Strings

Nome do tipo	Espaço utilizado
CHAR(M)	M bytes, 1 <= M <= 255
VARCHAR(M)	L+1 bytes, onde L <= M e 1 <= M <= 255
TINYBLOB, TINYTEXT	L+1 bytes, onde L < 2^8
BLOB, TEXT	L+2 bytes, onde L < 2^16
MEDIUMBLOB, MEDIUMTEXT	L+3 bytes, onde L < 2^24
LONGBLOB, LONGTEXT	L+4 bytes, onde L < 2^32
ENUM('value1','value2',...)	1 ou 2 bytes
SET('value1','value2',...)	1, 2, 3, 4 ou 8 bytes

APÊNDICE 03 – Tipos suportados pelo mySQL

Numéricos

Nome do tipo	Espaço utilizado
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 if X <= 24 or 8 if 25 <= X <= 53
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M bytes (D+2, if M < D)
NUMERIC(M,D)	M bytes (D+2, if M < D)

Data e Hora

Nome do tipo	Espaço utilizado
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uksort

```
void uksort(array &arr, function compara);
```

Esta função ordena o array através dos índices, mantendo os relacionamentos com os elementos., e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois índices do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

asort

```
void asort(array &arr);
```

Tem o funcionamento bastante semelhante à função `sort`. Ordena os elementos de um array em ordem crescente, porém mantém os relacionamentos com os índices.

arsort

```
void arsort(array &arr);
```

Funciona de maneira inversa à função `asort`. Ordena os elementos de um array em ordem decrescente e mantém os relacionamentos dos elementos com os índices.

ksort

```
void ksort(array &arr);
```

Função de ordenação baseada nos índices. Ordena os elementos de um array de acordo com seus índices, em ordem crescente, mantendo os relacionamentos.

usort

```
void usort(array &arr, function compara);
```

Esta é uma função que utiliza outra função como parâmetro. Ordena os elementos de um array sem manter os relacionamentos com os índices, e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uasort

```
void uasort(array &arr, function compara);
```

Esta função também utiliza outra função como parâmetro. Ordena os elementos de um array e mantém os relacionamentos com os índices, utilizando para efeito de comparação uma função definida pelo usuário, que deve comparar dois

índice do valor atual, e os elementos de índices 1 e “value” contém o valor do elemento atual indicado pelo ponteiro.

Esta função pode ser utilizada para percorrer todos os elementos de um array e determinar se já foi encontrado o último elemento, pois no caso de haver um elemento vazio, a função não retornará o valor `false`. A função `each` só retorna `false` depois q o último elemento do array foi encontrado.

Exemplo:

```
/*função que percorre todos os elementos de um array e imprime seus índices e valores */
function imprime_array($arr) {
  reset($arr);
  while (list($chave,$valor) = each($arr))
    echo “Chave: $chave. Valor: $valor”;
}
```

Funções de ordenação

São funções que servem para arrumar os elementos de um array de acordo com determinados critérios. Estes critérios são: manutenção ou não da associação entre índices e elementos; ordenação por elementos ou por índices; função de comparação entre dois elementos.

sort

```
void sort(array &arr);
```

A função mais simples de ordenação de arrays. Ordena os elementos de um array em ordem crescente, sem manter os relacionamentos com os índices.

rsort

```
void rsort(array &arr);
```

Funciona de maneir ainversa à função `sort`. Ordena os elementos de um array em ordem decrescente, sem manter os relacionamentos com os índices.

next

```
mixed next(array arr);
```

Seta o ponteiro interno para o próximo elemento do array, e retorna o conteúdo desse elemento.

Obs.: esta não é uma boa função para determinar se um elemento é o último do array, pois pode retornar `false` tanto no final do array como no caso de haver um elemento vazio.

prev

```
mixed prev(array arr);
```

Seta o ponteiro interno para o elemento anterior do array, e retorna o conteúdo desse elemento. Funciona de maneira inversa a `next`.

pos

```
mixed pos(array arr);
```

Retorna o conteúdo do elemento atual do array, indicado pelo ponteiro interno.

key

```
mixed key(array arr);
```

Funciona de maneira bastante semelhante a `pos`, mas ao invés de retornar o elemento atual indicado pelo ponteiro interno do array, retorna seu índice.

each

```
array each(array arr);
```

Retorna um array contendo o índice e o elemento atual indicao pelo ponteiro interno do array. o valor de retorno é um array de quatro elementos, cujos índices são 0, 1, “key” e “value”. Os elementos de índices 0 e “key” armazenam o

A função range cria um array cujos elementos são os inteiros pertencentes ao intervalo fornecido, inclusive. Se o valor do primeiro parâmetro for maior do que o do segundo, a função retorna `false` (valor vazio).

shuffle

```
void shuffle(array &arr);
```

Esta função "embaralha" o array, ou seja, troca as posições dos elementos aleatoriamente e não retorna valor algum.

sizeof

```
int sizeof(array arr);
```

Retorna um valor inteiro contendo o número de elementos de um array. Se for utilizada com uma variável cujo valor não é do tipo array, retorna 1. Se a variável não estiver setada ou for um array vazio, retorna 0.

Funções de "navegação"

Toda variável do tipo array possui um ponteiro interno indicando o próximo elemento a ser acessado no caso de não ser especificado um índice. As funções seguintes servem para modificar esse ponteiro, permitindo assim percorrer um array para verificar seu conteúdo (chaves e elementos).

reset

```
mixed reset(array arr);
```

Seta o ponteiro interno para o primeiro elemento do array, e retorna o conteúdo desse elemento.

end

```
mixed end(array arr);
```

Seta o ponteiro interno para o último elemento do array, e retorna o conteúdo desse elemento.

APÊNDICE 02 - Funções para tratamento de arrays

Funções Genéricas

Array

```
array array(...);
```

É a função que cria um array a partir dos parâmetros forncidos. É possível fornecer o índice de cada elemento. Esse índice pode ser um valor de qualquer tipo, e não apenas de inteiro. Se o índice não for fornecido o PHP atribui um valor inteiro sequencial, a partir do 0 ou do último índice inteiro explicitado. Vejamos alguns exemplos:

```
Exemplo 1
$teste = array("um", "dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => “um” (0 é o primeiro índice, se não houver um explicito)
1 => “dois” (o inteiro seguinte)
“tr” => “tres”
5 => “quatro” (valor explicitado)
6 => “cinco” (o inteiro seguinte ao último atribuído, e não o
próximo valor, que seria 2)

Exemplo 2
$teste                    =                    array("um",
6=>"dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => “um”
6 => “dois”
“tr” => tres
5 => “quatro” (seria 7, se não fosse explicitado)
7 => “cinco” (seria 6, se não estivesse ocupado)
```

Em geral, não é recomendável utilizar arrays com vários tipos de índices, já que isso pode confundir o programador. No caso de realmente haver a necessidade de utilizar esse recurso, deve-se ter bastante atenção ao manipular os índices do array.

range

```
array range(int minimo, int maximo);
```

str_replace

```
string str_replace(string str1, string str2, string str3);
```

Altera todas as ocorrências de `str1` em `str3` pela string `str2`.

Funções diversas

chr

```
string chr(int ascii);
```

Retorna o caracter correspondente ao código ASCII fornecido.

ord

```
int ord(string string);
```

Retorna o código ASCII correspontente ao caracter fornecido.

echo

```
echo(string arg1, string [argn]... );
```

Imprime os argumentos fornecidos.

print

```
print(string arg);
```

Imprime o argumento fornecido.

strlen

```
int strlen(string str);
```

Retorna o tamanho da string fornecida.

strtolower

```
string strtolower(string str);
```

Retorna a string fornecida com todas as letras minúsculas.

```
Exemplo:
strtolower(“Teste”); // retorna “teste”
```

strtoupper

```
string strtoupper(string str);
```

Retorna a string fornecida com todas as letras maiúsculas.

```
Exemplo:
strtolower(“Teste”); // retorna “TESTE”
```

ucfirst

```
string ucfirst(string str);
```

Retorna a string fornecida com o primeiro caracter convertido para letra maiúscula.

```
Exemplo:
ucfirst(“teste de funcao”); // retorna “Teste de funcao”
```

ucwords

```
string ucwords(string str);
```

Retorna a string fornecida com todas as palavras iniciadas por letras maiúsculas.

```
Exemplo:
ucwords(“teste de funcao”); // retorna “Teste De Funcao”
```

Funções para edição de strings

chop

```
string chop(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
chop(“   Teste      \n  \n  “); // retorna “   Teste”
```

ltrim

```
string ltrim(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
ltrim(“   Teste  \n  \n “); // retorna “Teste  \n  \n”
```

trim

```
string trim(string str);
```

Retira espaços e linhas em branco do início e do final da string fornecida.

```
Exemplo:
trim(“   Teste  \n  \n “); // retorna “Teste”
```

strrev

```
string strrev(string str);
```

Retorna a string fornecida invertida.

```
Exemplo:

strrev(“Teste”); // retorna “etseT”
```

Funciona de maneira semelhante à função `strcasecmp`, com a diferença que esta é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

strstr

```
string strstr(string str1, string str2);
string strchr(string str1, string str2);
```

As duas funções são idênticas. Procura a primeira ocorrência de `str2` em `str1`. Se não encontrar, retorna uma string vazia, e se encontrar retorna todos os caracteres de `str1` a partir desse ponto.

```
Exemplo:

strstr("Mauricio Vivas", "Viv"); // retorna “Vivas”
```

stristr

```
string strstr(string str1, string str2);
```

Funciona de maneira semelhante à função `strstr`, com a diferença que esta é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strpos

```
int strpos(string str1, string str2, int [offset] );
```

Retorna a posição da primeira ocorrência de `str2` em `str1`, ou zero se não houver. O parâmetro opcional `offset` determina a partir de qual caracter de str1 será efetuada a busca. Mesmo utilizando o offset, o valor de retorno é referente ao início de str1.

strrpos

```
int strrpos(string haystack, char needle);
```

Retorna a posição da última ocorrência de `str2` em `str1`, ou zero se não houver.

explode

```
array explode(string padrao, string str);
```

Funciona de maneira bastante semelhante à função `split`, com a diferença que não é possível estabelecer um limite para o número de elementos do array.

