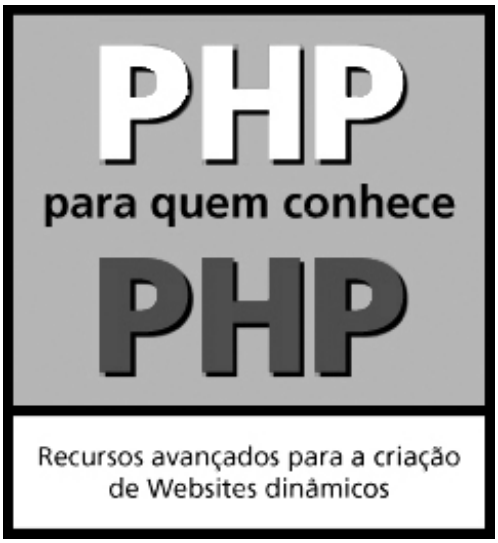

Juliano Niederauer

Novatec

Capítulo 1

Revisão de PHP

Este é um livro que aborda tópicos avançados da linguagem PHP (`www.php.net`). Porém, para você que teve pouco ou nenhum contato com essa linguagem, este capítulo será de grande importância, visto que contém uma revisão de PHP, apresentando sua sintaxe básica e ensinando a utilizar constantes, variáveis, operadores e estruturas de controle (como `if`, `switch`, `while` e `for`). Além disso, será mostrado como realizar operações sobre um banco de dados MySQL, que é um sistema bastante utilizado com o PHP. Se você é um programador mais experiente, pode passar diretamente ao capítulo 2.

Iniciando em PHP

Um programa PHP pode ser escrito em qualquer editor de texto, como, por exemplo, o Bloco de Notas (Notepad), do Windows, ou o Vi, do Linux. Já existem também diversos editores específicos para o PHP, que exibem cada elemento (variáveis, textos, palavras reservadas etc.) com cores diferentes, para melhorar a visualização.

Um trecho de código PHP deve estar entre as tags `<?php` e `?>`, para que o servidor web possa reconhecer que se trata de um código de programação e possa chamar o interpretador PHP para executá-lo. Para começar a treinar, abra o editor de texto de sua preferência e digite as linhas a seguir:

```
<?php
   // legal, estou escrevendo meu primeiro programa em PHP
   echo  "<h1 align='center'>Este é meu primeiro programa!</h1>";
?>
```

Salve esse programa como `prog1.php` e envie-o para o diretório que você está utilizando para hospedar o site. Esse programa vai gerar como resultado a frase `"Este é meu primeiro programa!"` centralizada na página, conforme mostrado na figura

1.1. Para ver o resultado, basta você acessar pelo navegador (browser) o endereço `http://<seu_endereço>/prog1.php`, onde você deve substituir *<seu_endereço>* pelo endereço do servidor que está utilizando para executar os programas PHP.

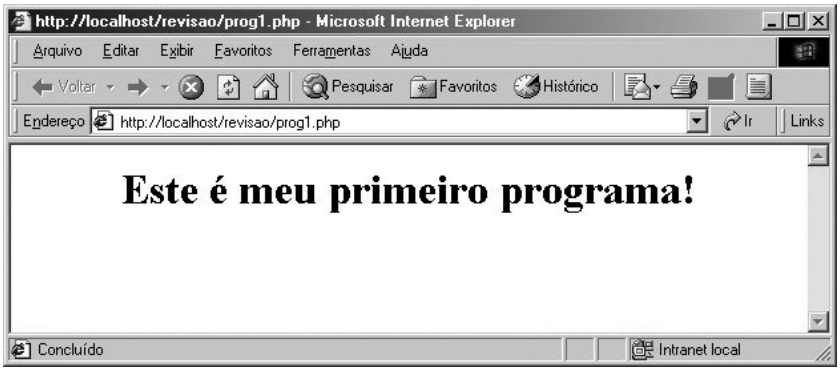

Figura 1.1 – Resultado do programa `prog1.php`.

