fields</code>         | Retorna o número de campos do resultado.  |
| <code>mysql_num_rows</code>           | Retorna o número de linhas do resultado.  |
| <code>mysql_pconnect</code>           | Abre uma conexão persistente com um servidor MySQL.   |
| <code>mysql_ping</code>               | Envia um ping para verificar se a conexão com o servidor está ativa. Se não estiver, tenta reconectar automaticamente.                            |
| <code>mysql_query</code>              | Envia uma consulta MySQL.   |
| <code>mysql_real_escape_string</code> | Adiciona caracteres de escape a uma string, para que ela possa ser usada em um comando SQL. Considera o conjunto de caracteres em uso na conexão. |
| <code>mysql_result</code>             | Obtém os dados do resultado.  |
| <code>mysql_select_db</code>          | Seleciona um banco de dados MySQL.  |
| <code>mysql_stat</code>               | Retorna uma string contendo informações sobre o estado corrente do sistema.   |
| <code>mysql_tablename</code>          | Obtém o nome da tabela do campo.  |
| <code>mysql_thread_id</code>          | Retorna um valor inteiro que representa o identificador da thread atual.  |
| <code>mysql_unbuffered_query</code>   | Envia uma consulta SQL ao servidor MySQL, sem armazenar em um buffer as linhas resultantes.   |

## MySQLi

| Função  | Descrição   |
|---|---|
| <code>mysqli_affected_rows</code>             | Obtém o número de linhas afetadas em uma operação.                                      |
| <code>mysqli_autocommit</code>                | Habilita ou desabilita a confirmação automática ( <i>autocommit</i> ) das modificações. |
| <code>mysqli_bind_param</code>                | Associa uma variável a um parâmetro de um comando preparado.                            |
| <code>mysqli_bind_result</code>               | Associa uma variável a um comando preparado, para armazenamento do resultado.           |
| <code>mysqli_change_user</code>               | Altera o usuário na conexão ativa.  |
| <code>mysqli_character_set_name</code>        | Retorna o nome do conjunto de caracteres utilizado pelo cliente.                        |
| <code>mysqli_client_encoding</code>           | Apelido para <code>mysqli_character_set_name()</code> .                                 |
| <code>mysqli_close</code>                     | Fechá uma conexão MySQL.  |
| <code>mysqli_commit</code>                    | Confirma a transação corrente.  |
| <code>mysqli_connect_errno</code>             | Retorna o código de erro da última chamada de conexão.                                  |
| <code>mysqli_connect_error</code>             | Retorna a descrição do erro da última chamada de conexão.                               |
| <code>mysqli_connect</code>                   | Abre uma conexão com o servidor MySQL.  |
| <code>mysqli_data_seek</code>                 | Move o ponteiro interno de resultado.   |
| <code>mysqli_debug</code>                     | Realiza operações de depuração.   |
| <code>mysqli_disable_reads_from_master</code> | Desabilita a leitura do servidor Master.  |
| <code>mysqli_disable_rpl_parse</code>         | Desabilita a análise (parse) RPL.   |
| <code>mysqli_dump_debug_info</code>           | Grava informações de depuração no registro (log) do MySQL.                              |
| <code>mysqli_embedded_connect</code>          | Abre uma conexão com um servidor MySQL embutido.  |
| <code>mysqli_enable_reads_from_master</code>  | Habilita a leitura a partir do servidor Master.   |
| <code>mysqli_enable_rpl_parse</code>          | Habilita a análise (parse) RPL.   |
| <code>mysqli_errno</code>                     | Retorna o número do erro de uma operação executada.                                     |
| <code>mysqli_error</code>                     | Retorna a mensagem de erro de uma operação executada.                                   |
| <code>mysqli_escape_string</code>             | Apelido para <code>mysqli_real_escape_string()</code> .                                 |
| <code>mysqli_execute</code>                   | Executa uma consulta previamente preparada.   |
| <code>mysqli_fetch_array</code>               | Coloca uma linha do resultado em um array associativo, numérico ou ambos.               |
| <code>mysqli_fetch_assoc</code>               | Coloca uma linha do resultado em um array associativo.                                  |
| <code>mysqli_fetch_field_direct</code>        | Obtém os metadados para um determinado campo.   |
| <code>mysqli_fetch_field</code>               | Retorna um objeto com informações sobre um campo da consulta.                           |
| <code>mysqli_fetch_fields</code>              | Retorna um array de objetos com informações sobre os campos da consulta.                |
| <code>mysqli_fetch_lengths</code>             | Retorna o tamanho de cada campo em uma consulta.  |
| <code>mysqli_fetch_object</code>              | Busca o resultado como um objeto.   |
| <code>mysqli_fetch_row</code>                 | Obtém uma linha do resultado como um array enumerado.                                   |
| <code>mysqli_fetch</code>                     | Armazena nas variáveis associadas (bind) os resultados de um comando preparado.         |
| <code>mysqli_field_count</code>               | Retorna o número de colunas da consulta mais recente.                                   |
| <code>mysqli_field_seek</code>                | Posiciona o ponteiro para um campo específico da consulta.                              |
| <code>mysqli_field_tell</code>                | Retorna o número do campo corrente em um ponteiro de resultado.                         |

| Função                                 | Descrição (cont.)   |
|--|---|
| <code>mysqli_free_result</code>        | Libera a memória reservada para uma consulta.   |
| <code>mysqli_get_client_info</code>    | Retorna informações sobre a versão do cliente MySQL.  |
| <code>mysqli_get_host_info</code>      | Retorna o tipo de conexão estabelecida com o servidor.  |
| <code>mysqli_get_metadata</code>       | Apelido para <code>mysqli_stmt_result_metadata()</code> .   |
| <code>mysqli_get_proto_info</code>     | Retorna informações do protocolo utilizado na conexão.  |
| <code>mysqli_get_server_info</code>    | Retorna a versão do servidor MySQL.   |
| <code>mysqli_get_server_version</code> | Retorna a versão de um servidor MySQL como um inteiro.  |
| <code>mysqli_info</code>               | Retorna informações detalhadas sobre a última consulta realizada.   |
| <code>mysqli_init</code>               | Inicializa a MySQLi e retorna um recurso para uso na função <code>mysqli_real_connect</code> .  |
| <code>mysqli_insert_id</code>          | Obtém o ID gerado por uma operação INSERT executada anteriormente.  |
| <code>mysqli_kill</code>               | Solicita ao servidor o encerramento de um determinado processo (thread) MySQL.  |
| <code>mysqli_master_query</code>       | Em uma configuração master/slave (mestre/escravo), força a execução da consulta no master.  |
| <code>mysqli_more_results</code>       | Verifica se há mais resultados em uma consulta múltipla (multi query).  |
| <code>mysqli_multi_query</code>        | Executa uma ou mais consultas em um banco de dados.   |
| <code>mysqli_next_result</code>        | Prepara o próximo resultado de uma consulta múltipla (multi query).   |
| <code>mysqli_num_fields</code>         | Retorna o número de campos do resultado.  |
| <code>mysqli_num_rows</code>           | Retorna o número de linhas do resultado.  |
| <code>mysqli_options</code>            | Configura opções do MySQL.  |
| <code>mysqli_param_count</code>        | Retorna o número de parâmetros de um determinado comando.   |
| <code>mysqli_ping</code>               | Envia um ping para verificar se a conexão com o servidor está ativa. Se não estiver, tenta reconectar automaticamente.                            |
| <code>mysqli_prepare</code>            | Prepara um comando SQL para execução.   |
| <code>mysqli_query</code>              | Envia uma consulta MySQL.   |
| <code>mysqli_real_connect</code>       | Abre uma conexão com um servidor MySQL.   |
| <code>mysqli_real_escape_string</code> | Adiciona caracteres de escape a uma string, para que ela possa ser usada em um comando SQL. Considera o conjunto de caracteres em uso na conexão. |
| <code>mysqli_real_query</code>         | Executa uma consulta SQL.   |
| <code>mysqli_report</code>             | Habilita ou desabilita as funções internas que reportam erros.  |
| <code>mysqli_rollback</code>           | Cancela a transação corrente.   |
| <code>mysqli_select_db</code>          | Seleciona um banco de dados MySQL.  |
| <code>mysqli_send_long_data</code>     | Apelido para <code>mysqli_stmt_send_long_data()</code> .  |
| <code>mysqli_send_query</code>         | Envia uma consulta e retorna.   |
| <code>mysqli_server_end</code>         | Encerra o servidor MySQL embutido (embedded).   |
| <code>mysqli_server_init</code>        | Inicializa o servidor embutido (embedded) MySQL.  |
| <code>mysqli_set_charset</code>        | Define o conjunto de caracteres padrão do cliente.  |