Comparações entre strings

similar_text

```
int similar_text(string str1, string str2, double [porcentagem]);
```

Compara as duas strings fornecidas e retorna o número de caracteres coincidentes. Opcionalmente pode ser fornecida uma variável, passada por referência *(ver tópico sobre funções)*, que receberá o valor percentual de igualdade entre as strings. Esta função é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

```
Exemplo:

$num = similar_text("teste", "testando",&$porc);

As variáveis passam a ter os seguintes valores:

$num == 4; $porc == 61.538461538462
```

strcasecmp

```
int strcasecmp(string str1, string str2);
```

Compara as duas strings e retorna 0 (zero) se forem iguais, um valor maior que zero se `str1 > str2`, e um valor menor que zero se `str1 < str2`. Esta função é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strcmp

```
int strcasecmp(string str1, string str2);
```

urldecode

```
string urldecode(string str);
```

Funciona de maneira inversa a urlencode, desta vez decodificando a string fornecida do formato urlencode para texto normal.

Funções relacionadas a arrays

Implode e join

```
string implode(string separador, array partes);
string join(string separador, array partes);
```

As duas funções são idênticas. Retornam uma string contendo todos os elementos do array fornecido separados pela string também fornecida.

```
Exemplo:
$partes = array("a", "casa número", 13, "é azul");
$inteiro = join(" ",$partes);

$inteiro passa a conter a string:
“a casa número 13 é azul”
```

split

```
array split(string padrao, string str, int [limite]);
```

Retorna um array contendo partes da string fornecida separadas pelo padrão fornecido, podendo limitar o número de elementos do array.

```
Exemplo:
$data = “11/14/1975”;
$data_array = split(“/”,$data);

O código acima faz com que a variável $data_array receba o valor:
array(11,14,1975);
```

get_meta_tags

```
array get_meta_tags(string arquivo);
```

Abre um arquivo html e percorre o cabeçalho em busca de “meta” tags, retornando num array todos os valores encontrados.

```
Exemplo:
No arquivo teste.html temos:
...
<head>
<meta name="author" content="jose">
<meta name="tags" content="php3 documentation">
...
</head><!-- busca encerra aqui -->
...

a execução da função:

get_meta_tags(“teste.html”);

retorna o array:

array(“author”=>”jose”,”tags”=>"php3 documentation");
```

strip_tags

```
string strip_tags(string str);
```

Retorna a string fornecida, retirando todas as tags html e/ou PHP encontradas.

```
Exemplo:
strip_tags('<a href="teste1.php3">testando</a><br>');

Retorna a string “testando”
```

urlencode

```
string urlencode(string str);
```

Retorna a string fornecida, convertida para o formato urlencode. Esta função é útil para passar variáveis para uma próxima página.

APÊNDICE 01 - Funções para tratamento de strings

Funções relacionadas a HTML

htmlspecialchars

```
string htmlspecialchars(string str);
```

Retorna a string fornecida, substituindo os seguintes caracteres:

- & para '&'
- " para '"'
- < para '<'
- > para ‘>'

htmlentities

```
string htmlentities(string str);
```

Funciona de maneira semelhante ao comando anterior, mas de maneira mais completa, pois converte todos os caracteres da string que possuem uma representação especial em html, como por exemplo:

- º para 'º'
- ª para 'ª'
- á para 'á'
- ç para ‘ç'

nl2br

```
string nl2br(string str);
```

Retorna a string fornecida substituindo todas as quebras de linha (“\n”) por quebras de linhas em html (“
”).

```
Exemplo:
echo nl2br(“Mauricio\nVivas\n”);

Imprime:
Maurício<br>Vivas<br>
```

20. Bibliografia e Referências

A pesquisa foi baseada no manual de PHP, disponível em ***www.php.net***, e em diversos tutoriais disponíveis no site ***www.phpbuilder.com***. Esses dois endereços contém uma vasta documentação sobre a linguagem, além de endereços para listas de discussão, onde pode-se solicitar ajuda de programadores mais experientes.

Uma boa referência em português é a lista *"PHP para quem fala Português"*, que pode ser assinada no endereço ***www.egroups.com/group/php-pt/***.

Em inglês, além dos endereços citados acima, uma boa fonte é o site *PHPWizard*, que pode ser encontrado em ***www.phpwizard.net***.

Também em inglês, uma documentação mais completa sobre cookies pode ser encontrada em ***www.netscape.com/newsref/std/cookie_spec.html***.

19. Enviando e-mail

Para enviar e-mail através de um script PHP é bastante simples. Basta utilizar a função `mail`:

```
mail(string to, string subject, string message, string [headers]);
```

onde:

to – string contendo o e-mail do destinatário;

subject – assunto da mensagem;

message – o corpo da mensagem.

headers – outras informações de cabeçalho, como por exemplo “from”, “reply-to”, “bcc”, etc.

Para facilitar a compreensão dos scripts, os argumentos (como geralmente são strings grandes) devem ser atribuídos a variáveis antes da chamada da função `mail`.

```
<?
$id = fopen($teste, “r”); /* abre o arquivo para leitura */
$teste_conteudo = fread($id,filesize($teste)); /* le o conteudo
do arquivo e grava na variavel $conteudo */
fclose($id); /* fecha o arquivo */
?>
```

Com o exemplo acima, teremos o conteudo do arquivo enviado armazenado na string `$teste_conteudo`, podendo assim ser armazenado onde for mais adequado.

```
<?
$arquivo = "teste.gif"; /* este nome deve ser alterado para o nome do arquivo a ser utilizado */
$id = fopen($arquivo, "r"); /* abre o arquivo para leitura */
$conteudo = fread($id,filesize($arquivo)); /* le o conteudo do arquivo e grava na variavel $conteudo */
fclose($id); /* fecha o arquivo */

header("Content-type: image/gif"); /* esta linha envia um header ao browser informando que o tipo de arquivo que está sendo enviado é uma imagem no formato gif */
echo $conteudo; /* esta última linha envia ao browser o conteúdo do arquivo */
?>
```

Para que o exemplo funcione corretamente é preciso que o script seja apenas o que está listado, não podendo haver texto algum (nem mesmo espaço ou linha em branco) antes e depois do script. Visualisando o script pelo browser, teremos a imagem selecionada.

Uploads com formulários HTML

Vimos no capítulo 03 que os formulários HTML têm um tipo de componente utilizado em upload de arquivos. Vimos também que todos os elementos de formulários quando submetidos a scripts PHP criam variáveis com os mesmos nomes. mas no caso do elemento `"file"`, o tratamento é diferente. Ao ser submetido o formulário, o arquivo uploadeado é gravado num arquivo temporário do disco, que será apagado ao final da execução do script. Além disso, quatro variáveis são criadas no contexto do script PHP.

$meuarquivo – nome do arquivo temporario criado;

$meuarquivo_name – nome original do arquivo selecionado pelo usuário;

$meuarquivo_size – tamanho do arquivo enviado;

$meuarquivo_type – tipo do arquivo, se esta informação for fornecida pelo browser;

Neste exemplo, "meu arquivo" é o nome do elemento do formulário.

Para armazenar o conteúdo de um arquivo numa tabela da base de dados ou até num arquivo definitivo (neste segundo caso é mais simples utilizar a função `copy`) podemos utilizar o seguinte script, supondo que o campo do formulário tenha o nome "teste":

Para encerrar a manipulação de um arquivo deve-se utilizar a função fclose, que tem o seguinte formato:

```
int fclose(int fp);
```

Onde fp é o identificador do arquivo, retornado pela função fopen.

Lendo de um arquivo

```
string fread(int fp, int tamanho);
```

Esta função retorna uma string com o conteúdo do arquivo. O segundo parâmetro determina até onde o arquivo será lido. Se o tamanho determinado for maior que o arquivo, não ocorre erro, tendo como retorno apenas o arquivo. Na maioria dos casos a função `filesize` é bastante útil, como no exemplo abaixo:

```
$meuarquivo = "c:/autoexec.bat";
$id = fopen($meuarquivo, "r");
$conteudo = fread($id,filesize($meuarquivo));
```

A função fread é "binary-safe", ou seja, pode ser usada para ler o conteúdo de um arquivo binário. Obviamente nesse caso é preciso saber exatamente onde utilizar o valor lido, para não obter resultados indesejados.

Escrevendo em um arquivo

```
int fwrite(int fp, string conteudo, int [tamanho]);
```

Esta função grava num arquivo o conteúdo do segundo parâmetro. Se o tamanho é fornecido e for menor que o tamanho da string, será feita a gravação apenas de uma parte da mesma, determinada pelo terceiro parâmetro.

Exemplo

Para demonstrar a leitura de um arquivo, utilizaremos um exemplo que necessita apenas de uma imagem do tipo GIF, que deve estar no mesmo diretório que nosso script de exemplo.

w	Abre o arquivo com permissão apenas para escrita. Se o arquivo existir, todo o conteúdo é apagado. Se não existir, o PHP tenta criá-lo. O ponteiro é posicionado no início do arquivo
w+	Abre o arquivo com permissão para escrita e leitura. Se o arquivo existir, todo o conteúdo é apagado. Se não existir, o PHP tenta criá-lo. O ponteiro é posicionado no início do arquivo
a	Abre o arquivo com permissão apenas para escrita. Se o arquivo não existir, o PHP tenta criá-lo. O ponteiro é posicionado no final do arquivo
a+	Abre o arquivo com permissão para escrita e leitura. Se o arquivo não existir, o PHP tenta criá-lo. O ponteiro é posicionado no final do arquivo.