A seguir é apresentado o significado de cada uma das linhas que você digitou no programa `prog1.php`:

Elemento	Descrição
`<?php`	Indica o início de um trecho de código PHP.
`//`	Representa uma linha de comentário. Tudo que vem após essas barras na mesma linha é ignorado pelo PHP. Os comentários são muito úteis para uma boa documentação de seu programa. As duas barras servem para transformar uma única linha em comentário, mas você pode usar `/*` para iniciar uma seqüência de comentários e, depois, finalizar os comentários com `*/`.
`echo`	Trata-se de um dos comandos mais utilizados em PHP. Serve para escrever algo na tela.
`?>`	Indica o término de um trecho de código PHP.

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

```
<h1 align='center'>Este é meu primeiro programa!</h1>
```

Note que o navegador não recebe nenhuma linha em PHP, somente o código HTML puro. Isso acontece porque o PHP roda no servidor, que processa todos os trechos de programação e retorna somente o resultado final para o navegador.

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 irá testar linha a linha em busca de códigos de programação, por isso o processo torna-se um pouco mais lento. Por essa razão, só coloque extensão `.php` nas páginas que realmente possuem códigos PHP, senão você estará desperdiçando tempo ao procurar 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 ser processadas que as páginas com HTML puro, diversos testes realizados por programadores americanos comprovaram que são processadas muito mais rápido do que aquelas que usam programação ASP. Além disso, o PHP possui um gerenciamento de memória superior ao do ASP.

Embutindo PHP na HTML

Normalmente uma página PHP não contém apenas códigos de programação PHP, mas também tags de marcação HTML. Enquanto o PHP representa a parte dinâmica da página, a HTML representa a parte estática. Ou seja, toda vez que você acessar a página, a saída HTML será a mesma, enquanto a saída gerada pelos códigos PHP pode ser diferente a cada acesso.

As tags HTML devem aparecer fora das tags `<?php` e `?>`, pois estas delimitam um trecho de programa PHP. Dentro dessas tags até podem aparecer tags HTML, mas somente se utilizarmos um comando de exibição (como o `echo`) para escrevê-las.

Você pode concatenar scripts PHP com tags HTML, podendo, dessa forma, escrever vários trechos de código PHP em uma única página. Cada script PHP existente na página deve começar com a tag `<?php` e terminar com `?>`. A maioria das linhas de programação que serão escritas entre as tags deve terminar com o caractere ; (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.

Essa combinação entre HTML e PHP é muito útil, pois utilizamos o PHP para gerar os dados de forma dinâmica, enquanto o HTML é utilizado para formatar e exibir esses dados nas páginas mostradas no navegador.

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>
```

Note a combinação existente entre as tags HTML e o código PHP. No início do programa, atribuímos a data atual à variável `$data_de_hoje`, utilizando o comando `date`. Essa variável estará disponível para uso em qualquer parte da página. Depois utilizamos HTML para escrever a frase `"Hoje é dia"` e completamos abrindo um novo trecho de PHP, escrevendo a data atual armazenada na variável `$data_de_hoje` por meio do comando `echo`.

Exibindo a página no navegador

Para o navegador mostrar qualquer coisa na tela é necessário que a página tenha pelo menos um comando de exibição (como o `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 e imprime sempre as mesmas informações na tela. Veja o seguinte exemplo:

```
<html>
<body>
<?php
    $base = 10;
    $altura = 20;
    $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. 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
    $pais = "Brasil";
    $frase = "O $pais é pentacampeão mundial de futebol";
    echo "<h2 align=center>$frase</h2>";
?>
</body>
</html>
```

Abrindo essa página em seu navegador, você obterá o resultado exibido na figura 1.2.

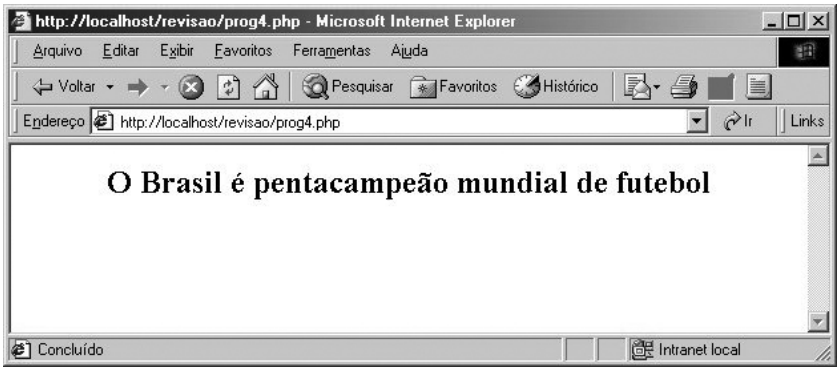

Figura 1.2 – Resultado gerado pelo programa `prog4.php`.

Com esses exemplos você descobriu duas informações fundamentais:

1. A importância do comando `echo`: para gerar dados dinâmicos é fundamental dominar esse comando ou algum outro que produza saída na tela.

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`, na variável chamada `$frase`, incluímos o valor da variável `$pais`. Ou seja, dentro da string que é atribuída a uma variável, podemos colocar outra variável e, no momento do processamento do programa, o PHP irá substituir a variável por seu valor.

Como veremos no tópico Variáveis, é importante lembrar que em PHP as variáveis começam sempre pelo símbolo de cifrão (`$`).

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`. Leve em consideração que, por padrão, o PHP diferencia letras maiúsculas de minúsculas. O nome da constante deve ser referenciado no programa exatamente do mesmo modo que você a definiu. A sintaxe do comando `define` é a seguinte:

```
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 de minúsculas quando houver uma referência a essa constante. Veja o exemplo a seguir, nomeado como `prog5.php`, que mostra como devemos usar as constantes:

prog5.php

```
<html>
<body>
<?php
    define ("meunome","Juliano Niederauer");
    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 Niederauer
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 neste 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 predefinidas. A tabela a seguir mostra algumas delas:

Constante	Descrição
`TRUE`	Valor verdadeiro (utilizado para comparação).
`FALSE`	Valor falso.

`__FILE__`	Contém o nome do script que está sendo executado.
`__LINE__`	Contém o número da linha do script que está sendo executado.
`PHP_VERSION`	Contém a versão corrente do PHP.
`PHP_OS`	Nome do sistema operacional no qual o PHP está rodando.
`E_ERROR`	Exibe um erro ocorrido em um script. A execução é interrompida.
`E_WARNING`	Exibe uma mensagem de aviso do PHP. A execução não pára.
`E_PARSE`	Exibe um erro de sintaxe. A execução é interrompida.
`E_NOTICE`	Mostra que ocorreu algo, não necessariamente um erro. A execução não pára.

Variáveis

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 declarar variáveis. Basta atribuir diretamente um valor a estas e, a partir desse momento, já estarão criadas e associadas a um tipo (numérico, alfanumérico etc.), dependendo do valor que lhes foi atribuído.

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 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_america
```

- **Inválidas**

```
$100vergonha
$5
$20assustar
$60nacadeira
```

É recomendável que você utilize sempre identificadores com letras minúsculas, pois o PHP faz a distinção entre maiúsculas e minúsculas. Se você misturar os dois

tipos de letras, poderá confundir-se ao utilizar as variáveis, 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 digitará a seguinte linha:

```
<?php
    echo $Nota_aluno;
?>
```

O aluno será reprovado, pois você imprimiu a variável errada. Portanto, tome cuidado: o PHP distingue letras maiúsculas de minúsculas para nomes de variáveis.

Escopo das variáveis

Em relação ao escopo, quando uma variável é utilizada dentro de uma função, pode haver uma outra variável com o mesmo nome sendo utilizada em outra função ou no código do programa principal. São espaços de memória diferentes, e cada um funciona dentro de seu contexto, ou seja, a variável definida dentro da função só pode ser acessada ali dentro. Fora dali, seu valor não é acessível em nenhuma outra parte do programa.

No entanto, é possível usar dentro de uma função o valor de uma variável existente também no programa principal. 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.

Como exemplo, acompanhe o programa `prog6.php`:

prog6.php

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

Ao executar esse programa, serão mostrados os seguintes valores na tela:

```
5000
5005
```

Isso ocorreu porque no início da função `testa_escopo1` se indicou que a variável `$num` era a mesma utilizada globalmente. Desse modo, todas as alterações feitas na variável `$num` dentro da função, alteraram o valor que existia nessa variável global. Se retirássemos da função a declaração `global`, a variável `$num` não seria reconhecida.