| Função                                   | Descrição (cont.)   |
|--|---|
| <code>mysqli_set_opt</code>              | Apelido para <code>mysqli_options()</code> .  |
| <code>mysqli_sqlstate</code>             | Retorna o erro SQLSTATE referente à última operação do MySQL.   |
| <code>mysqli_ssl_set</code>              | Estabelece uma conexão segura com o MySQL usando SSL.   |
| <code>mysqli_stat</code>                 | Retorna uma string contendo informações sobre o estado corrente do sistema.                                       |
| <code>mysqli_stmt_affected_rows</code>   | Número de linhas alteradas, excluídas ou inseridas pelo último comando executado.                                 |
| <code>mysqli_stmt_bind_param</code>      | Associa variáveis aos parâmetros de um comando preparado.   |
| <code>mysqli_stmt_bind_result</code>     | Associa variáveis a um comando preparado para armazenamento do resultado.   |
| <code>mysqli_stmt_close</code>           | Fechá um comando.   |
| <code>mysqli_stmt_data_seek</code>       | Procura por uma determinada linha no conjunto de resultados.  |
| <code>mysqli_stmt_errno</code>           | Retorna o código de erro referente ao último comando executado.   |
| <code>mysqli_stmt_error</code>           | Retorna a descrição do erro referente ao último comando executado.  |
| <code>mysqli_stmt_execute</code>         | Executa uma consulta preparada.   |
| <code>mysqli_stmt_fetch</code>           | Busca resultados a partir de um comando preparado, armazenando-os nas variáveis associadas.                       |
| <code>mysqli_stmt_free_result</code>     | Liberá a memória alocada pelo resultado do comando especificado.  |
| <code>mysqli_stmt_init</code>            | Inicializa um comando e retorna um objeto para uso com <code>mysqli_stmt_prepare</code> .                         |
| <code>mysqli_stmt_num_rows</code>        | Retorna o número de linhas existentes no conjunto de resultados do comando.                                       |
| <code>mysqli_stmt_param_count</code>     | Retorna o número de parâmetros do comando especificado.   |
| <code>mysqli_stmt_prepare</code>         | Prepara um comando SQL para execução.   |
| <code>mysqli_stmt_reset</code>           | Reinicia um comando preparado.  |
| <code>mysqli_stmt_result_metadata</code> | Retorna os metadados do conjunto de resultados de um comando preparado.   |
| <code>mysqli_stmt_send_long_data</code>  | Envia dados em blocos.  |
| <code>mysqli_stmt_sqlstate</code>        | Retorna o erro SQLSTATE referente à última operação.  |
| <code>mysqli_store_result</code>         | Transfere o resultado da última consulta para ser usado pela função <code>mysqli_data_seek()</code> .             |
| <code>mysqli_thread_id</code>            | Retorna um valor inteiro que representa o identificador da thread atual.  |
| <code>mysqli_thread_safe</code>          | Indica se a segurança de um processo (thread) está definida ou não.   |
| <code>mysqli_use_result</code>           | Inicia o retorno de um resultado (result set) da última consulta executada por <code>mysqli_real_query()</code> . |
| <code>mysqli_warning_count</code>        | Retorna o número de avisos (warnings) gerados pela última consulta executada.                                     |

## PostgreSQL

| Função                            | Descrição  |
|-----------------------------------|--|
| <code>pg_affected_rows</code>     | Retorna o número de registros afetados por uma operação <i>INSERT</i> , <i>UPDATE</i> ou <i>DELETE</i> . Antigamente, essa função era chamada de <code>pg_affected_rows()</code> . |
| <code>pg_cancel_query</code>      | Cancela uma consulta assíncrona enviada pela função <code>pg_send_query()</code> .   |
| <code>pg_client_encoding</code>   | Retorna a codificação de caracteres usada pelo cliente.  |
| <code>pg_close</code>             | Fechá uma conexão PostgreSQL.  |
| <code>pg_connect</code>           | Abre uma conexão PostgreSQL.   |
| <code>pg_connection_busy</code>   | Verifica se uma conexão está ocupada ou não.   |
| <code>pg_connection_reset</code>  | Reinicia uma conexão. É útil para recuperação de erros.  |
| <code>pg_connection_status</code> | Retorna o estado da conexão ( <code>PGSQL_CONNECTION_OK</code> ou <code>PGSQL_CONNECTION_BAD</code> ).   |
| <code>pg_convert</code>           | Converte os valores de um array associativo para uso em um comando SQL.  |
| <code>pg_copy_from</code>         | Insere registros em uma tabela a partir de um array.   |
| <code>pg_copy_to</code>           | Copia o conteúdo de uma tabela para um array.  |
| <code>pg_dbname</code>            | Retorna o nome do banco de dados.  |
| <code>pg_delete</code>            | Exclui registros de acordo com uma determinada condição.   |
| <code>pg_end_copy</code>          | Sincroniza com o backend do PostgreSQL.  |
| <code>pg_escape_bytea</code>      | Adiciona caracteres de escape a uma string do tipo de dados <i>bytea</i> .   |
| <code>pg_escape_string</code>     | Adiciona caracteres de escape a uma string.  |
| <code>pg_fetch_all</code>         | Retorna um array contendo todas as linhas resultantes de uma consulta.   |
| <code>pg_fetch_array</code>       | Retorna uma linha do resultado como um array.  |
| <code>pg_fetch_assoc</code>       | Retorna uma linha do resultado em um array associativo.  |
| <code>pg_fetch_object</code>      | Retorna uma linha do resultado como um objeto.   |
| <code>pg_fetch_result</code>      | Retorna valores a partir de um recurso do tipo <i>result</i> , retornado pela função <code>pg_query()</code> .   |
| <code>pg_fetch_row</code>         | Retorna uma linha do resultado como um array enumerado.  |
| <code>pg_field_is_null</code>     | Testa se um campo é nulo.  |
| <code>pg_field_name</code>        | Retorna o nome do campo.   |
| <code>pg_field_num</code>         | Retorna o número do campo.   |
| <code>pg_field_prtlen</code>      | Retorna o tamanho do campo para impressão.   |
| <code>pg_field_size</code>        | Retorna o tamanho do campo definido no banco de dados.   |
| <code>pg_fieldtype</code>         | Retorna o tipo de dado de um campo da consulta.  |
| <code>pg_free_result</code>       | Libera a memória alocada para uma consulta.  |

| Função           | Descrição (cont.)   |
|------------------|---|
| pg_get_notify    | Obtém o estado da conexão enviando um comando SQL <b>NOTIFY</b> .   |
| pg_get_pid       | Retorna o identificador do processo (PID) do servidor de banco de dados.  |
| pg_get_result    | Obtém os resultados de uma consulta assíncrona, executada pela função <b>pg_send_query()</b> .                                |
| pg_host          | Retorna o nome do servidor associado à conexão.   |
| pg_insert        | Inseres os valores de um array associativo em uma tabela.   |
| pg_last_error    | Obtém a última mensagem de erro para a conexão especificada.  |
| pg_last_notice   | Retorna a última mensagem gerada pelo servidor PostgreSQL.  |
| pg_last_oid      | Retorna o último identificador do objeto (OID).   |
| pg_host          | Retorna o nome do host associado com a conexão.   |
| pg_lo_close      | Fechá um objeto extenso (large object).   |
| pg_lo_create     | Cria um objeto extenso.   |
| pg_lo_export     | Exporta um objeto extenso para um arquivo.  |
| pg_lo_import     | Importa um objeto extenso de um arquivo.  |
| pg_lo_open       | Abre um objeto extenso.   |
| pg_lo_read       | Lê um objeto extenso.   |
| pg_lo_read_all   | Lê um objeto extenso inteiro.   |
| pg_lo_seek       | Busca por uma posição em um objeto extenso.   |
| pg_lo_tell       | Retorna a posição corrente em um objeto extenso.  |
| pg_lo_unlink     | Exclui um objeto extenso.   |
| pg_lo_write      | Escreve em um objeto extenso.   |
| pg_meta_data     | Obtém os metadados para uma determinada tabela.   |
| pg_num_fields    | Retorna o número de campos de uma consulta.   |
| pg_num_rows      | Retorna o número de linhas de uma consulta.   |
| pg_options       | Obtém opções associadas com uma conexão.  |
| pg_pconnect      | Abre uma conexão persistente com um servidor PostgreSQL.  |
| pg_ping          | Envia um <b>ping</b> para verificar se a conexão com o servidor está ativa. Se não estiver, tenta reconectar automaticamente. |
| pg_port          | Retorna o número da porta associado com a conexão.  |
| pg_put_line      | Envia string terminada em <b>NULL</b> para o backend do PostgreSQL.   |
| pg_query         | Executa uma consulta. Antigamente, essa função era chamada de <b>pg_exec()</b> .  |
| pg_result_error  | Obtém a mensagem de erro associada ao resultado.  |
| pg_result_seek   | Configura o offset interno das linhas do resultado.   |
| pg_result_status | Obtém o estado do resultado de uma consulta.  |
| pg_select        | Seleciona registros de acordo com uma determinada condição.   |
| pg_send_query    | Envia uma consulta assíncrona ao servidor. Os resultados são obtidos posteriormente pela função <b>pg_get_result()</b> .      |