O ponteiro citado na tabela dos modos de abrir um arquivo refere-se à posição a partir de onde os dados serão lidos e/ou gravados. Para alterar a posição desse ponteiro, pode-se utilizar a função fseek:

```
int fseek(int fp, int posição);
```

Onde fp é um identificador de arquivo, retornado da função fopen.

O terceiro parâmetro da função fopen, que pode ter valor `"0"` ou `"1"` indica se o `include_path` deverá ser utilizado para localizar o arquivo. O include_path é um parâmetro determinado no php.ini que indica exatamente em quais diretórios determinados arquivos serão procurados.

Além de abrir arquivos localmente, utilizando o sistema de arquivos, a função fopen também permite abrir arquivos remotos, utilizando os protocolos http ou ftp, da seguinte maneira:

Se a string como o nome do arquivo iniciar por “http://” (maiúsculas e minúsculas são iguais), uma conexão é aberta com o servidor e o arquivo contendo o texto de retorno será aberto. ATENÇÃO: qualquer alteração feita no arquivo afetará apenas o arquivo temporário local. O original será mantido.

Se a string como o nome do arquivo iniciar por “ftp://” (maiúsculas e minúsculas são iguais), uma conexão é aberta com o servidor e o arquivo será aberto. utilizando ftp o arquivo poderá ser aberto para leitura ou escrita, mas não simultaneamente.

Esta função tem um comportamento booleano: retorna apenas `true` ou `false`, não informando mais nada sobre o arquivo.

Limpando o cache

Por terem execução lenta, algumas funções que verificam o estado de arquivos utilizam um cache, ou seja, chamadas sucessivas da mesma função com relação ao mesmo arquivo não verificam se houve mudança no mesmo, retornando sempre o mesmo valor. Para eliminar esse cache, obrigando o PHP a reavaliar o valor de retorno de uma função, deve ser utilizada a seguinte função:

```
void clearstatcache();
```

A palavra “void” indica que a função não retorna valor algum.

As funções `filesize` e `file_exists` utilizam cache.

Abrindo arquivos para leitura e/ou escrita

Para ler ou escrever num arquivo é preciso antes de qualquer coisa abri-lo. Para isso deve ser utilizada a função fopen, como visto a seguir:

```
int fopen(string arquivo, string modo, int [use_include_path]);
```

A função fopen retorna `false` em caso de erro, e um identificador do arquivo em caso de sucesso. Esse identificador será utilizado em outras funções que manipulam o conteúdo do arquivo. O primeiro argumento é uma string contendo o nome do arquivo; o segundo, o modo como o arquivo será aberto, que pode ser um dos seguintes:

r	Abre o arquivo com permissão apenas para leitura.
r+	Abre o arquivo com permissão para escrita e leitura, posicionando o ponteiro no início do mesmo.

18. Manipulando arquivos

Através do PHP é possível ter acesso aos arquivos do sistema, e até arquivos remotos. A seguir veremos alguns dos comandos utilizados para manipular arquivos no PHP.

Copiando Arquivos

Para fazer uma cópia de arquivo utilizando PHP basta utilizar a função copy, desde que o usuário tenha as permissões necessárias para isso. A assinatura da função copy é a seguinte:

```
int copy(string origem string destino);
```

Lembrando que as strings contendo origem e destino devem conter os caminhos completos. Retorna false caso a cópia não seja realizada.

Verificando o tamanho de um arquivo

A função filesize pode ser bastante útil na criação de um script que liste o conteúdo de um diretório, mas também é utilizada em casos como a função fread, que será vista mais adiante.

```
int filesize(string arquivo);
```

Esta função retorna um inteiro com o tamanho do arquivo, em bytes, ou false em caso de erro.

Verificando se um arquivo existe

Para evitar erros em tratamento de arquivos, em certos casos é aconselhável verificar se determinado arquivo existe. para isso deve ser utilizada a função file_exists:

```
int file_exists(string arquivo);
```

secure: se tiver valor 1, indica que o cookie só pode ser transmitido por uma conexão segura (https).

Observações:

Um cookie não pode ser recuperado na mesma página que o gravou, a menos que esta seja recarregada pelo browser.

Múltiplas chamadas à função `setcookie` serão executadas em ordem inversa;

Cookies só podem ser gravados antes do envio de qualquer informação para o cliente. Portanto todas as chamadas à função `setcookie` devem ser feitas antes do envio de qualquer header ou texto.

Lendo cookies gravados

Os cookies lidos por um script PHP ficam armazenados em duas variáveis. no array `$HTTP_COOKIE_VARS[]`, tendo como índice a string do nome do cookie, e numa variável cujo nome é o mesmo do cookie, precedido pelo símbolo `$`.

Exemplo:

```
Um cookie que foi gravado numa página anterior pelo seguinte comando:
setcookie(“teste”, “meu cookie”);
Pode ser lida pela variável

$HTTP_COOKIE_VARS[“teste”]

ou pela variável

$teste
```

17. Utilizando cookies

O que são

Cookies são variáveis gravadas no cliente(browser) por um determinado site. Somente o site que gravou o cookie pode ler a informação contida nele. Este recurso é muito útil para que determinadas informações sejam fornecidas pelo usuário apenas uma vez. Exemplos de utilização de cookies são sites que informam a quantidade de vezes que você já visitou, ou alguma informação fornecida numa visita anterior.

Existem cookies persistentes e cookies de sessão. Os persistentes são aqueles gravados em arquivo, e que permanecem após o browser ser fechado, e possuem data e hora de expiração. Os cookies de sessão não são armazenados em disco e permanecem ativos apenas enquanto a sessão do browser não for encerrada.

Por definição, existem algumas limitações para o uso de cookies, listadas a seguir:

- 300 cookies no total
- 4 kilobytes por cookie.
- 20 cookies por servidor ou domínio.

Gravando cookies

Para gravar cookies no cliente, deve ser utilizada a função setcookie, que possui a seguinte assinatura:

```
int setcookie(string nome, string valor, int exp, string path, string dominio, int secure);
```

onde:

nome: nome do cookie;

valor: valor armazenado no cookie;

exp: data de expiração do cookie (opcional), no formato Unix. Se não for definida, o cookie será de sessão;

path: path do script que gravou o cookie;

dominio: domínio responsável pelo cookie;

16. Utilizando headers

O comando header permite enviar cabeçalhos html para o cliente. Deve ser utilizado por usuários que conheçam a função de cada header que pode ser enviado. Não pode ser enviado depois de algum texto. veja o seguinte exemplo:

```
<html>
<? header("Location: http://www.php.net"); ?>
<body>
...
```

O código acima causará um erro, já que tentou-se enviar um header depois de ter sido enviado um texto("<html>\n").

A sintaxe do comando header é bastante simples:

```
int header(string header);
```

Algumas utilizações do header são:

```
//redirecionar para outra página:
header("Location: http://www.php.net");

// Definir o script como uma mensagem de erro:
header("http/1.0 404 Not Found");

// Definir a expiração da página:
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Pragma: no-cache"); // HTTP/1.0
```

Para obter uma lista completa dos headers HTTP, visite o seguinte endereço:

```
http://www.w3.org/Protocols/rfc2068/rfc2068
```

```
int pg_result(int result, int linha, mixed [campo] );
```

Retorna o conteúdo de uma célula da tabela de resultados.

`result` é o identificador do resultado;

`linha` é o número da linha, iniciado por 0;

`campo` é uma string com o nome do campo, ou um número correspondente ao número da coluna. Se foi utilizado um alias na consulta, este deve ser utilizado no comando pg_result.

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

```
array pg_fetch_array(int result, int linha);
```

Lê uma linha do resultado e devolve um array, cujos índices são os nomes dos campos. O índice das linhas é iniciado por zero.

```
array pg_fetch_row(int result, int linha);
```

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

qual a base de dados selecionada, basta utilizar a função string pg_dbname, que tem a seguinte assinatura:

```
string pg_dbname(int conexão);
```

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

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

Verificando o erro na execução de uma query

Para ter acesso à mensagem de erro no caso de falha na execução de uma query SQl, basta utilizar o comando pg_errormessage():

```
string pg_errormessage(int connection);
```

Apagando o resultado

```
int pg_freeresult(int result);
```

O comando pg_freeresult deve ser utilizado para apagar da memória o resultado indicado. No PHP 4, este comando tornou-se obsoleto, já que o interpretador trata de apagar o resultado automaticamente em caso de não ser mais utilizado.

Número de linhas

```
int pg_numrows(int result);
```

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

Utilizando os resultados

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

15. Acessando o PostgreSQL via PHP

Estabelecendo conexões

Para acessar bases de dados num servidor Postgres, é necessário antes estabelecer uma conexão. Para isso, deve ser utilizado o comando `pg_connect`, ou o `pg_pconnect`. A diferença entre os dois comandos é que o pg_pconnect estabelece uma conexão permanente, ou seja, que não é encerrada ao final da execução do script. As assinaturas dos dois comandos são semelhantes, como pode ser verificado a seguir:

```
int pg_connect(string host, string porta, string opcoes, string tty, string db);
int pg_pconnect(string host, string porta, string opcoes, string tty, string db);
```

O valor de retorno é um inteiro que identifica a conexão, ou falso se a conexão falhar. Uma conexão estabelecida com o comando `pg_connect` é encerrada ao final da execução do script. Para encerrá-la antes disso deve ser utilizado o comando `pg_close`, que tem a seguinte assinatura:

```
int pg_close(int identificador da conexão );
```

IMPORTANTE: o comando `pg_close` não encerra conexões estabelecidas com o comando `pg_pconnect`.