Variáveis geradas dinamicamente

Em diversas ocasiões é muito útil criarmos variáveis dinamicamente, ou seja, durante a execução do programa. Essa técnica funciona assim: 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:

prog7.php

```
<?php
    $nome = "Juliano";
    $futuro_identificador = "autor";
    $$futuro_identificador = $nome;
    echo "O nome do autor é ";
    echo $autor;
?>
```

Ao executar o programa `prog7.php` em seu navegador, você verá a seguinte frase:

```
O nome do autor é Juliano
```

Veja que a variável `$autor` foi criada dinamicamente. Primeiro o valor "autor" foi atribuído à variável `$futuro_identificador` e, depois, com o uso de `$$`, o valor "autor" tornou-se o identificador da variável recém-criada. A variável criada `$autor` recebeu o valor da variável `$nome` e, em seguida, seu conteúdo foi mostrado na tela após a frase `"O nome do autor é "`.

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

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 trata-se 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, existe 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 a seu índice, que é o número de 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 a 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 irá procurar 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 nesses casos o texto é chamado de chave associativa:

```
$vetor["time"] = "Grêmio";
$vetor["titulo"] = "Tetracampeão da Copa do Brasil";
$vetor["ano"] = 2001;
```

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 situação que pode ocorrer é o array possuir índices numéricos e strings ao mesmo tempo. Não há problema algum em usar os dois tipos de índices no mesmo array.

Existem também as matrizes, que são arrays multidimensionais. Essas estruturas de armazenamento também contêm 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";
```

Esse exemplo 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 `prog8.php`, apresentado a seguir:

prog8.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 é iniciado 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 se criou um array que possui índices numéricos e também uma chave

associativa. Com o uso do comando `echo`, mostramos na tela os valores de duas posições desse array.

Objetos

Também é possível programar em PHP com o modelo orientado a objetos. Podemos definir uma classe e, dentro dela, as variáveis e funções que estarão disponíveis a seus objetos. Um objeto é a variável que utilizamos para instanciar uma classe. Se você alguma vez já estudou programação orientada a objetos, esse conceito deve lhe ser familiar. Veja um pequeno exemplo:

prog9.php

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

Ao criar uma instância da classe na variável `$objeto`, podemos acessar as funções definidas dentro da classe. Esse programa mostrará a mensagem `"Oi pessoal!"`.

Uma das principais novidades do PHP 5 é o novo modelo de orientação a objetos. A partir dessa versão, o PHP passa a operar com a Zend Engine 2.0. Para obter mais informações, consulte o Apêndice B deste livro, que apresenta as principais novidades desse novo modelo.

Operadores

Como o próprio nome indica, os operadores informam ao PHP quais operações devem ser executadas com os valores recebidos, como, por exemplo, atribuir um valor a uma variável, realizar operações aritméticas (soma, subtração etc.), realizar comparação de valores, testar se um é maior ou menor que o outro etc. Veremos os seguintes tipos de operadores: aritméticos, binários, de comparação, de atribuição, e ternário.

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-se do incremento utilizando o operador `++`. No PHP também é possível utilizá-lo. Vamos conhecer todos esses operadores com a tabela a seguir:

Operador	Descrição
`-oper`	Troca o sinal do operando.
`++oper`	Pré-incremento. Primeiro incrementa o valor do operando e depois realiza a operação.
`--oper`	Pré-decremento. Primeiro decrementa o valor do operando e depois realiza a operação.
`oper++`	Pós-incremento. Primeiro realiza a operação e depois incrementa o operando.
`oper--`	Pós-decremento. Primeiro realiza a operação e depois decrementa o operando.

Os operadores apresentados nessa 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.

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

Exemplo:

```
$contador++;
```

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. Ao entender como funcionam, seus códigos ficarão mais simples e claros.

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 conhecê-los:

Operador	Descrição
~*op1*	Inverte os bits de *op1*.
op1 & *op2*	Operação E (AND) bit a bit.
op1 | *op2*	Operação OU (OR) bit a bit.
op1 ^ *op2*	Operação OU exclusivo (XOR).
op1 >> *n*	Desloca *op1 n* bits à direita.
op1 << *n*	Desloca *op1 n* bits à esquerda.

Veja um exemplo:

prog10.php