| Função                              | Descrição (cont.)   |
|-------------------------------------|---|
| <code>pg_set_client_encoding</code> | Define a codificação do cliente.  |
| <code>pg_trace</code>               | Habilita o tracing em uma conexão PostgreSQL.                                   |
| <code>pg_tty</code>                 | Retorna o nome do terminal associado a uma conexão.                             |
| <code>pg_unescape_bytea</code>      | Remove caracteres de escape de uma string do tipo de dados <code>bytea</code> . |
| <code>pg_untrace</code>             | Desabilita o tracing de uma conexão PostgreSQL.                                 |
| <code>pg_update</code>              | Atualiza registros em uma tabela de acordo com uma determinada condição.        |

## SQLite

| Função                               | Descrição  |
|--------------------------------------|--|
| <code>sqlite_array_query</code>      | Executa uma consulta em um banco de dados, retornando os resultados em um array. É semelhante ao uso da função <code>sqlite_query()</code> seguida por <code>sqlite_fetch_array()</code> .   |
| <code>sqlite_busy_timeout</code>     | Define o tempo máximo de espera do SQLite pela disponibilidade de um handle de banco de dados.   |
| <code>sqlite_changes</code>          | Retorna o número de linhas que foram alteradas pelo último comando SQL executado.  |
| <code>sqlite_close</code>            | Fechá um banco de dados SQLite.  |
| <code>sqlite_column</code>           | Obtém o valor de uma coluna da linha corrente do resultado.  |
| <code>sqlite_create_aggregate</code> | Registra uma função definida pelo usuário (UDF) para o uso em comandos SQL.  |
| <code>sqlite_create_function</code>  | Registra uma função do PHP para a utilização em comandos SQL.  |
| <code>sqlite_current</code>          | Armazena a linha corrente do resultado em um array. Não avança o ponteiro do resultado, ao contrário da função <code>sqlite_fetch_array()</code> .   |
| <code>sqlite_error_string</code>     | Retorna a mensagem correspondente a um determinado código de erro.   |
| <code>sqlite_escape_string</code>    | Adiciona caracteres de escape a uma string, para que ela possa ser usada como parâmetro em uma consulta.   |
| <code>sqlite_exec</code>             | Executa uma consulta (sem resultados) no banco de dados especificado.  |
| <code>sqlite_factory</code>          | Abre um banco de dados SQLite e retorna um objeto <code>SQLiteDatabase</code> .  |
| <code>sqlite_fetch_all</code>        | Busca todas as linhas de um conjunto de resultados ( <code>result set</code> ) como um array de arrays.  |
| <code>sqlite_fetch_array</code>      | Obtém a próxima linha do resultado, armazenando-a em um array. Os tipos de resultado possíveis são <code>SQLITE_NUM</code> (chaves numéricas), <code>SQLITE_ASSOC</code> (chaves associativas) e <code>SQLITE_BOTH</code> (ambos). |

| Função                                 | Descrição (cont.)  |
|--|--|
| <code>sqlite_fetch_column_types</code> | Retorna um array com os tipos de colunas de uma determinada tabela.  |
| <code>sqlite_fetch_object</code>       | Busca a próxima linha de um conjunto de resultados como um objeto.   |
| <code>sqlite_fetch_single</code>       | Retorna o valor da primeira coluna de um resultado.  |
| <code>sqlite_fetch_string</code>       | Apelido para a função <code>sqlite_fetch_single()</code> .   |
| <code>sqlite_field_name</code>         | Retorna o nome de um determinado campo do resultado.   |
| <code>sqlite_has_more</code>           | Verifica se ainda existem linhas disponíveis no resultado.   |
| <code>sqlite_has_prev</code>           | Indica se há uma linha anterior à atual.   |
| <code>sqlite_key</code>                | Retorna o índice da linha corrente.  |
| <code>sqlite_last_error</code>         | Retorna o código do último erro ocorrido no banco de dados.  |
| <code>sqlite_last_insert_rowid</code>  | Retorna o identificador da linha (rowid) do último registro incluído.  |
| <code>sqlite_libencoding</code>        | Retorna a codificação utilizada atualmente pela biblioteca SQLite.   |
| <code>sqlite_libversion</code>         | Retorna a versão corrente da biblioteca SQLite.  |
| <code>sqlite_next</code>               | Avança o ponteiro do resultado para a próxima linha.   |
| <code>sqlite_num_fields</code>         | Retorna o número de campos existentes em um resultado.   |
| <code>sqlite_num_rows</code>           | Retorna o número de linhas resultantes de uma consulta.  |
| <code>sqlite_open</code>               | Abre um banco de dados SQLite. Se o banco de dados não existir, ele será criado.                                       |
| <code>sqlite_popen</code>              | Abre um handle persistente para um banco de dados SQLite. Se o banco de dados não existir, ele será criado.            |
| <code>sqlite_prev</code>               | Procura pela linha anterior à atual no conjunto de resultados.   |
| <code>sqlite_query</code>              | Executa uma consulta no banco de dados especificado e retorna um ponteiro para o resultado.                            |
| <code>sqlite_rewind</code>             | Posiciona o ponteiro de um resultado na primeira linha.  |
| <code>sqlite_seek</code>               | Posiciona o ponteiro de um resultado na linha especificada. As linhas iniciam em 0.                                    |
| <code>sqlite_single_query</code>       | Executa uma consulta e retorna um array (para uma coluna simples) ou somente o valor da primeira linha.                |
| <code>sqlite_udf_decode_binary</code>  | Decodifica dados binários passados para uma função definida pelo usuário (UDF).  |
| <code>sqlite_udf_encode_binary</code>  | Aplica uma codificação binária aos dados fornecidos, para que eles possam ser retomados de forma segura nas consultas. |
| <code>sqlite_unbuffered_query</code>   | Executa uma consulta em um banco de dados, sem armazenar em um buffer as linhas resultantes.                           |
| <code>sqlite_valid</code>              | Indica se há mais linhas disponíveis.  |

## InterBase/Firebird

| Função                   | Descrição  |
|--------------------------|--|
| ibase_add_user           | Adiciona um usuário ao banco de dados de segurança (somente para IB6 ou posterior).              |
| ibase_affected_rows      | Retorna o número de linhas afetadas pela última operação executada.                              |
| ibase_backup             | Inicia uma tarefa de cópia (backup) no gerenciador de serviços e retorna imediatamente.          |
| ibase_blob_add           | Adiciona dados em um objeto do tipo BLOB.  |
| ibase_blob_cancel        | Cancela a criação de um objeto do tipo BLOB.   |
| ibase_blob_close         | Fechá um objeto do tipo BLOB.  |
| ibase_blob_create        | Cria um objeto do tipo BLOB para adicionar dados.  |
| ibase_blob_echo          | Exibe no navegador o conteúdo de um objeto BLOB.   |
| ibase_blob_get           | Retorna o tamanho, em bytes, de um objeto BLOB aberto.   |
| ibase_blob_import        | Cria um objeto BLOB, copia um arquivo dentro dele e o fecha.                                     |
| ibase_blob_info          | Retorna o tamanho de um objeto BLOB e outras informações úteis.                                  |
| ibase_blob_open          | Abre um objeto BLOB para retornar partes de dados.   |
| ibase_close              | Fecha a conexão com um banco de dados InterBase.   |
| ibase_commit             | Confirma uma transação, tornando suas alterações permanentes no banco de dados.                  |
| ibase_connect            | Abre uma conexão com um banco de dados InterBase.  |
| ibase_db_info            | Obtém as estatísticas de um banco de dados.  |
| ibase_delete_user        | Exclui um usuário do banco de dados de segurança. Funciona somente para IB6 ou posterior.        |
| ibase_drop_db            | Exclui um banco de dados.  |
| ibase_errcode            | Retorna o código de erro correspondente à última chamada de função do InterBase.                 |
| ibaseerrmsg              | Retorna mensagens de erro.   |
| ibase_execute            | Executa uma consulta previamente preparada.  |
| ibase_fetch_assoc        | Obtém uma linha do resultado, armazenando-a em um array associativo.                             |
| ibase_fetch_object       | Obtém um objeto de um banco de dados InterBase.  |
| ibase_fetch_row          | Busca uma linha de um banco de dados InterBase.  |
| ibase_field_info         | Obtém informações sobre um determinado campo.  |
| ibase_free_event_handler | Cancela um handler de eventos registrado anteriormente.  |
| ibase_free_query         | Libera a memória alocada para uma consulta preparada.  |
| ibase_free_result        | Libera o conjunto de linhas de um resultado (result set).  |
| ibase_gen_id             | Incrementa o gerador especificado e retorna seu novo valor.                                      |
| ibase_maintain_db        | Executa um comando de manutenção no servidor de bancos de dados.                                 |
| ibase_modify_user        | Altera dados de um usuário no banco de dados de segurança. Funciona somente no IB6 ou posterior. |
| ibase_name_result        | Associa um nome ao resultado de uma consulta.  |