Os comandos pg_connect e pg_pconnect também podem ser utilizados da seguinte forma:

```
pg_connect("dbname=db port=n host=”localhost” tty=”tty” options=opcoes user=usuario password=senha");
```

Realizando consultas

Para executar consultas SQL no Postgres, utiliza-se o comando `pg_exec`, que tem a seguinte assinatura:

```
int pg_exec(int conexao, string query );
```

Onde query é a expressão SQL a ser executada, sem o ponto-e-vírgula no final, e conexao é o identificador da conexão a ser utilizada. A consulta será executada na base de dados selecionada quando for efetuada a conexão com o banco. Para saber

alterar a posição indicada por esse ponteiro deve ser utilizada a função `mysql_data_seek`, sendo que o número da primeira linha de um resultado é zero.

Utilizando os resultados

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

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

Retorna o conteúdo de uma célula da tabela de resultados.

`result` é o identificador do resultado;

`linha` é o número da linha, iniciado por 0;

`campo` é uma string com o nome do campo, ou um número correspondente ao número da coluna. Se foi utilizado um alias na consulta, este deve ser utilizado no comando mysql_result.

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

```
array mysql_fetch_array(int result);
```

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

```
array mysql_fetch_row(int result);
```

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

Alterando o ponteiro de um resultado

```
int mysql_data_seek(int result, int numero);
```

Cada resultado possui um “ponteiro”, que indica qual será a próxima linha lida com o comando `mysql_fetch_row` (ou `mysql_fetch_array`). Para

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

Realizando consultas

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

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

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

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

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

Apagando o resultado

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

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

Número de linhas

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

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

14. Acessando o mySQL via PHP

Estabelecendo conexões

Para acessar bases de dados num servidor mySQL, é necessário antes estabelecer uma conexão. Para isso, deve ser utilizado o comando `mysql_connect`, ou o `mysql_pconnect`. A diferença entre os dois comandos é que o mysql_pconnect estabelece uma conexão permanente, ou seja, que não é encerrada ao final da execução do script. As assinaturas dos dois comandos são semelhantes, como pode ser verificado a seguir:

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

O valor de retorno é um inteiro que identifica a conexão, ou falso se a conexão falhar. Antes de tentar estabelecer uma conexão, o interpretador PHP verifica se já existe uma conexão estabelecida com o mesmo host, o mesmo login e a mesma senha. Se existir, o identificador desta conexão é retornado. Senão, uma nova conexão é criada.

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

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

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

IMPORTANTE: o comando `mysql_close` não encerra conexões estabelecidas com o comando `mysql_pconnect`.

Selecionando a base de dados

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

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

O DELETE exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para Null.

Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério e então, execute a consulta exclusão. Mantenha os backups de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo.

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

Comando DELETE

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

Sintaxe

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

onde:

tabela.* - O nome opcional da tabela da qual os registros são excluídos.

tabela - O nome da tabela da qual os registros são excluídos.

critério - Uma expressão que determina qual registro deve ser excluído.

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução DROP.

Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Você pode usar DELETE para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta. Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

Comando INSERT

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

Sintaxe básica

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

A instrução INSERT INTO tem as partes abaixo:

Destino- O nome da tabela ou consulta em que os registros devem ser anexados.

campo1, campo2 - Os nomes dos campos aos quais os dados devem ser anexados

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

Comando UPDATE

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

Sintaxe:

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

Onde:

Tabela - O nome da tabela cujos os dados você quer modificar.

Valornovo - Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.

critério - Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizados.

Comando Alter

Este comando permite inserir/eliminar atributos nas tabelas já existentes.

Comando:

```
ALTER  TABLE  < nome_tabela >  ADD / DROP (
    nome_atributo1  < tipo >   [ NOT NULL ],
    nome_atributoN  < tipo >   [ NOT NULL ]
) ;
```

Manipulando dados das tabelas

Comando SELECT

Permite recuperar informações existentes nas tabelas.

Sintaxe básica:

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

onde:

DISTINCT : Para eliminar linhas duplicadas na saída.

Expressao: Define os dados que queremos na saída, normalmente uma ou mais colunas de uma tabela da lista FROM.

AS nom-atributo : um *alias* para o nome da coluna, exemplo:

FROM : lista das tabelas na entrada

WHERE : critérios da seleção

ORDER BY : Critério de ordenação das tabelas de saída. ASC ordem ascendente, DESC ordem descendente

Exemplo:

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

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

Estrutura das tabelas

Comando Create

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

Sintaxe:

```
CREATE DATABASE < nome_db >;
```

onde:

nome_db - indica o nome do Banco de Dados a ser criado.

Sintaxe:

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

onde:

nome_table - indica o nome da tabela a ser criada.

nome_atributo - indica o nome do campo a ser criado na tabela.

tipo - indica a definição do tipo de atributo (integer(n), char(n), ...).

Comando Drop

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

Sintaxe:

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

13. Noções de SQL

Introdução

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM, desenvolveu a SQL como forma de interface para o sistema de BD relacional denominado SYSTEM R, início dos anos 70. Em 1986 o American National Standard Institute (ANSI), publicou um padrão SQL.

A SQL estabeleceu-se como linguagem padrão de Banco de Dados Relacional.

SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (Data Definition Language), composta entre outros pelos comandos Create, que é destinado a criação do Banco de Dados, das Tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop.

Os comandos da série DML (Data Manipulation Language), destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos Select, Insert, Update e Delete.

Uma subclasse de comandos DML, a DCL (Data Control Language), dispõe de comandos de controle como Grant e Revoke.

A Linguagem SQL tem como grandes virtudes sua capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem SQL é a capacidade que dispomos de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma seqüência de atualizações. Os comandos Commit e Rollback são responsáveis por estas facilidades.

```
class conta {
   var $saldo;

   function conta () {
     $this.saldo = 0;
   }

   function saldo() {
     return $this->saldo;
   }
   function credito($valor) {
     $this->saldo += $valor;
   }
}
```

Podemos perceber que a classe conta agora possui um construtor, que inicializa a variável `$saldo` com o valor 0.

Um construtor pode conter argumentos, que são opcionais, o que torna esta ferramenta mais poderosa. No exemplo acima, o construtor da classe conta pode receber como argumento um valor, que seria o valor inicial da conta.

Vale observar que para classes derivadas, o construtor da classe pai não é automaticamente herdado quando o construtor da classe derivada é chamado.

Como exemplo da utilização de classes e objetos, podemos utilizar a classe conta, que define uma conta bancária bastante simples, com funções para ver saldo e fazer um crédito.

```
class conta {
   var $saldo;
   function saldo() {
     return $this->saldo;
   }
   function credito($valor) {
     $this->saldo += $valor;
   }
}

$minhaconta = new conta;
$minhaconta->saldo();     // a variavel interna não foi
                          // inicializada, e não contém
                          // valor algum
$minhaconta->credito(50);
$minhaconta->saldo(); // retorna 50
```

SubClasses

Uma classe pode ser uma extensão de outra. Isso significa que ela herdará todas as variáveis e funções da outra classe, e ainda terá as que forem adicionadas pelo programador. Em PHP não é permitido utilizar herança múltipla, ou seja, uma classe pode ser extensão de apenas uma outra.Para criar uma classe extendida, ou derivada de outra, deve ser utilizada a palavra reservada `extends`, como pode ser visto no exemplo seguinte:

```
class novaconta extends conta {
  var $num;
  function numero() {
     return $this->num;
  }
}
```

A classe acima é derivada da classe conta, tendo as mesmas funções e variáveis, com a adição da variável `$numero` e a função `numero()`.

Construtores

Um construtor é uma função definida na classe que é automaticamente chamada no momento em que a classe é instanciada (através do operador `new`). O construtor deve ter o mesmo nome que a classe a que pertence. Veja o exemplo:

12. Classes e Objetos

Classe

Uma classe é um conjunto de variáveis e funções relacionadas a essas variáveis. Uma vantagem da utilização de programação orientada a objetos é poder usufruir do recurso de encapsulamento de informação. Com o encapsulamento o usuário de uma classe não precisa saber como ela é implementada, bastando para a utilização conhecer a interface, ou seja, as funções disponíveis. Uma classe é um tipo, e portanto não pode ser atribuída a uma variável. Para definir uma classe, deve-se utilizar a seguinte sintaxe:

```
class Nome_da_classe {
   var $variavel1;
   var $variavel2;
   function funcao1 ($parametro) {
    /* === corpo da função === */
   }
}
```

Objeto

Como foi dito anteriormente, classes são tipos, e não podem ser atribuídas a variáveis. Variáveis do tipo de uma classe são chamadas de objetos, e devem ser criadas utilizando o operador `new`, seguindo o exemplo abaixo:

```
$variavel = new $nome_da_classe;
```

Para utilizar as funções definidas na classe, deve ser utilizado o operador `"->"`, como no exemplo:

```
$variavel->funcao1(
```

A variável $this

Na definição de uma classe, pode-se utilizar a variável `$this`, que é o próprio objeto. Assim, quando uma classe é instanciada em um objeto, e uma função desse objeto na definição da classe utiliza a variável `$this`, essa variável significa o objeto que estamos utilizando.

assim como configurações da máquina, sistema operacional, servidor http e versão do PHP instalada.

Definindo constantes

Para definir constantes utiliza-se a função `define`. Uma vez definido, o valor de uma constante não poderá mais ser alterado. Uma constante só pode conter valores escalares, ou seja, não pode conter nem um array nem um objeto. A assinatura da função `define` é a seguinte:

```
int define(string nome_da_constante, mixed valor);
```

A função retorna `true` se for bem-sucedida. Veja um exemplo de sua utilização a seguir:

```
define ("pi", 3.1415926536);
$circunf = 2*pi*$raio;
```

variável, será criada uma nova variável de mesmo nome e de conteúdo vazio, a não ser que a chamada seja pela função `isset`. Se a operação for bem sucedida, retorna `true`.