```
<html>
<body>
<?php
    $num = 50;
    $deslocado = $num >> 1;   // desloca 1 bit para direita
    echo  $deslocado;
?>
</body>
</html>
```

No exemplo apresentado, o valor 50 (que equivale a 110010 na base binária) é deslocado um bit à direita, o que equivale a dividi-lo por 2. O resultado escrito na tela será a divisão de 50 por 2, que dá 25 (equivalente na base binária a 11001).

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.

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 o da outra ou se possui um valor maior que o de um determinado número, ou se o retorno dado pela chamada de uma função é verdadeiro ou falso. Veja a tabela a seguir:

Operador	Descrição
op1 == op2	Verdadeiro se *op1* for igual a *op2*.
op1 === op2	Verdadeiro se *op1* for igual a *op2* e se ambos forem do mesmo tipo.
op1 >= op2	Verdadeiro se *op1* for maior que ou igual a *op2*.
op1 <= op2	Verdadeiro se *op1* for menor que ou igual a *op2*.
op1 != op2	Verdadeiro se *op1* for diferente de *op2*.
op1 !== op2	Verdadeiro se *op1* for diferente de *op2* ou se ambos não forem do mesmo tipo.
op1 <> *op2*	Também serve para representar diferença.
op1 > *op2*	Verdadeiro se *op1* for maior que *op2*.
op1 < *op2*	Verdadeiro se *op1* for menor que *op2*.

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

Atribuição

Atribuição é o termo utilizado 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 à 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*.

Muitas vezes programas podem apresentar problemas em razão da troca do operador de comparação == pelo operador de atribuição =. O programa acaba gerando resultado incorreto, pois se quisermos fazer uma comparação, por exemplo, entre as variáveis $a e $b, deveremos usar a expressão $a==$b, e não $a=$b.

Vamos ver um exemplo envolvendo operadores de atribuição:

prog11.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.

Lógicos

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

Operador	Descrição
!op1	Verdadeiro se *op1* for falso.
op1 AND *op2*	Verdadeiro se *op1* E *op2* forem verdadeiros.

op1 OR *op2*	Verdadeiro se *op1* OU *op2* forem verdadeiros.
op1 XOR *op2*	Verdadeiro se só *op1* ou só *op2* for verdadeiro.
op1 && *op2*	Verdadeiro se *op1* E *op2* forem verdadeiros.
op1 || *op2*	Verdadeiro se *op1* OU *op2* 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 ao usá-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 termos um formulário em que os campos nome, e-mail e CPF são obrigatórios. Certamente no programa que recebe os dados do formulário haverá um teste, como o mostrado no trecho de programa a seguir:

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

Nesse exemplo, temos uma expressão sendo avaliada. A função empty(), que significa vazio em português, retorna verdadeiro se a variável estiver vazia e falso se houver algo na variável. Então, estamos testando se a variável $nome está vazia ou a variável $email ou a variável $cpf está vazia. Se pelo menos uma das três estiver vazia, o resultado será verdadeiro, e isso fará 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.

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)

Exp1	**Resultado**
V	F
F	V

Ternário

É uma forma abreviada de usar o comando condicional `if`, que veremos mais adiante. Uma condição é avaliada e, caso seja verdadeira, atribui-se um valor à variável, e caso seja 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. Observe:

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

Quando o PHP executar a linha anterior, se a variável `$frequencia` possuir um valor maior que ou igual a 0,75, a variável `$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`, *torna-se* bem mais fácil entender qual é o objetivo do código, pois conseguimos distinguir perfeitamente qual é a condição e quais operações serão executadas após a avaliação desta.

Precedência de operadores

Para evitar erros de lógica em 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
`new`	Criação de objetos.
`[`	Colchete.
`! ~ ++ -- (int) (float)`	
`(string) (array)`	
`(object) @`	Não-lógico, inversão de bits, incremento e decremento, conversão de tipos e controle de erro.
`* / %`	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.
`print`	Impressão.

AND	AND lógico (de menor prioridade).
XOR	XOR lógico (de menor prioridade).
OR	OR lógico (de menor prioridade).
,	Vírgula.

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 usada 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 mesma expressão, e não existirem parênteses, o PHP resolverá a expressão da esquerda para a direita.

Observe o seguinte trecho de programa:

prog12.php

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

O resultado mostrado na tela será:

```
6
20
```

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.

Analisando esse exemplo, você pode perceber a importância da precedência dos operadores. Muitas vezes o uso incorreto dessas informações causa a exibição de resultados errados na saída dos programas.

Estruturas de controle 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 e repetir um bloco de comandos por um certo número de vezes ou até que uma condição seja atingida. Veremos os seguintes comandos:

- Comandos condicionais: `if` e `switch`.
- Comandos de repetição: `while`, `do...while`, `for` e `foreach`.

Vamos ver então como utilizar cada um deles.

if

Comando que avalia uma expressão e, dependendo do resultado, executa-se 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 else, "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 vier após o else.

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

```
<?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 tirasse nota 10.

switch

O comando `switch` é similar `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. 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, se deve utilizar 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 a seguir um exemplo de uso desse comando:

```
<?php
switch ($numero)
{
    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 aparecer após o `default`. O uso do `default` não é obrigatório no comando `switch`.

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 é 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
}
```

Devemos tomar cuidado para não colocar 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 `while`:

prog13.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 que o loop seja encerrado.

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, 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. A sintaxe do comando é a seguinte:

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

Veja a seguir um exemplo:

prog14.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 dado número 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
    }
```

Como inicialização, geralmente determinamos o valor inicial da variável que controlará o loop. O parâmetro de inicialização poderá ser `$cont=0`. No segundo parâmetro, devemos colocar a condição que deve ser atingida para que o laço continue. Se quiséssemos executar o loop 20 vezes, o valor do parâmetro de condição seria `$cont<20`. Quando essa condição retorna 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 interação do loop, o valor da variável de controle é atualizado automaticamente, dependendo do terceiro parâmetro que você definiu quando utilizou o comando `for`. Veja um exemplo de utilização do comando:

prog15.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
```

foreach

O comando `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 `$nome_array`, e a cada interação o valor do elemento corrente do array é atribuído à variável `$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 segue o mesmo critério, mas com uma diferença: além de o valor do elemento corrente do array ser atribuído à variável `$elemento`, a chave (ou índice) do elemento atual é atribuído à variável `$chave`. Acompanhe o exemplo a seguir:

prog16.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, todas as chaves e valores do array `$a`. O segundo `foreach` desse exemplo mostrará o seguinte:

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

Integração com bancos de dados

O PHP suporta diversos SGBDs (Sistemas de Gerência de Bancos de Dados), oferecendo um conjunto de funções para executar operações (consultas, inclusões, alterações, exclusões etc.) sobre cada um deles. Entre eles, temos: MySQL, PostgreSQL, SQLite, InterBase, Oracle, SQL Server, Sybase, entre outros que oferecem suporte à linguagem SQL (Structured Query Language). Os bancos de dados não suportados diretamente pelo PHP podem ser acessados via ODBC. Os comandos existentes para cada um dos SGBDs estão disponíveis na documentação do PHP, que pode ser obtida no site oficial (`http://www.php.net`).

Este livro não tem o objetivo de ensinar a linguagem SQL nem de mostrar como instalar um determinado sistema de banco de dados. Neste tópico, faremos apenas uma rápida revisão de como utilizar o PHP para acessar e realizar consultas sobre o MySQL (`http://www.mysql.com`), que é um dos SGBDs mais utilizados com o PHP. Essa revisão será importante, visto que alguns exemplos que serão apresentados neste livro necessitam de acesso ao banco de dados.

Mais adiante, no capítulo 10, você irá aprender outra forma de realizar operações sobre um banco de dados. Nele serão apresentadas bibliotecas de abstração, que visam a oferecer um conjunto de comandos único para acessar diferentes SGBDs.