| Função                  | Descrição (cont.)   |
|-------------------------|---|
| ibase_num_fields        | Obtém o número de campos existentes em um result set.   |
| ibase_num_params        | Retorna o número de parâmetros existentes em um consulta preparada.   |
| ibase_param_info        | Retorna informações sobre um parâmetro de uma consulta preparada.   |
| ibase_pconnect          | Cria uma conexão persistente com um banco de dados InterBase.   |
| ibase_prepare           | Prepara uma consulta para posterior ligação (binding) de parâmetros e execução.                               |
| ibase_query             | Executa uma consulta em um banco de dados InterBase.  |
| ibase_restore           | Inicia uma tarefa de restauração no gerenciador de serviços e retorna imediatamente.                          |
| ibase_rollback_ret      | Desfaz as alterações feitas por uma transação, sem encerrá-la.  |
| ibase_rollback          | Desfaz as alterações feitas por uma transação e a encerra.  |
| ibase_server_info       | Solicita informações sobre um servidor de banco de dados.   |
| ibase_service_attach    | Conecta ao gerenciador de serviços.   |
| ibase_service_detach    | Desconecta do gerenciador de serviços.  |
| ibase_set_event_handler | Registra uma função callback ao ocorrer um determinado evento.  |
| ibase_timefmt           | Define o formato das colunas do tipo <i>timestamp</i> , <i>date</i> e <i>time</i> retornadas pelas consultas. |
| ibase_trans             | Inicia uma transação.   |
| ibase_wait_event        | Suspende a execução do programa até que seja gerado um determinado evento pelo banco de dados.                |

**dbx**

|                   |  |
|-------------------|--|
| dbx_close         | Fechá uma conexão (aberta com a função <code>dbx_connect</code> ) com um banco de dados.                             |
| dbx_compare       | Compara duas linhas para fins de ordenação.  |
| dbx_connect       | Abre uma conexão com um banco de dados.  |
| dbx_error         | Retorna a mensagem de erro referente à última função executada pelo módulo do banco de dados.                        |
| dbx_escape_string | Adiciona caracteres de escape a um texto, para que ele possa ser usado em um comando SQL sem gerar erros na sintaxe. |
| dbx_query         | Envia um comando SQL ao banco de dados e obtém os resultados.  |
| dbx_sort          | Ordena os registros resultantes de uma consulta utilizando uma função de ordenação personalizada.                    |

## Microsoft SQL Server

| Função                                  | Descrição   |
|---|---|
| <code>mssql_bind</code>                 | Adiciona um parâmetro a um procedimento armazenado local ou remoto.   |
| <code>mssql_close</code>                | Fechá uma conexão estabelecida com o MS SQL Server.   |
| <code>mssql_connect</code>              | Abre uma conexão com um servidor MS SQL Server.   |
| <code>mssql_data_seek</code>            | Moving o ponteiro interno das linhas.   |
| <code>mssql_execute</code>              | Executa um procedimento armazenado em um servidor MS SQL Server.  |
| <code>mssql_fetch_array</code>          | Obtém uma linha do resultado como um array.   |
| <code>mssql_fetch_assoc</code>          | Retorna um array associativo contendo a linha corrente do conjunto de resultados (result set) especificado. |
| <code>mssql_fetch_batch</code>          | Retorna o próximo conjunto de registros armazenados em memória.   |
| <code>mssql_fetch_field</code>          | Obtém informações sobre um campo.   |
| <code>mssql_fetch_object</code>         | Obtém uma linha do resultado como um objeto.  |
| <code>mssql_fetch_row</code>            | Obtém uma linha do resultado como um array com índices numéricos.   |
| <code>mssql_field_length</code>         | Obtém o tamanho de um campo.  |
| <code>mssql_field_name</code>           | Obtém o nome de um campo.   |
| <code>mssql_field_seek</code>           | Faz uma busca no campo pelo offset especificado.  |
| <code>mssql_field_type</code>           | Obtém o tipo de um campo.   |
| <code>mssql_free_result</code>          | Libera a memória alocada para armazenar um resultado.   |
| <code>mssql_free_statement</code>       | Libera a memória alocada para execução de um comando.   |
| <code>mssql_get_last_message</code>     | Retorna a última mensagem gerada pelo servidor.   |
| <code>mssql_guid_string</code>          | Converte um GUID binário de 16 bytes para string.   |
| <code>mssql_init</code>                 | Inicializa um procedimento armazenado local ou remoto.  |
| <code>mssql_min_error_severity</code>   | Define a menor severidade para um erro.   |
| <code>mssql_min_message_severity</code> | Define a menor severidade para uma mensagem.  |
| <code>mssql_next_result</code>          | Moving o ponteiro interno do resultado para a próxima linha.  |
| <code>mssql_num_fields</code>           | Obtém o número de campos do resultado.  |
| <code>mssql_num_rows</code>             | Obtém o número de linhas do resultado.  |
| <code>mssql_pconnect</code>             | Abre uma conexão persistente com o servidor MS SQL.   |
| <code>mssql_query</code>                | Envia uma consulta ao servidor MS SQL Server.   |
| <code>mssql_result</code>               | Obtém os dados de um resultado.   |
| <code>mssql_rows_affected</code>        | Retorna o número de registros afetados por uma operação.  |
| <code>mssql_select_db</code>            | Seleciona um banco de dados MS SQL Server.  |

## Oracle

| Função         | Descrição  |
|----------------|--|
| ora_bind       | Associa uma variável PHP a um parâmetro do Oracle.                             |
| ora_close      | Fecha um cursor.   |
| ora_columnname | Obtém o nome de uma coluna do resultado.                                       |
| ora_columnsize | Obtém o tamanho de uma coluna do resultado.                                    |
| ora_columntype | Obtém o tipo de uma coluna do resultado.                                       |
| ora_commit     | Confirma uma transação.  |
| ora_commitoff  | Desabilita a confirmação automática (autocommit).                              |
| ora_committon  | Habilita a confirmação automática (autocommit).                                |
| ora_do         | Combinação das funções ora_parse, ora_exec e ora_fetch.                        |
| ora_error      | Analisa e executa um comando, e logo após obtém a primeira linha do resultado. |
| ora_errorcode  | Retorna uma mensagem de erro do Oracle.  |
| ora_exec       | Retorna um código de erro do Oracle.   |
| ora_fetch_Into | Executa um comando analisado em um cursor.                                     |
| ora_fetch      | Obtém uma linha do resultado, armazenando-a no array especificado.             |
| ora_getcolumn  | Busca por uma linha a partir de um determinado cursor.                         |
| ora_logoff     | Obtém os dados de uma determinada coluna.                                      |
| ora_logon      | Fecha uma conexão com o Oracle.  |
| ora_numcols    | Abre uma conexão com o Oracle.   |
| ora_numrows    | Retorna o número de colunas resultantes.                                       |
| ora_open       | Retorna o número de linhas resultantes.  |
| ora_parse      | Abre um cursor.  |
| ora_pllogin    | Analisa um comando SQL, associando-o a um determinado cursor.                  |
| ora_rollback   | Abre uma conexão persistente com o Oracle.                                     |
|                | Desfaz uma transação.  |