Verificando se uma variável possui um valor

Existem dois tipos de teste que podem ser feitos para verificar se uma variável está setada: com a função `isset` e com a função `empty`.

A função isset

Possui o seguinte protótipo:

```
int isset(mixed var);
```

E retorna `true` se a variável estiver setada (ainda que com uma string vazia ou o valor zero), e `false` em caso contrário.

A função empty

Possui a seguinte assinatura:

```
int empty(mixed var);
```

E retorna `true` se a variável não contiver um valor (não estiver setada) ou possuir valor 0 (zero) ou uma string vazia. Caso contrário, retorna `false`.

Constantes pré-definidas

O PHP possui algumas constantes pré-definidas, indicando a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução, e diversas outras informações. Para ter acesso a todas as constantes pré-definidas, pode-se utilizar a função `phpinfo()`, que exibe uma tabela contendo todas as constantes pré-definidas,

Função que retorna o tipo da variável

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

```
string gettype(mixed var);
```

A palavra "mixed" indica que a variável var pode ser de diversos tipos.

A função `gettype` pode retornar as seguintes strings: `"integer"`, `"double"`, `"string"`, `"array"`, `"object"` e `"unknown type"`.

Funções que testam o tipo da variável

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

```
int is_integer(mixed var);
```

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

Destruindo uma variável

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

```
int unset(mixed var);
```

A função destrói a variável, ou seja, libera a memória ocupada por ela, fazendo com que ela deixe de existir. Se mais na frente for feita uma chamada á

Uma boa técnica de programação é utilizar a notação de arrays para nomes de cookies ou itens de um formulário html. Para um conjunto de checkboxes, por exemplo, podemos utilizar a seguinte notação:

```
<input type="checkbox" name="teste[]" value="valor1">opcao1
<input type="checkbox" name="teste[]" value="valor2">opcao2
<input type="checkbox" name="teste[]" value="valor3">opcao3
<input type="checkbox" name="teste[]" value="valor4">opcao4
<input type="checkbox" name="teste[]" value="valor5">opcao5
```

Ao submeter o formulário, o script que recebe os valores submetidos terá uma variável chamada `$teste` contendo os valores marcados num array, com índices a partir de zero. Assim, se forem marcadas as opçoes 2, 3 e 5, poderemos fazer as seguintes afirmações:

```
$teste == array(“valor2”, “valor3”, valor5”);
$teste[0] == “valor2;
$teste[1] == “valor3;
$teste[2] == “valor5;
```

O mesmo artifício pode ser utilizado com outros elementos de formulários e até com cookies.

Variáveis de ambiente

O PHP possui diversas variáveis de ambiente, como a `$PHP_SELF`, por exemplo, que contém o nome e o path do próprio arquivo. Algumas outras contém informações sobre o navegador do usuário, o servidor http, a versão do PHP e diversas informações. Para ter uma listagem de todas as variáveis e constantes de ambiente e seus respectivos conteúdos, deve-se utilizar a função `phpinfo()`.

Verificando o tipo de uma variável

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

“teste”. Note que o conteúdo da variável está no formato `urlencode`. Os formulários html já enviam informações automaticamente nesse formato, e o PHP decodifica sem necessitar de tratamento pelo programador.

URLencode

O formato `urlencode` é obtido substituindo os espaços pelo caracter “+” e todos os outros caracteres não alfa-numéricos (com exceção de “_”) pelo caracter “%” seguido do código ASCII em hexadecimal.

Por exemplo: o texto “Testando 1 2 3 !!” em urlencode fica “Testando+1+2+3+%21%21”

O PHP possui duas funções para tratar com texto em urlencode. Seguem suas sintaxes:

```
string urlencode(string texto);
string urldecode(string texto);
```

Essas funções servem respectivamente para codificar ou decodificar um texto passado como argumento. Para entender melhor o que é um argumento e como funciona uma função, leia o tópico “funções”.

Utilizando arrays

Cada elemento de um formulário HTML submetido a um script PHP cria no ambiente do mesmo uma variável cujo nome é o mesmo nome do elemento. Por exemplo: um campo definido como:

```
<input type=”text” name=”endereco”>
```

ao ser submetido a um script PHP fará com que seja criada uma variável com o nome `$endereco`. Isto acontece de forma semelhante para cookies, como veremos mais adiante.

Exemplo:

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

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

Variáveis Variáveis

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

```
$a = “teste”;
$$a = “Mauricio Vivas”;

O exemplo acima e equivalente ao seguinte:

$a = “teste”;
$teste = “Mauricio Vivas”;
```

Variáveis enviadas pelo navegador

Para interagir com a navegação feita pelo usuário, é necessário que o PHP possa enviar e receber informações para o software de navegação. A maneira de enviar informações, como já foi visto anteriormente, geralmente é através de um comando de impressão, como o *echo*. Para receber informações vindas do navegador através de um *link* ou um formulário html o PHP utiliza as informações enviadas através da URL. Por exemplo: se seu script php está localizado em `“http://localhost/teste.php3”` e você o chama com a url `“http://localhost/teste.php3?vivas=teste”`, automaticamente o PHP criará uma variável com o nome $vivas contendo a string

11. Variáveis e Constantes

Declaração de uma variável

Como a tipagem em PHP é dinâmica, as variáveis não precisam ser declaradas. Uma variável é inicializada no momento em que é feita a primeira atribuição. O tipo da variável será definido de acordo com o valor atribuído.

O modificador static

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

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

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

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

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

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

```
Exemplo:
$vivas = “Testando”;

function Teste() {
  global $vivas;
  echo $vivas;
}

Teste();
```

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

```
Exemplo:
$vivas = "Testando";

function Teste() {
  echo $GLOBALS["vivas"]; // imprime $vivas
  echo $vivas; // não imprime nada
}

Teste();
```

```
      /* A função não vai funcionar da maneira esperada,
ocorrendo um erro no interpretador. A declaração correta é: */

      function teste2($cor, $figura = circulo) {
        echo “a figura é um “, $figura, “ de cor “ $cor;
      }

      teste2(azul);

      /* Aqui a funcao funciona da maneira esperada, ou seja,
imprime o texto: “a figura é um círculo de cor azul” */
```

Contexto

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

Escopo

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

```
      Exemplo:
      $vivas = “Testando”;

      function Teste() {
        echo $vivas;
      }

      Teste();
```

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

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

```
function mais5(&$num1, $num2) {
   $num1 += 5;
   $num2 += 5;
}

$a = $b = 1;
mais5($a, $b); /* Neste caso, só $num1 terá seu valor alterado, pois a passagem por referência está definida na declaração da função. */


mais5($a, &$b); /* Aqui as duas variáveis terão seus valores alterados. */
```

Argumentos com valores pré-definidos (default)

Em PHP é possível ter valores *default* para argumentos de funções, ou seja, valores que serão assumidos em caso de nada ser passado no lugar do argumento. Quando algum parâmetro é declarado desta maneira, a passagem do mesmo na chamada da função torna-se opcional.

```
function teste($vivas = "testando") {
   echo $vivas;
}

teste(); // imprime "testando"
teste("outro teste"); // imprime "outro teste"
```

É bom lembrar que quando a função tem mais de um parâmetro, o que tem valor *default* deve ser declarado por último:

```
function teste($figura = circulo, $cor) {
  echo "a figura é um ", $figura, " de cor " $cor;
}

teste(azul);
```

Argumentos

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

```
Exemplo:

function imprime($texto){
   echo $texto;
}

imprime(“teste de funções”);
```

Passagem de parâmetros por referência

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

```
Exemplo:

function mais5($numero) {
   $numero += 5;
}

$a = 3;
mais5($a); //$a continua valendo 3
```

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

10. Funções

Definindo funções

A sintaxe básica para definir uma função é:

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

Qualquer código PHP válido pode estar contido no interior de uma função. Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser declarado, sendo necessário que o programador esteja atento para que a função retorne o tipo desejado. É recomendável que esteja tudo bem documentado para facilitar a leitura e compreensão do código. Para efeito de documentação, utiliza-se o seguinte formato de declaração de função:

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

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

Valor de retorno

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

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

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

```
<inicializacao>
while (<condicao>) {
comandos
...
<incremento>
}
```

Quebra de fluxo

Break

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

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

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

Continue

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

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

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

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

for

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

```
for (<inicializacao>;<condicao>;<incremento>)
   <comando>;
```

```
for (<inicializacao>;<condicao>;<incremento>) :
   <comando>;
   . . .
   <comando>;
endfor;
```

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

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

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

Incremento: Comando executado ao final de cada execução do laço.