Vamos ver então as principais funções para operar sobre o MySQL. Antes de acessar um banco de dados e começar a realizar operações sobre ele, precisamos estabelecer uma conexão com o servidor MySQL. Para isso, utilizaremos a função de conexão `mysql_connect`, que possui a seguinte sintaxe:

```
recurso mysql_connect ([string servidor [, string usuário [, string senha
   [, bool novo_link [, int flags_cliente ]]]]])
```

Parâmetro	Descrição
servidor	Endereço do servidor no qual está localizado o banco de dados.
usuário	Nome de usuário a ser utilizado para a abertura da conexão.
senha	Senha a ser utilizada para a abertura da conexão.
novo_link	Indica se deve ser aberto um novo link quando for feita mais de uma chamada a essa função com os mesmos parâmetros.

flags_cliente	Define algumas configurações do cliente. Pode ser uma combinação das constantes MYSQL_CLIENT_COMPRESS, MYSQL_CLIENT_IGNORE_SPACE e MYSQL_CLIENT_INTERACTIVE.

Em caso de sucesso, essa função retorna o identificador da conexão, que posteriormente será passado como parâmetro para a função de fechamento desta. Em caso de falha, retorna FALSE. Para fazer, por exemplo, a conexão com um banco de dados chamado bdteste, que possui como nome de usuário juliano e como senha teste, podemos executar o seguinte comando:

```
$conexao = mysql_connect ("localhost", "juliano", "teste");
```

Se o banco de dados estiver localizado em um servidor diferente, bastará substituir localhost pelo nome ou endereço IP desse servidor. Após abrir a conexão, o próximo passo é selecionar, por meio do comando mysql_select_db, qual será o banco de dados utilizado. Por exemplo:

```
mysql_select_db ("bdteste");
```

A partir daí, já podemos realizar operações sobre o banco de dados. A função PHP responsável por executar comandos SQL é a mysql_query, que possui a seguinte sintaxe:

```
recurso mysql_query (string comando [, recurso id_conexão])
```

Parâmetro	Descrição
comando	Comando SQL a ser executado.
id_conexão	Pode ser usado caso exista mais de uma conexão aberta; deve conter o identificador da conexão na qual o comando deve ser executado.

No caso da execução de comandos SELECT, a função mysql_query irá retornar um conjunto de resultados (se houver). Existem várias formas de obter os valores desses resultados, utilizando funções como mysql_result, mysql_fetch_row, mysql_fetch_assoc e mysql_fetch_array. Após realizar todas as operações necessárias, devemos encerrar a conexão com o banco de dados executando a função mysql_close.

Vamos ver rapidamente um exemplo de programa PHP que realiza todos os passos descritos nesse tópico. Antes de criar o programa, vamos criar uma tabela no MySQL, na qual ficarão armazenados os dados. Essa tabela, que irá armazenar informações sobre livros, será criada da seguinte forma no utilitário mysql:

```
create table livros
(
   isbn varchar(13),
   titulo varchar(80) NOT NULL,
   autor varchar(80) NOT NULL,
   paginas smallint NOT NULL,
```

```
    preco float NOT NULL
);
```

O programa PHP deverá abrir uma conexão com o servidor MySQL, selecionar todos os nomes de livros e seus respectivos autores e, logo após, exibir essas informações na tela, uma em cada linha. Acompanhe, então, o código do programa `prog17.php` apresentado a seguir:

prog17.php

```
<?php
$servidor = "localhost";
$usuario = "juliano";
$senha = "12345";
$banco = "test";
$con = mysql_connect($servidor, $usuario, $senha);
    mysql_select_db ($banco);
    $res = mysql_query("SELECT titulo,autor FROM livros");
    $num_linhas = mysql_num_rows($res);
    for ($i = 0; $i < $num_linhas; $i++)
    {
        $dados = mysql_fetch_row ($res);
        $titulo = $dados[0];
        $autor = $dados[1];
        echo "$titulo - $autor <br>";
    }
mysql_close($con);
?>
```

Veja que para percorrer o resultado com o comando `for`, utilizou-se antes a função `mysql_num_rows`, que retorna o número de linhas resultantes em uma consulta SQL. Ao executar esse programa no navegador, será exibido, para cada um dos livros, seu título e, ao lado, o nome do autor.

Importante: se sua versão do MySQL for 4.1 ou superior, para aproveitar todos seus recursos, você deverá usar os comandos da biblioteca "mysqli" (Improved MySQL). Por exemplo:

- em vez de `mysql_connect`, use a função `mysqli_connect`;
- em vez de `mysql_query`, use a função `mysqli_query`;

e assim por diante. No site da Novatec Editora, o exemplo anterior está disponível também na versão da biblioteca "mysqli".