## OCI8

| Função            | Descrição   |
|-------------------|---|
| ocibindbyname     | Associa uma variável PHP a um placeholder do Oracle.                      |
| ocicancel         | Cancela a leitura de dados a partir de um cursor.                         |
| ocicloselob       | Fecha o descritor do objeto.  |
| ocicollappend     | Adiciona um objeto à coleção.   |
| ocicollassign     | Atribui uma coleção a partir de outra já existente.                       |
| ocicollassignelem | Atribui um valor do elemento à coleção, utilizando o índice especificado. |

| Função             | Descrição (cont.)  |
|--------------------|--|
| ocicollgetelem     | Retorna o valor da coleção em um determinado índice.   |
| ocicollmax         | Retorna o valor máximo de uma coleção.   |
| ocicollsize        | Retorna o tamanho de uma coleção.  |
| ocicoltrim         | Remove um determinado número de elementos do final de uma coleção.   |
| ocicolumnisnull    | Testa se uma coluna do resultado é nula (NULL).  |
| ocicolumnname      | Retorna o nome de uma coluna.  |
| ocicolumnprecision | Retorna a precisão de uma coluna.  |
| ocicolumnscale     | Retorna a escala de uma coluna.  |
| ocicolumnsize      | Retorna o tamanho de uma coluna do resultado.  |
| ocicolumntype      | Retorna o tipo de dados de uma coluna.   |
| ocicolumntypewr    | Retorna o tipo de uma determinada coluna como um número que representa o menor nível de mapeamento de colunas no Oracle. |
| ocicommit          | Confirma as transações atuais.   |
| ocidefinebyname    | Obtém uma coluna do resultado, armazenando-a em uma variável do PHP.   |
| ocierror           | Retorna um array associativo contendo informações sobre o último erro ocorrido.  |
| ociexecut e        | Executa um comando.  |
| ocifetch           | Obtém a próxima linha no buffer de resultados.   |
| ocifetchinto       | Obtém a próxima linha do resultado como um array.  |
| ocifetchstatement  | Obtém todas as linhas do resultado como um array.  |
| ocifreecollection  | Exclui um objeto de uma coleção.   |
| ocifreecursor      | Libera todos os recursos associados a um cursor.   |
| ocifreedesc        | Exclui o descritor de um objeto extenso.   |
| ocifreestatement   | Libera todos os recursos associados a um comando.  |
| ociinternaldebug   | Habilita ou desabilita a saída de depuração interna.   |
| ociloadlob         | Carrega um objeto extenso.   |
| ociologoff         | Encerra a conexão com um servidor Oracle.  |
| ociologon          | Estabelece uma conexão com um servidor Oracle.   |
| ocinewcollection   | Inicializa uma nova coleção.   |
| ocinewcursor       | Retorna um novo cursor.  |
| ocinewdescriptor   | Inicializa um novo descritor de objeto ou de arquivo.  |
| ocinlogon          | Estabelece uma nova conexão com um servidor Oracle.  |
| ocinumcols         | Retorna o número colunas resultantes de um comando.  |
| ociparse           | Analisa uma consulta e retorna um comando Oracle.  |
| ocipllogon         | Estabelece uma conexão persistente com um servidor Oracle.   |
| ociresult          | Retorna valores das colunas das linhas obtidas.  |
| ocirollback        | Desfaz as transações atuais.   |

| Função               | Descrição (cont.)  |
|----------------------|--|
| ocirowcount          | Obtém o número de linhas afetadas por uma operação.                              |
| ocisavelob           | Salva um objeto extenso.   |
| ocisavelobfile       | Salva um arquivo do objeto extenso.  |
| ociserverversion     | Retorna uma string contendo informações sobre a versão do servidor.              |
| ociprefetch          | Define o número de linhas a serem obtidas no processo de pré-busca (prefetched). |
| ocistatementtype     | Retorna o tipo de um comando OCI.  |
| ociwritelobtofile    | Salva o objeto extenso no arquivo.   |
| ociwritetemporarylob | Escreve em um objeto (BLOB) temporário.  |

## ODBC

| Função                | Descrição   |
|-----------------------|---|
| odbc_autocommit       | Habilita o autocommit (confirmação automática das operações realizadas sobre o banco de dados).                               |
| odbc_binmode          | Define como serão manipuladas as colunas com dados binários.  |
| odbc_close_all        | Fecha todas as conexões ODBC.   |
| odbc_close            | Fecha uma conexão ODBC.   |
| odbc_columnprivileges | Retorna um identificador de resultado que pode ser usado para buscar uma lista de colunas e os privilégios associados a elas. |
| odbc_columns          | Retorna um identificador de resultado contendo a lista dos nomes das colunas existentes nas tabelas especificadas.            |
| odbc_commit           | Confirma uma transação ODBC.  |
| odbc_connect          | Estabelece uma conexão com uma fonte de dados (datasource).   |
| odbc_cursor           | Obtém o nome de um cursor.  |
| odbc_data_source      | Retorna informações sobre a conexão corrente.   |
| odbc_do               | Apelido para a função <code>odbc_exec</code> .  |
| odbc_error            | Obtém o código do último erro ocorrido.   |
| odbc_errormsg         | Obtém a mensagem referente ao último erro ocorrido.   |
| odbc_exec             | Prepara e executa um comando SQL.   |
| odbc_execute          | Executa um comando preparado.   |
| odbc_fetch_array      | Obtém uma linha do resultado como um array associativo.   |
| odbc_fetch_into       | Obtém uma linha do resultado como um array.   |
| odbc_fetch_object     | Obtém uma linha do resultado como um objeto.  |
| odbc_fetch_row        | Obtém uma linha do resultado.   |
| odbc_field_len        | Obtém o tamanho (precisão) de um campo.   |
| odbc_field_name       | Obtém o nome de uma coluna.   |

| Função                             | Descrição (cont.)   |
|------------------------------------|---|
| <code>odbc_field_num</code>        | Retorna o número de uma coluna.   |
| <code>odbc_field_precision</code>  | Apelido para a função <code>odbc_field_len</code> .   |
| <code>odbc_field_scale</code>      | Obtém a escala de um campo.   |
| <code>odbc_field_type</code>       | Obtém a tipo de dado de um campo.   |
| <code>odbc_foreignkeys</code>      | Retorna a lista de chaves estrangeiras na tabela especificada ou em outras tabelas que referenciam sua chave primária.  |
| <code>odbc_free_result</code>      | Libera os recursos associados ao resultado.   |
| <code>odbc_gettypeinfo</code>      | Retorna um identificador de resultado contendo informações sobre os tipos de dados suportados pela fonte de dados (data source).  |
| <code>odbc_longreadlen</code>      | Manipulador de colunas do tipo LONG.  |
| <code>odbc_next_result</code>      | Verifica se estão disponíveis múltiplos resultados.   |
| <code>odbc_num_fields</code>       | Retorna o número de colunas existentes em um resultado.   |
| <code>odbc_num_rows</code>         | Retorna o número de linhas existentes em um resultado.  |
| <code>odbc_pconnect</code>         | Abre uma conexão persistente com o banco de dados.  |
| <code>odbc_prepare</code>          | Prepara um comando para execução.   |
| <code>odbc_primarykeys</code>      | Retorna um identificador de resultado, que pode ser usado para buscar os nomes das colunas que compõem a chave primária da tabela.  |
| <code>odbc_procedurecolumns</code> | Retorna informações sobre parâmetros de um procedimento.  |
| <code>odbc_procedures</code>       | Retorna um identificador de resultado contendo a lista de procedimentos armazenados em uma determinada fonte de dados.  |
| <code>odbc_result_all</code>       | Exibe o resultado como uma tabela HTML.   |
| <code>odbc_result</code>           | Obtém os dados do resultado.  |
| <code>odbc_rollback</code>         | Desfaz uma transação.   |
| <code>odbc_setoption</code>        | Ajusta as configurações ODBC. Se algum erro ocorrer, retorna FALSE.   |
| <code>odbc_specialcolumns</code>   | Retorna o melhor conjunto de colunas que identifica de forma única uma linha da tabela, ou as colunas que são atualizadas automaticamente quando algum valor na linha é modificado por uma transação. |
| <code>odbc_statistics</code>       | Retorna estatísticas sobre uma tabela.  |
| <code>odbc_tableprivileges</code>  | Lista as tabelas e os privilégios associados a cada uma delas.  |
| <code>odbc_tables</code>           | Retorna um identificador de resultado contendo a lista dos nomes das tabelas armazenadas em uma determinada fonte de dados.   |

# Apêndice C

## Tipos de recursos do PHP

Neste apêndice veremos uma lista de funções que criam e finalizam os diversos recursos oferecidos pelo PHP. Se você quiser testar se uma variável é considerada um recurso, basta usar o comando `is_resource` do PHP, que possui a seguinte sintaxe:

```
is_resource (var)
```

Esse comando retorna verdadeiro se a variável definida no parâmetro `var` é um recurso. Para obter o tipo de recurso de uma variável, utilizamos a função `get_resource_type`, que possui a seguinte sintaxe:

```
get_resource_type (handle)
```

Essa função retorna uma string contendo o tipo de recurso que foi passado. Será gerado um erro caso o recurso passado for desconhecido.