Um comando `for` funciona de maneira semelhante a um `while` escrito da seguinte forma:

comandos de repetição

while

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

```
while (<expressao>)
   <comando>;
```

```
while (<expressao>):
   <comando>;
    . . .
   <comando>;
endwhile;
```

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

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

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

do... while

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

```
do {
   <comando>
   . . .
   <comando>
} while (<expressao>);
```

momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outro teste, até o fim do bloco. por isso usa-se o comando `break`, quebrando o fluxo e fazendo com que o código seja executado da maneira desejada. Veremos mais sobre o `break` mais adiante. Veja o exemplo:

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

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

Em outras linguagens que implementam o comando switch, ou similar, os valores a serem testados só podem ser do tipo inteiro. Em PHP é permitido usar valores do tipo string como elementos de teste do comando switch. O exemplo abaixo funciona perfeitamente:

```
switch ($s) {
case “casa”:
     print “A casa é amarela”;
case “arvore”:
     print “a árvore é bonita”;
case “lampada”:
     print “joao apagou a lampada”;
    }
```

```
if (expressao1) :
  comando;
   . . .
  comando;
[ elseif (expressao2)
  comando;
   . . .
  comando; ]
[ else
  comando;
   . . .
  comando; ]
endif;
```

switch

O comando `switch` atua de maneira semelhante a uma série de comandos `if` na mesma expressão. Frequentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável. Quando isso for necessário, deve-se usar o comando `switch`. O exemplo seguinte mostra dois trechos de código que fazem a mesma coisa, sendo que o primeiro utiliza uma série de `if`'s e o segundo utiliza `switch`:

```
if ($i == 0)
   print “i é igual a zero”;
elseif ($i == 1)
   print “i é igual a um”;
elseif ($i == 2)
   print “i é igual a dois”;

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

É importante compreender o funcionamento do `switch` para não cometer enganos. O comando `switch` testa linha a linha os cases encontrados, e a partir do

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

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

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

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

foi criado o comando, também opcional `elseif`. Ele tem a mesma função de um `else` e um `if` usados sequencialmente, como no exemplo acima. Num mesmo `if` podem ser utilizados diversos `elseif`'s, ficando essa utilização a critério do programador, que deve zelar pela legibilidade de seu script.

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

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

if

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

```
if (expressão)
   comando;
```

```
if (expressão):
   comando;
   . . .
   comando;
endif;
```

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

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

```
if (expressão)
   comando;
else
   comando;
```

```
if (expressão):
   comando;
   . . .
   comando;
else
   comando;
   . . .
   comando;
endif;
```

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

09. Estruturas de Controle

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

Blocos

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

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

Para que comando2 esteja relacionado ao if é preciso utilizar um bloco:

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

Comandos de seleção

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

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

```
Exemplos:

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

Existem dois operadores para “e” e para “ou porque eles têm diferentes posições na ordem de precedência.

Comparação

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

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

Expressão condicional

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

```
(expressao1)?(expressao2):( expressao3)
```

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

de incremento e decremento

++	incremento
--	decremento

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

```
Exemplo:

$a = 7;
$a += 2; // $a passa a conter o valor 9
```

bit a bit

Comparam dois números bit a bit.

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

Lógicos

Utilizados para inteiros representando valores booleanos

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

08. Operadores

Aritméticos

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

+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo

de strings

Só há um operador exclusivo para strings:

.	concatenação

de atribuição

Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído. No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro. A atribuição é sempre por valor, e não por referência.

```
Exemplo:
$vivas = 15;              // $vivas é integer (15)
$vivas = (double) $vivas  // $vivas é double (15.0)
$vivas = 3.9              // $vivas é double (3.9)
$vivas = (int) $vivas     // $vivas é integer (3)
                          // o valor decimal é truncado
```

Os tipos de *cast* permitidos são:

(int), (integer) ⇒ muda para integer;

(real), (double), (float) ⇒ muda para float;

(string) ⇒ muda para string;

(array) ⇒ muda para array;

(object) ⇒ muda para objeto.

Com a função settype

A função `settype` converte uma variável para o tipo especificado, que pode ser “integer”, “double”, “string”, “array” ou “object”.

```
Exemplo:
$vivas = 15;              // $vivas é integer
settype($vivas,double)    // $vivas é double
```

O tipo para o qual os valores dos operandos serão convertidos é determinado da seguinte forma: Se um dos operandos for `float`, o outro será convertido para `float`, senão, se um deles for `integer`, o outro será convertido para `integer`.

```
Exemplo:
$vivas = “1”;        // $vivas é a string “1”
$vivas = $vivas + 1;// $vivas é o integer 2
$vivas = $vivas + 3.7;// $vivas é o double 5.7
$vivas = 1 + 1.5          // $vivas é o double 2.5
```

Como podemos notar, o PHP converte `string` para `integer` ou `double` mantendo o valor. O sistema utilizado pelo PHP para converter de *strings* para números é o seguinte:

- É analisado o início da `string`. Se contiver um número, ele será avaliado. Senão, o valor será 0 (zero);
- O número pode conter um sinal no início (“+” ou “-“);
- Se a `string` contiver um ponto em sua parte numérica a ser analisada, ele será considerado, e o valor obtido será `double`;
- Se a `string` contiver um “e” ou “E” em sua parte numérica a ser analisada, o valor seguinte será considerado como expoente da base 10, e o valor obtido será `double`;

```
Exemplos:
$vivas = 1 + “10.5”;      // $vivas == 11.5
$vivas = 1 + “-1.3e3”;    // $vivas == -1299
$vivas = 1 + “teste10.5”; // $vivas == 1
$vivas = 1 + “10testes”;  // $vivas == 11
$vivas = 1 + "  10testes";     // $vivas == 11
$vivas = 1 + "+  10testes";    // $vivas == 1
```

Transformação explícita de tipos

A sintaxe do *typecast* de PHP é semelhante ao C: basta escrever o tipo entre parenteses antes do valor

No apêndice 02 está disponível uma lista das funções mais comuns para o tratamento de arrays.

Objetos

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

```
Exemplo:
class teste {
          function nada() {
               echo “nada”;
          }
}

$vivas = new teste;
$vivas -> nada();
```

A utilização de objetos será mais detalhada mais à frente.

Booleanos

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

Transformação de tipos

A transformação de tipos em PHP pode ser feita das seguintes maneiras:

Coerções

Quando ocorrem determinadas operações (“+”, por exemplo) entre dois valores de tipos diferentes, o PHP converte o valor de um deles automaticamente (coerção). É interessante notar que se o operando for uma variável, seu valor não será alterado.

Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas. Através de listas é possível atribuir valores que estão num array para variáveis. Vejamos o exemplo:

```
Exemplo:

 list($a, $b, $c) = array(“a”, “b”, “c”);
```

O comando acima atribui valores às três variáveis simultaneamente. É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos. No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero. Um fator importante é que cada variável da lista possui um índice inteiro e ordinal, iniciando com zero, que serve para determinar qual valor será atribuído. No exemplo anterior temos $a com índice 0, $b com índice 1 e $c com índice 2. Vejamos um outro exemplo:

```
$arr = array(1=>”um”,3=>”tres”,”a”=>”letraA”,2=>”dois);
list($a,$b,$c,$d) = $arr;
```

Após a execução do código acima temos os seguintes valores:

```
$a == null
$b == “um”
$c == “dois”
$d == “tres”
```

Devemos observar que à variável $a não foi atribuído valor, pois no array não existe elemento com índice 0 (zero). Outro detalhe importante é que o valor “tres” foi atribuído à variável $d, e não a $b, pois seu índice é 3, o mesmo que $d na lista. Por fim, vemos que o valor “letraA” não foi atribuído a elemento algum da lista pois seu índice não é inteiro.

Os índices da lista servem apenas como referência ao interpretador PHP para realizar as atribuições, não podendo ser acessados de maneira alguma pelo programador. De maneira diferente do array, uma lista não pode ser atribuída a uma variável, servindo apenas para fazer múltiplas atribuições através de um array.

A tabela seguinte lista os caracteres de escape:

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

No apêndice 01 está disponível uma lista das funções utilizadas no tratamento de strings.

Arrays

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

Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

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

Equivalentemente, pode-se escrever:

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

Números em Ponto Flutuante (double ou float)

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

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

Strings

Strings podem ser atribuídas de duas maneiras:

a) utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \' – ver tabela abaixo)
b) utilizando aspas duplas (") – Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

Exemplo:

```
<?
$teste = "Mauricio";
$vivas = '---$teste--\n';
echo "$vivas";
?>
```

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

```
<?
$teste = "Mauricio";
$vivas = "---$teste---\n";
echo "$vivas";
?>
```

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

07. Tipos

Tipos Suportados

PHP suporta os seguintes tipos de dados:

- Inteiro
- Ponto flutuante
- String
- Array
- Objeto

PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução.

Ainda assim, é permitido converter os valores de um tipo para outro desejado, utilizando o *typecasting* ou a função `settype` (ver adiante).

Inteiros (integer ou long)

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

```
$vivas = 1234; # inteiro positivo na base decimal
$vivas = -234; # inteiro negativo na base decimal
$vivas = 0234; # inteiro na base octal-simbolizado pelo 0
               # equivale a 156 decimal
$vivas = 0x34; # inteiro na base hexadecimal(simbolizado
               # pelo 0x) – equivale a 52 decimal.
```

A diferença entre inteiros simples e `long` está no número de bytes utilizados para armazenar a variável. Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

```
Exemplos:

<?
  echo “teste”; /* Isto é um comentário com mais
de uma linha, mas não funciona corretamente ?>
*/

<?
  echo “teste”; /* Isto é um comentário com mais
de uma linha que funciona corretamente
*/
?>
```

Imprimindo código html

Um script php geralmente tem como resultado uma página html, ou algum outro texto. Para gerar esse resultado, deve ser utilizada uma das funções de impressão, `echo` e `print`. Para utilizá-las deve-se utilizar um dos seguintes formatos:

```
print(argumento);
echo (argumento1, argumento2, ... );
echo argumento;
```

Nomes de variáveis

Toda variável em PHP tem seu nome composto pelo caracter $ e uma string, que deve iniciar por uma letra ou o caracter "_". **PHP é case sensitive**, ou seja, as variáveis $vivas e $VIVAS são diferentes. Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui alguma variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

Comentários

Há dois tipos de comentários em código PHP:

Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de código PHP – o que vier antes. Pode ser delimitado pelo caracter "#" ou por duas barras (//).

```
Exemplo:
<?  echo "teste"; #isto é um teste  ?>
<?  echo "teste"; //este teste é similar ao anterior ?>
```

Comentários de mais de uma linha:

Tem como delimitadores os caracteres `"/*"` para o início do bloco e `"*/"` para o final do comentário. Se o delimitador de final de código PHP (`?>`) estiver dentro de um comentário, não será reconhecido pelo interpretador.

06. Sintaxe Básica

Delimitando o código PHP

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

```
<?php
comandos
?>

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

<?
comandos
?>

<%
comandos
%>
```

O tipo de *tags* mais utilizado é o terceiro, que consiste em uma “abreviação” do primeiro. Para utilizá-lo, é necessário habilitar a opção *short-tags* na configuração do PHP. O último tipo serve para facilitar o uso por programadores acostumados à sintaxe de ASP. Para utilizá-lo também é necessário habilitá-lo no PHP, através do arquivo de configuração *php.ini.*

Separador de instruções

Entre cada instrução em PHP é preciso utilizar o ponto-e-vírgula, assim como em C, Perl e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

05. Instalação e configuração em ambiente linux RedHat

(Disponível na próxima versão)

As configurações necessárias são relativas a segurança, e exigem um conhecimento mais avançado de administração de servidores. Como essa instalação destina-se apenas a praticar o PHP, não é necessário fazer muitas alterações na segurança, bastando apenas saber como adicionar usuários.

Para isto, basta utilizar o comando GRANT, que tem a seguinte sintaxe:

```
GRANT privilegio [(lista_colunas)]
      [, privilegio [(colunas)] ...]
 ON {tabela | * | *.* | db.*}
 TO usuario [IDENTIFIED BY 'senha']
    [, usuario [IDENTIFIED BY 'senha'] ...]
 [WITH GRANT OPTION]
```

Onde privilégio é uma das palavras reservadas listadas a seguir:

ALL PRIVILEGES	INDEX	SHUTDOWN	DROP
FILE	SELECT	DELETE	REFERENCES
RELOAD	CREATE	PROCESS	USAGE
ALTER	INSERT	UPDATE	

Cada palavra representa um tipo de acesso à(s) coluna(s), tabela(s) ou base(s) de dados listadas logo depois da cláusula ON.

Usuário deve conter o nome do usuário (login) e o host permitido (ex.: `teste@localhost`).

Abaixo temos um exemplo da utilização do comando grant:

```
GRANT SELECT, INSERT, UPDATE ON *
TO vivas@localhost IDENTIFIED BY "senhateste";
```

O exemplo cria o usuário "vivas", com a senha "senhateste", que só pode acessar da mesma máquina onde está o servidor (localhost), e só pode utilizar os comandos select, insert e update.

Também é possível adicionar usuários utilizando o comando INSERT, pra alterar diretamente na tabela de privilégios, que é a tabela "user" da base de dados "mysql", que possui os campos para definir nome de usuário, host, senha, e permissões específicas.

copiado, o arquivo pode ser editado, bastando modificar a linha `extension_dir`, que deve conter o diretório onde estão os módulos (c:\php3). Veja o exemplo:

```
extension_dir  =   c:\php3
```

Além disso é necessário descomentar a linha referente o módulo mysql, já que iremos utilizá-lo basta tirar o ";" (ponto-e-vírgula) do início da linha:

```
;extension=php3_mysql.dll
```

Feito isso, podemos partir para a configuração do servidor Apache, necessária para que este reconheça o PHP. Editando novamente o arquivo httpd.conf, as linhas a seguir devem ser adicionadas no final do arquivo:

```
ScriptAlias /php3/ "c:/php3/"
AddType application/x-httpd-php3 .php3 .php
Action application/x-httpd-php3 "/php3/php.exe"
```

A primeira linha define o diretório onde está o PHP. A segunda cria um "tipo" para o PHP, definido que todos os arquivos com as extensões ".php3" e ".php" devem passar pelo interpretador PHP. A terceira linha define o executável do interpretador PHP.

Depois de salvar o arquivo, podemos testar se a instalação do PHP foi bem sucedida. A melhor maneira é criar um arquivo chamado teste.php3 e salvar no diretório raiz do servidor Apache. O arquivo deve conter a seguinte informação:

```
<?
 phpinfo();
?>
```

Acessando a página através do servidor (http://localhost/teste.php3), devemos ter como resultado uma listagem de todas as configurações do PHP, incluindo o módulo mysql, que foi definido como ativo no arquivo php3.ini.

mySQL

O banco de dados mySQL pode ser conseguido em "http://www.mysql.com/download.html". Sua instalação também é bastante simples, também no modelos de instalação de qualquer aplicativo para Windows.

04. Instalação e configuração em ambiente windows

Servidor Apache

O servidor http que será utilizado neste curso é o Apache, que está disponível para download em "http://www.apache.org/httpd.html". A instalação do Apache é bastante simples, similar a qualquer aplicação windows. A única restrição é que o winsock2 deve estar instalado no sistema. Se não estiver, o download pode ser feito em:

http://www.microsoft.com/windows95/downloads/contents/wuadmintools/s_wunetworkingtools/w95sockets2/

Depois de instalado, é necessário fazer a configuração do servidor, através do arquivo httpd.conf. Todas as configurações estão comentadas. O mínimo que deve ser configurado é o diretório onde os documentos estarão, através da opção `DocumentRoot`. Basta procurar a opção e escrever o nome do diretório em seguida, como no exemplo:

```
DocumentRoot "C:\vivas\"
```

Uma outra configuração básica é a `DirectoryIndex`, que informa ao servidor quais os arquivos serão exibidos automaticamente como índice do diretório. É isso que faz com que ao digitar, por exemplo, "www.guia-aju.com.br", o servidor saiba qual dos arquivos do diretório deve ser exibido. Abaixo temos um exemplo da utilização do `DirectoryIndex`:

```
DirectoryIndex index.html index.htm index.php3
```

Feito isso, crie um arquivo com um dos nomes definidos como índice e coloque no diretório definido como root. Execute o servidor Apache e tente acessar o endereço "http://localhost" pelo browser. Se a página for exibida, é porque o servidor foi instalado corretamente.

PHP

O PHP pode ser conseguido em "www.php.net", e sua instalação também é simples. Basta descompactar os arquivos para o diretório "c:\php3" e editar o arquivo de configuração. O arquivo "php3.ini-dist" deve ser copiado para o diretório do windows (geralmente c:\windows ou c:\winnt) com o nome php3.ini. Depois de

Exibe na tela do browser um campo de texto e um botão, que ao clicado abre uma janela para localizar um arquivo no disco. Para utilizar este tipo de componente, o formulário deverá utilizar o método “POST” e ter o parâmetro “enctype” com o valor `"multipart/form-data".`

Parâmetros:

Size – O tamanho do campo de texto exibido.

Wrap – Maneira como são tratadas as quebras de linha automáticas. O valor soft faz com que o texto “quebre” somente na tela, sendo enviado para o servidor o texto da maneira como foi digitado; O valor “hard” faz com que seja enviado para o servidor da maneira como o texto aparece na tela, com todas as quebras de linhas inseridas automaticamente; o valor “off” faz com que o texto não quebre na tela e nem quando enviado ao servidor.

Value – O elemento do tipo textarea não possui o parâmetro “value”. O valor pré-definido do campo é o texto que fica entre as tags <textarea> e </textarea>.

Select

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

Se o parâmetro “size” tiver o valor 1 e não houver o parâmetro “multiple”, exibe na tela uma “combo box”. Caso contrário, exibe na tela uma “select list”.

Parâmetros:

Size – número de linhas exibidas. Default: 1;

Multiple – parâmetro que, se presente, permite que sejam selecionadas duas ou mais linhas, através das teclas Control ou Shift;

option – Cada item do tipo “option” acrescenta uma linha ao select;

value – Valor a ser enviado ao servidor se aquele elemento for selecionado. Default: o texto do item;

text – valor a ser exibido para aquele item. Não é definido por um parâmetro, mas pelo texto que fica entre as tags <option> e </option>

Upload de arquivos

```
<input type="file" name="" size="">
```

Parâmetros:

Value – o texto que aparecerá no corpo do botão.

Reset Button

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

Utilizado para fazer todos os campos do formulário retornem ao valor original, quando a página foi carregada. Bastante utilizado como botão “limpar”, mas na realidade só limpa os campos se todos eles têm como valor uma string vazia.

Parâmetros:

Value – o texto que aparecerá no corpo do botão.

Button

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

Utilizado normalmente para ativar funções de scripts client-side (JavaScript, por exemplo). Sem essa utilização, não produz efeito algum

Parâmetros:

Value – o texto que aparecerá no corpo do botão.

TextArea

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

Exibe na tela uma caixa de texto, com o tamanho definido pelos parâmetros “cols” e “rows”.

Parâmetros:

Cols – número de colunas do campo, em caracteres;

Rows– número de linhas do campo, em caracteres;

Checkbox

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

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

Parâmetros:

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

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

Radio Button

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

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

Parâmetros:

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

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

Submit Button

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

Utilizado para enviar os dados do formulário para o script descrito na seção “action” da definição do formulário

parâmetros em comum: type, que define o tipo de elemento, e name, que como já foi dito define o nome daquele elemento.

Campo de Texto

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

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

Parâmetros:

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

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

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

Campo de Texto com Máscara

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

Tipo de campo semelhante ao anterior, com a diferença que neste caso os dados digitados são substituídos por asteriscos, e por isso são os mais recomendados para campos que devam conter senhas. É importante salientar que nenhuma criptografia é utilizada. Apenas não aparece na tela o que está sendo digitado.

Parâmetros:

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

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

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

03. Formulários HTML

Definindo um formulário

Por ser uma linguagem de marcação, a sintaxe do HTML na maioria dos casos exige uma “tag” de início e uma de final daquele bloco. É Exatamente isso que ocorre com a definição de um formulário: uma tag no início e outra no final, sendo que todos os elementos do formulário devem estar entre as duas tags. Isto torna possível a inclusão de mais de um formulário num mesmo html. As tags citadas são:

```
<form name=”” action=”” method=”” enctype=””>
```

Onde temos:

`name`: o identificador do formulário. Utilizado principalmente em Scripts client-side (JavaScript);

`action`: nome do script que receberá os dados do formulário ao ser submetido. Mais à frente estão abordadas as maneiras de tratar esses dados recebidos;

`method`: método de envio dos dados: get ou post;

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

```
Exemplo:
<form action="exemplo.php" method="post">

(textos e elementos do form)

</form>
```

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

A tag <input>

Muitos elementos de um formulário html são definidos pela tag `<input>`. Cada tipo de elemento possui parâmetros próprios, mas todos possuem pelo menos dois

O método POST

Através da utilização de headers é possível enviar os parâmetros da URL solicitada sem expor esses dados ao usuário, e também sem haver um limite de tamanho.