A tabela a seguir mostra os recursos que o PHP oferece, além de apresentar as funções que criam e finalizam esses recursos.

| Tipo de recurso | Criado por                 | Finalizado por             | Definição                          |
|-----------------|----------------------------|----------------------------|------------------------------------|
| aspell          | <code>aspell_new()</code>  | -                          | Dicionário aspell                  |
| bzip2           | <code>bzopen()</code>      | <code>bzclose()</code>     | Arquivo bzip2                      |
| COM             | <code>com_load()</code>    | -                          | Objeto COM                         |
| cpdf            | <code>cpdf_open()</code>   | <code>cpdf_close()</code>  | Documento PDF com biblioteca CPDF  |
| cpdf outline    | -                          | -                          | -                                  |
| curl            | <code>curl_init()</code>   | <code>curl_close()</code>  | Sessão CURL                        |
| dbm             | <code>dbmopen()</code>     | <code>dbmclose()</code>    | Link para um banco de dados DBM    |
| dba             | <code>dba_popen()</code>   | <code>dba_close()</code>   | Link para uma base DBA             |
| dba persistent  | <code>dba_open()</code>    | -                          | Link persistente para uma base DBA |
| dbase           | <code>dbase_open()</code>  | <code>dbase_close()</code> | Link para uma base Dbase           |
| dbx (link)      | <code>dbx_connect()</code> | <code>dbx_close()</code>   | Conexão com dbx                    |
| dbx (resultado) | <code>dbx_query()</code>   | -                          | Resultado de uma consulta com dbx  |
| domxml document | -                          | -                          | -                                  |
| domxml node     | -                          | -                          | -                                  |

| Tipo de recurso           | Criado por   | Finalizado por                    | Definição (cont.)                                 |
|---------------------------|--|-----------------------------------|---|
| domxml attribute          | -  | -                                 | -   |
| xpath context             | -  | -                                 | -   |
| xpath object              | -  | -                                 | -   |
| fbsql (link)              | fbsql_change_user()<br>fbsql_connect()   | fbsql_close()                     | Conexão FrontBase                                 |
| fbsql (plink)             | fbsql_change_user()<br>fbsql_pconnect()  | fbsql_free_result()               | Conexão persistente FrontBase                     |
| fbsql (resultado)         | fbsql_db_query()<br>fbsql_list_dbs()<br>fbsql_query()<br>fbsql_list_fields()<br>fbsql_list_tables()<br>fbsql_tablename()   | fbsql_free_result()               | Resultado de uma consulta FrontBase               |
| fdf                       | fdf_open()   | fdf_close()                       | Arquivo FDF                                       |
| ftp                       | ftp_connect()  | ftp_quit()                        | Corrente FTP                                      |
| gd                        | imagecreate(),<br>imagecreatefromgif(),<br>imagecreatefromjpeg(),<br>imagecreatefrompng(),<br>imagecreatefromwbmp(),<br>imagecreatefromstring(),<br>imagecreatetruecolor() | imagedestroy()                    | Imagem GD   |
| gd font                   | imageloadfont()  | -                                 | Fonte para GD                                     |
| gd PS font                | imagepsloadfont()  | imagepsfreefont()                 | Fonte PS para GD                                  |
| gd PS encoding            | -  | -                                 | -   |
| GMP integer               | gmp_init()   | -                                 | Número GMP  |
| hyperwave link            | hw_connect()   | hw_close(),<br>hw_free_document() | Link para um servidor Hyperwave                   |
| hyperwave link persistent | hw_pconnect()  | -                                 | Link persistente para um servidor Hyperwave       |
| hyperwave document        | hw_cp(),<br>hw_docbyanchor(),<br>hw_getremote(),<br>hw_getremotecchildren()  | hw_deleteobject()                 | Objeto Hyperwave                                  |
| icap                      | icap_open()  | icap_close()                      | Link para um servidor icap                        |
| imap                      | imap_open()  | imap_close()                      | Link para um servidor IMAP, POP3                  |
| imap persistent           | -  | -                                 | -   |
| imap chain persistent     | -  | -                                 | -   |
| ingres                    | ingres_connect()   | ingres_close()                    | Link persistente para uma base ingresII           |
| ingres persistent         | ingres_pconnect()  | -                                 | Link para uma base ingresII                       |
| interbase result          | ibase_query()  | ibase_free_result()               | Resultado Interbase                               |
| interbase query           | ibase_prepare()  | ibase_free_query()                | Consulta Interbase                                |
| interbase blob            | -  | -                                 | -   |
| interbase link            | ibase_connect()  | ibase_close()                     | Link para um banco de dados Interbase             |
| interbase link persistent | ibase_pconnect()   | -                                 | Link persistente para um banco de dados Interbase |
| interbase transaction     | ibase_trans()  | ibase_rollback()                  | Transação Interbase                               |
| java                      | -  | -                                 | -   |
| ldap result               | ldap_read()  | ldap_free_result()                | Busca de resultado ldap                           |

| Tipo de recurso        | Criado por  | Finalizado por                              | Definição (cont.)  |
|------------------------|---|---|--|
| ldap link              | ldap_connect(),<br>ldap_search()  | ldap_close()                                | Conexão ldap   |
| mcal                   | mcal_open(),<br>mcal_popen()  | mcal_close()                                | Link para o servidor de calendário                           |
| mssql query            | mssql_query()   | mssql_free_result(),<br>mssql_free_result() | Resultado mSQL   |
| mssql link             | mssql_connect()   | mssql_close()                               | Link para um banco de dados mSQL                             |
| mssql link persistent  | mssql_pconnect()  | -   | Link persistente para um banco de dados mSQL                 |
| mssql result           | mssql_query()   | mssql_free_result()                         | Resultado do Microsoft SQL Server                            |
| mssql link             | mssql_connect()   | mssql_close()                               | Link para um banco de dados Microsoft SQL Server             |
| mssql link persistent  | mssql_pconnect()  | -   | Link persistente para um banco de dados Microsoft SQL Server |
| mysql result           | mysql_db_query(),<br>mysql_list_dbs(),<br>mysql_list_fields(),<br>mysql_list_tables(),<br>mysql_query() | mysql_free_result()                         | Resultado do MySQL   |
| mysql link             | mysql_connect()   | mysql_close()                               | Link para um banco de dados MySQL                            |
| mysql link persistent  | mysql_pconnect()  | -   | Link persistente para um banco de dados MySQL                |
| oci8 statement         | ocinewdescriptor()  | ocifreestatement()                          | Cursor do Oracle   |
| oci8 collection        | -   | -   |  |
| oci8 connection        | ocielogon(),<br>ociplogon(),<br>ocinlogon()   | ociilogoff()                                | Link para um banco de dados Oracle                           |
| oci8 descriptor        | -   | -   |  |
| oci8 server            | -   | -   |  |
| oci8 session           | -   | -   |  |
| odbc result            | odbc_prepare()  | odbc_free_result()                          | Resultado ODBC   |
| odbc link              | odbc_connect()  | odbc_close()                                | Link para um banco de dados ODBC                             |
| odbc link persistent   | odbc_connect()  | -   | Link persistente para um banco de dados ODBC                 |
| velocis link           | -   | -   |  |
| velocis result         | -   | -   |  |
| OpenSSL key            | openssl_get_privatekey(),<br>openssl_get_publickey()  | openssl_free_key()                          | Chave OpenSSL  |
| OpenSSL X.509          | openssl_x509_read()   | openssl_x509_free()                         | Chave pública  |
| oracle cursor          | ora_open()  | ora_close()                                 | Cursor do oracle   |
| oracle link            | ora_logon()   | ora_logoff()                                | Link para um banco de dados oracle                           |
| oracle link persistent | ora_plogon()  | -   | Link persistente para um banco de dados oracle               |
| pdf image              | pdf_open_image(),<br>pdf_open_image_file(),<br>pdf_open_memory_image()                                  | pdf_close_image()                           | Imagen em um arquivo PDF                                     |
| pdf outline            | -   | -   |  |
| pdf document           | pdf_new()   | pdf_close(), pdf_delete()                   | documento PDF  |
| pgsql link             | pg_connect()  | pg_close()                                  | Link para um banco de dados PostGreSQL                       |

| Tipo de recurso             | Criado por   | Finalizado por       | Definição (cont.)   |
|-----------------------------|--|----------------------|---|
| pgsql link persistent       | pg_pconnect()  | -                    | Link persistente para um banco de dados PostGreSQL                    |
| pgsql result                | pg_query()   | pg_freeresult()      | Resultado do PostGreSQL   |
| pgsql large object          | pg_getlastoid(),<br>pg_loimport(),<br>pg_loimport()            | pg_loclose()         | Objeto extenso do PostGreSQL  |
| pgsql string                | -  | -                    | -   |
| printer                     | -  | -                    | -   |
| printer pen                 | -  | -                    | -   |
| printer font                | -  | -                    | -   |
| printer brush               | -  | -                    | -   |
| pspell                      | pspell_new(),<br>pspell_new_config(),<br>pspell_new_personal() | -                    | Dicionário pspell   |
| pspell config               | pspell_config_create()   | -                    | Configuração pspell   |
| Sablotron XSLT              | xslt_create()  | xslt_free()          | Parser XSLT   |
| shmop                       | shm_open()   | shm_close()          | -   |
| sockets file descriptor set | socket()   | close()              | Socket  |
| sockets i/o vector          | -  | -                    | -   |
| dir                         | dir()  | closedir()           | Manipulador de Diretório  |
| file                        | fopen()  | fclose()             | Manipulador de arquivo  |
| pipe                        | popen()  | pclose()             | Manipulador de processo   |
| socket                      | fsockopen()  | fclose()             | Manipulador de socket   |
| sybase-db link              | sybase_connect()   | sybase_close()       | Link para um banco de dados Sybase usando a biblioteca DB             |
| sybase-db link persistent   | sybase_pconnect()  | -                    | Link persistente para um banco de dados Sybase usando a biblioteca DB |
| sybase-db result            | sybase_query()   | sybase_free_result() | Resultado do Sybase usando a biblioteca DB                            |
| sybase-ct link              | sybase_connect()   | sybase_close()       | Link para um banco de dados Sybase usando a biblioteca CT             |
| sybase-ct link persistent   | sybase_pconnect()  | -                    | Link persistente para um banco de dados Sybase usando a biblioteca CT |
| sybase-ct result            | sybase_query()   | sybase_free_result() | Resultado do Sybase usando a biblioteca CT                            |
| sysvsem                     | sem_get()  | sem_release()        | Semáforo do System V  |
| sysvshm                     | shm_attach()   | shm_detach()         | Memória compartilhada do System V                                     |
| wddx                        | wddx_packet_start()  | wddx_packet_end()    | Pacote WDDX   |
| xml                         | xml_parser_create()  | xml_parser_free()    | Parser XML  |
| zlib                        | gzopen()   | gzclose()            | Arquivo comprimido do tipo gz   |

# Apêndice D

## Links interessantes

Acessando os links apresentados a seguir você poderá se aprofundar ainda mais no estudo da linguagem PHP e de outras tecnologias. São sites que oferecem tutoriais, scripts gratuitos, documentação detalhada e diversas outras ferramentas úteis para o desenvolvimento de sites dinâmicos e interativos.

### Site oficial do PHP

- <http://www.php.net>

Neste site você encontra toda a documentação e o código do PHP, além das novidades e as versões mais recentes para download.

### Scripts, programas e tutoriais PHP

- <http://www.phpbuilder.com/>

Apresenta grande diversidade de recursos para o PHP.

- <http://phpeditors.linuxbackup.co.uk/>

Apresenta uma lista de editores que você pode usar para editar seus programas em PHP.

- <http://phpclasses.upperdesign.com/>

Um repositório de classes em PHP.

- <http://px.sklar.com/>

Apresenta muitos exemplos de scripts e funções úteis. Possui um bom mecanismo de busca para você encontrar as informações facilmente.

- <http://www.phpwizard.net/>  
Programas e tutoriais para o PHP.
- <http://www.weberdev.com/>  
Apresenta artigos, fóruns, tutoriais, scripts etc.
- [http://www.devshed.com/Server\\_Side/PHP/](http://www.devshed.com/Server_Side/PHP/)  
Contém diversos artigos sobre PHP.

### Catálogos de links PHP

- <http://www.hotscripts.com/PHP/>  
Apresenta artigos, livros, revista, scripts, softwares etc.
- <http://php.resourceindex.com/>  
Apresenta grande diversidade de recursos PHP, incluindo documentação, scripts completos e funções. O site é todo dividido em categorias e possui uma ferramenta de busca para que você encontre facilmente as informações desejadas.

### Outros links

- <http://www.mysql.com>  
Site oficial do MySQL.
- <http://www.postgresql.org/>  
Site oficial do PostgreSQL.
- <http://www.apache.org>  
Site oficial do Apache.
- <http://www zend.com>  
Site oficial da Zend Technologies.

**Símbolos**

!= 52  
 \$ 37  
 \$\_COOKIE 181  
 \$\_GET 109  
 \$\_POST 109  
 \$\_SESSION 190  
 \$GLOBALS 38  
 % 48  
 %= 53  
 && 54  
 &= 53  
 (array) 41  
 (float) 41  
 (int) 41  
 (object) 41  
 (real) 41  
 (string) 41  
 \* 48  
 \*= 53  
 + 48  
 += 53  
 - 48  
 -= 53  
 .= 53  
 / 48  
 // 26  
 /= 53  
 < 52  
 <= 53  
 <= 52  
 <> 52  
 <?php 26  
 <br> 33  
 = 53  
 == 52  
 > 52  
 >= 52  
 >> 51  
 >>= 53  
 ? 56  
 ?> 26  
 \ 32  
 \\$ 34  
 \| 34  
 \d 138

\n 34  
 \r 34  
 \t 34  
 \xnn 34  
 ^= 53  
 \_\_FILE\_\_ 37  
 \_\_LINE\_\_ 37  
 |= 53  
 || 54

**A**

alfanuméricos 32  
 ALTER TABLE 142  
 AND 54  
 Apache 24  
 arquivos 197  
 Arrays 44  
 ASP 27  
 aspas duplas 34  
 aspas invertidas 35  
 aspas simples 32  
 atribuição 48  
 Autenticação 186  
 AUTH\_TYPE 127  
 AVG 150

**B**

Bcc 219  
 bigint 133  
 bit 133, 134  
 blob 135  
 bool 133, 134  
 boolean 134  
 box 133  
 break 72  
 BSD 129  
 bytea 133

**C**

case 63  
 Cc 219  
 char 133  
 chave primária 183  
 checkbox 105  
 chmod 203  
 cidr 133  
 circle 133

class 88  
classe 47, 87  
classes 75  
comandos condicionais 59  
comandos de repetição 59  
Comments 219  
Constantes 36  
constraints 129  
construtor 93  
Content-Transfer-Encoding 219  
Content-Type 218, 219  
CONTENT\_LENGTH 127  
CONTENT\_TYPE 127  
continue 74  
Conversão 40  
conversores 40  
cookies 180  
copy 204  
COUNT 149  
CPF 80  
create database 132  
createdb 131  
CURRVAL 157

**D**  
date 133  
datetime 134  
decimal 134  
default 65  
define 36  
describe 137  
destrutor 94  
DISTINCT 149  
do...while 67  
double 134  
DROP TABLE 143

**E**  
e-mails 213  
E\_ERROR 37  
E\_NOTICE 37  
E\_PARSE 37  
E\_WARNING 37  
echo 26  
else 60  
elseif 60

empty 55  
endif 62  
enum 135  
Errors-To 220  
Escopo 38  
expressão 50, 57  
extensão 27

**F**  
FALSE 37  
factorial 86  
fclose 200  
feof 205  
fgets 201  
file 105  
file\_exists 205  
file\_get\_contents 205  
filesize 207  
float 134  
float4 133  
float8 133  
fopen 198  
for 68  
foreach 71  
form 107, 108  
formulário 103, 112  
fread 200  
Frontpage 21  
fseek 208  
ftell 208  
FTP 23  
Funções 75  
funções recursivas 85  
fwrite 201