Uma conexão ao servidor HTTP utilizando o método POST seria algo semelhante ao que segue:

```
telnet www.guia-aju.com.br 80
Trying 200.241.59.16...
Connected to www.guia-aju.com.br.
Escape character is '^]'.
POST /index.php3
Accept */*
Content-type: application/x-www-form-urlencoded
Content-length:22

id=0024horas&tipo=Taxi

(... página solicitada ...)
Connection closed by foreign host.
```

Devemos observar os headers enviados ao servidor: a linha “Accept” informa os tipos de dados que podem ser enviados como resposta (no caso, todos). A linha “Content-type” informa o tipo de dado que está sendo enviado (urlencoded). O terceiro header é o mais importante pois informa o tamanho do corpo da mensagem, que contém os parâmetros. Após todos os headers há um salto de linha e então é iniciado o corpo da mensagem, no formato urlencoded.

Obviamente o usuário não deve se preocupar com os headers, em codificar os dados ou em calcular o tamanho do corpo da mensagem. O browser faz isso de maneira transparente.

Utilizando GET e POST

O método GET pode ser utilizado através da digitação de um endereço no local apropriado do navegador ou através de um hiperlink, ou seja, uma referência de uma página a outra. Nesses casos é preciso converter os dados para o formato urlencode. A terceira maneira de utilizar o GET é através de formulários HTML, e neste caso o usuário não precisa se preocupar com a codificação dos dados. A utilização de formulários HTML é a única maneira possível de submeter dados pelo método POST.

Obviamente a diferença do browser é que ele trata as informações recebidas e exibe a página já formatada.

Através do método GET também é possível passar parâmetros da requisição ao servidor, que pode tratar esses valores e até alterar a resposta a depender deles, como no exemplo abaixo:

```
telnet www.guia-aju.com.br 80
Trying 200.241.59.16...
Connected to www.guia-aju.com.br.
Escape character is '^]'.
GET /index.php3?id=0024horas&tipo=Taxi
(... página solicitada ...)
Connection closed by foreign host.
```

No exemplo são passados dois parâmetros: id e tipo. Esses parâmetros estão no formato conhecido por URLencode, que é detalhado no capítulo 09.

Apesar de ser possível passar parâmetros utilizando o método GET, e com isso gerar páginas dinamicamente, este método tem pelo menos dois problemas que em determinadas circunstâncias podem ser considerados sérios:

O primeiro é que o GET permite uma quantidade de dados passados limitada a 1024 caracteres, o que pode gerar perda de informações em certos casos.

O segundo é que pelo fato de que as informações fazem parte da URL, todos os dados podem ser vistos pelo usuário. Isso pode ser extremamente perigoso quando informações sigilosas estão envolvidas (senha, por exemplo).

Headers

A versão 1.0 do protocolo HTTP trouxe boas inovações ao mesmo. Uma delas foi a criação de headers nas mensagens de requisição e de resposta. Os headers são informações trocadas entre o navegador e o servidor de maneira transparente ao usuário, e podem conter dados sobre o tipo e a versão do navegador, a página de onde partiu a requisição (link), os tipos de arquivos aceitos como resposta, e uma série de outras informações.

Assim foi possível definir um outro método de requisição de arquivos, que resolveu os principais problemas do método GET.

02. Enviando Dados para o Servidor HTTP

Programar para a web pode ser considerado como um jogo que consiste em receber os dados do usuário, processá-los e enviar a resposta dinâmica. Uma vez enviada a resposta, é encerrado o contato entre o servidor e o cliente. Portanto a primeira coisa a aprender é como fazer para receber os dados enviados pelo browser para o servidor.

O protocolo HTTP provê dois principais métodos para enviar informações para o servidor web, além da URL referente ao arquivo solicitado. Esses métodos são o POST e o GET.

O protocolo HTTP/1.0 também especifica o método HEAD, utilizado apenas para transmitir informações do header, além dos métodos PUT e DELETE, que não serão abordados neste curso.

O método GET

A especificação do protocolo HTTP/0.9 (a primeira implementação do HTTP) possuía a definição do método GET, utilizado pelo browser para solicitar um documento específico.

Por exemplo: a seguinte requisição HTTP retornaria o documento "index.html", localizado no diretório do servidro chamado “teste”:

```
GET /teste/index.html CRLF
```

Devemos notar que a requisição GET inicia com a palavra GET, inclui o documento solicitado e encerra com a combinação dos caracteres *carriage return* e *line feed*.

Para um melhor entendimento, você pode fazer uma requisição GET conectando diretamente em algum servidor WEB, conectando através de um programa de telnet (geralmente o servidor http utiliza a porta 80). A resposta será o código da página solicitada.

```
telnet www.guia-aju.com.br 80
Trying 200.241.59.16...
Connected to www.guia-aju.com.br.
Escape character is '^]'.
GET /index.php3
(... página solicitada ...)
Connection closed by foreign host.
```

Home Page Tools com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.

Estima-se que em 1996 PHP/FI estava sendo usado por cerca de 15.000 *sites* pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP. Ele deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por **Zeev Suraski** e **Andi Gutmans**, e esse novo interpretador foi a base para a versão 3.

O lançamento do PHP4, ocorrido em 22/05/2000, trouxe muitas novidades aos programadores de PHP. Uma das principais foi o suporte a sessões, bastante útil pra identificar o cliente que solicitou determinada informação. Além das mudanças referentes a sintaxe e novos recursos de programação, o PHP4 trouxe como novidade um otimizador chamado Zend, que permite a execução muito mais rápida de scripts PHP. A empresa que produz o Zend promete para este ano o lançamento de um compilador de PHP. Códigos compilados serão executados mais rapidamente, além de proteger o fonte da aplicação.

O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.

O que pode ser feito com PHP?

Basicamente, qualquer coisa que pode ser feita por algum programa CGI pode ser feita também com PHP, como coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber *cookies*.

PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros. Construir uma página baseada em um banco de dados torna-se uma tarefa extremamente simples com PHP.

Além disso, PHP tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP. Ainda é possível abrir *sockets* e interagir com outros protocolos.

Como surgiu a linguagem PHP?

A linguagem PHP foi concebida durante o outono de 1994 por **Rasmus Lerdorf**. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua *home-page* apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas. A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como "**P**ersonal **H**ome **P**age Tools" (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas *macros* e alguns utilitários que rodavam "por trás" das *home-pages*: um livro de visitas, um contador e algumas outras coisas.

Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de **PHP/FI**, o "FI" veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (**F**orm **I**nterpreter). Ele combinou os scripts do pacote *Personal*

Figura 2. Requisição Normal

Figura 3. Requisição de página dinâmica

O que é PHP?

PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro. Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

01. Introdução

Client-Side Scripts

São responsáveis pelas ações executadas no browser, sem contato com o servidor. Os exemplos mais comuns de aplicações client-side são imagens e textos que mudam com o passar do mouse.

Os scripts client-side são muito úteis para fazer validações de formulários sem utilizar processamento do servidor e sem provocar tráfego na rede. Outra utilização comum é na construção de interfaces dinâmicas e “leves”.

Figura 1. Funcionamento de scripts client-side

Server-Side Scripts

Os scripts server-side são responsáveis pela criação de páginas em tempo real. Num mecanismo de busca, por exemplo, seria inviável manter um arquivo para cada consulta a ser realizada. O que existe é um modelo da página de resposta, que é mesclado com os dados no momento em que a página é requisitada.

Notas do autor

Este documento foi criado inicialmente como parte do projeto de conclusão de curso da Universidade Federal de Sergipe, e distribuído gratuitamente através da Internet.

Depois de terminado o projeto, recebi diversas sugestões sobre conteúdo a incluir, e também passei a dar cursos de PHP em diversas instituições de Sergipe. Diante disso, continuei a escrever o documento, sendo algumas inclusões para atender às sugestões e outras para utilizar nos cursos.

Como poderá ser observado principalmente no capítulo 05, o documento não está concluído, e nem sei se algum dia estará, tendo em vista que o uso de PHP cresce cada vez mais, e ainda falta muito a ser dito sobre ele aqui.

Apesar de citar em alguns pontos o PHP4, o documento ainda é baseado em PHP3, lançado há menos de um mês. Porém todo o conteúdo que segue é compatível com PHP4, e por isso não há problema em lançar esta nova versão baseada em PHP3.

Se você tem uma página com tutoriais, ou gostou deste documento e quer publicá-lo em seu site, fique à vontade. Só peço duas coisas:

1. Me avise, informando a URL do site (só por curiosidade minha);
2. Lembre-se que o autor do documento sou eu. Apesar de ainda não ter visto, já fui informado que há cópias piratas deste documento. Mas quero lembrar que não é preciso piratear algo completamente **GRATUITO**.

Se houver alguma informação incorreta, peço que me informem por e-mail. Se tiverem dúvidas sobre temas tratados aqui, ou até sobre os ainda não presentes neste documento, entrem em contato comigo por e-mail. Para obter a versão original do documento, você pode pedir por e-mail, ou visitar o site www.vivas.com.br.

Meu e-mail? **mauricio@vivas.com.br.**

Índice

Aplicações Web com PHP

por *Maurício Vivas de Souza Barreto*

Aracaju, junho/2000