**G**  
GATEWAY\_INTERFACE 127  
GET 107  
getenv 125  
global 38  
GPL 129  
GROUP BY 151

**H**  
HAVING 151  
hidden 104, 116  
hospedagem 23

- HTML 27, 103  
htmlspecialchars 110  
HTTP\_ACCEPT 127  
HTTP\_USER\_AGENT 127
- I**
- if 60  
image 105  
implements 92  
includes 97  
incremento 58  
índice 44  
inet 133  
input 104  
int 133, 134  
int2 134  
int4 133  
int8 133  
interfaces 92  
interpolação 30, 41  
interval 133  
INTO 155  
IP 126
- L**
- LIKE 147  
LIMIT 154  
line 133  
Linux 24  
login 183, 194  
logout 188, 195  
longblob 135  
longtext 135  
lseg 133
- M**
- macaddr 133  
mail 214  
mail headers 219  
maiúsculas 38  
MAX 150  
maxlength 105  
mediumint 134  
mediumtext 135  
métodos 92  
MIME 219  
MIN 150
- minúsculas 38  
move\_uploaded\_file 210  
MySQL 24, 129  
mysql\_affected\_rows 164  
mysql\_connect 159  
mysql\_fetch\_array 164, 167  
mysql\_fetch\_object 164  
mysql\_fetch\_row 164, 167  
mysql\_field\_name 164  
mysql\_num\_fields 164  
mysql\_num\_rows 164, 165  
mysql\_query 162  
mysql\_result 164, 165  
mysql\_select\_db 160  
mysqli 160
- N**
- NEXTVAL 157  
NOT NULL 139  
NULL 140  
numeric 133  
numéricas 44  
numéricos 31
- O**
- objeto 88  
Objetos 47  
OOP 90  
operadores 47  
Operadores aritméticos 48  
Operadores binários 50  
Operadores de atribuição 52  
Operadores lógicos 54  
OR 54  
orientada a objetos 90
- P**
- parâmetros 77  
password 104  
path 133  
PATH\_INFO 127  
PATH\_TRANSLATED 127  
pg\_affected\_rows 164  
pg\_connect 161  
pg\_fetch\_array 164  
pg\_fetch\_object 164  
pg\_fetch\_row 164

pg\_field\_name 164  
pg\_num\_fields 164  
pg\_num\_rows 164  
pg\_query 163  
pg\_result 164  
PHP 19  
PHP\_OS 37  
PHP\_VERSION 37  
point 133  
polygon 133  
pós-incremento 48  
POST 108  
PostgreSQL 24, 129  
pré-incremento 48  
precedência 57  
Priority 220  
psql 131

**Q**

QUERY\_STRING 127

**R**

radio 105  
real 133  
Received 220  
recursivas 85  
referência 82  
REMOTE\_ADDR 126, 127  
REMOTE\_HOST 127  
REMOTE\_IDENT 127  
REMOTE\_USER 127  
Reply-To 217, 220  
REQUEST\_METHOD 127  
require 101  
reset 105  
return 77  
rmdir 211

**S**

SCRIPT\_NAME 127  
SELECT 144  
select 105  
Sender 220  
sendmail\_from 216  
sendmail\_path 216  
seqüências 156  
serial 134  
SERVER\_NAME 127  
SERVER\_PORT 127  
SERVER\_PROTOCOL 127  
SERVER\_SOFTWARE 127  
session.auto\_start 192  
session.bug\_compat\_42 193  
session.bug\_compat\_warn 193  
session.cache\_expire 193  
session.cache\_limiter 193  
session.cookie\_domain 193  
session.cookie\_lifetime 193  
session.cookie\_path 193  
session.cookie\_secure 193  
session.entropy\_file 193  
session.entropy\_length 193  
session.gc\_maxlifetime 193  
session.gc\_probability 193  
session.name 192  
session.referer\_check 193  
session.save\_handler 192  
session.save\_path 192  
session.serialize\_handler 192  
session.use\_cookies 193  
session.use\_only\_cookies 193  
session.use\_trans\_sid 193  
session\_destroy 195  
session\_register 190  
session\_start 190  
session\_unset 195  
sessões 190  
set 135  
setcookie 180  
SETVAL 157  
SGBDs 129  
show 137  
SID 190  
sizeof 74  
smallint 134  
SMTP 216  
SQL 132, 159  
SQLite 130, 172  
sqlite\_changes 175  
sqlite\_close 175  
sqlite\_fetch\_array 175  
sqlite\_num\_fields 175  
sqlite\_num\_rows 175

- sqlite\_open 175  
sqlite\_query 175  
static 95  
stored procedures 129  
strings 44  
stripslashes 111  
strlen 114  
strstr 114  
subclasses 90  
Subject 220  
submit 105  
SUM 149  
switch 63
- T**  
tempnam 211  
text 134  
textarea 105  
time 134  
timestamp 134, 135  
timestamptz 134  
timetz 134  
tinyint 134  
tmpfile 212  
transações 129  
triggers 129  
TRUE 37
- U**  
unlink 212  
unset 194  
url\_rewriter.tags 193  
urldecode 112  
urlencode 112
- V**  
varbit 133  
varchar 134  
variáveis 37  
variáveis de ambiente 125  
velocidade 27  
views 129
- W**  
while 65
- X**  
X-Content-Type-Options 220  
X-Confirm-Reading-To 220  
XOR 54
- Y**  
year 135
- Z**  
Zend Engine 90

## NOVATEC EDITORA – A CASA DO PHP



### Web Interativa com Ajax e PHP

**Autor:** Juliano Niederauer

**ISBN:** 85-7522-126-6



### PHP para quem conhece PHP

**Autor:** Juliano Niederauer

**ISBN:** 85-7522-044-6



### Segurança em PHP

**Autor:** Márcio Pessoa

**ISBN:** 85-7522-140-2

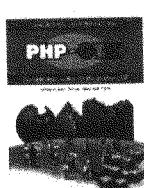


### PHP PROFISSIONAL

**Autor:** Alexandre Altair de Melo e

Mauricio G. F. Nascimento

**ISBN:** 85-7522-141-9



### PHP-GTK – 2ª Edição

**Autor:** Pablo Dall'Oglio

**ISBN:** 85-7522-110-5



### Frameworks para Desenvolvimento em PHP

**Autor:** Elton Luís Minetto

**ISBN:** 85-7522-124-2



Conheça o **site** da  
**novatec** editora

**Acesse** conteúdos adicionais exclusivos

**Compre** livros diretamente conosco

## **Downloads** de eBooks gratuitos

**Confira** os nossos lançamentos

## Sugira novos títulos

## Catálogo on-line

**[www.novatec.com.br](http://www.novatec.com.br)**

Cadastre seu e-mail e receba mais informações sobre os nossos lançamentos e promoções

# Desenvolvendo Websites com PHP

Desenvolvendo Websites com PHP apresenta técnicas de programação fundamentais para o desenvolvimento de sites dinâmicos e interativos. Você aprenderá a desenvolver sites com uma linguagem utilizada em mais de 10 milhões de sites no mundo inteiro. O livro abrange desde noções básicas de programação até a criação e manutenção de bancos de dados, mostrando como são feitas inclusões, exclusões, alterações e consultas a tabelas de uma base de dados. O autor apresenta diversos exemplos de programas para facilitar a compreensão da linguagem. Nesta obra, você irá encontrar os seguintes assuntos:

- O que é PHP e quais são suas características;
- Conceitos básicos e avançados de programação em PHP;
- Como manipular diversos tipos de dados com o PHP;
- Comandos PHP em conjunto com tags HTML;
- Utilização de includes para aumentar o dinamismo de seu site;
- Como tratar os dados enviados por um formulário HTML;
- Utilidade das variáveis de ambiente no PHP;
- Criação de banco de dados em MySQL, PostgreSQL ou SQLite;
- Comandos SQL para acessar o banco de dados via PHP;
- Como criar um sistema de username/password para seu site;
- Utilização de cookies e sessões;
- Leitura e gravação de dados em arquivos-texto;
- Como enviar e-mails pelo PHP.

## ● SOBRE O AUTOR

Juliano Niederauer ([www.niederauer.com.br](http://www.niederauer.com.br)) é graduado em Ciência da Computação e pós-graduado em Gestão Empresarial, com ênfase em Tecnologia Aplicada a Negócios. Possui larga experiência no desenvolvimento de aplicações para a Web. Já criou sites de sucesso, como o Só Matemática ([www.somatematica.com.br](http://www.somatematica.com.br)), o maior portal matemático do mundo. Atualmente desenvolve soluções para dispositivos móveis, como palmtops, além de ministrar cursos sobre PHP.

