Desarrollo web con PHP

Un primer acercamiento

Imobach González Sosa

imobachgs@softhome.net

El presente documento pretende servir como guía para aquellos que deseen empezar a desarrollar con uno de los lenguajes de *scripting* más extendidos del momento: *PHP: Hypertext Preprocessor*.

1. Presentación de este tutorial

Este tutorial no tiene por objetivo se una guía exhaustiva del lenguaje PHP, sino servir como punto de partida para aquellos que quieran a empezar sus primeras líneas en uno de los lenguajs más extendidos en la web.

Pese a que PHP no está confinado sólo al desarrollo web, sí que es éste el campo en el que tiene más aceptación. No hay más que mirar la gran cantidad de software para la web escrito en este lenguaje para darse cuenta de su importancia: Sistemas de Gestión de Contenidos, galerías comerciales, páginas corporativas... poco a poco, PHP ha ido desplazando a Perl, uno de los reyes de la web, mientras sigue aguantando las embestidas del que muchos consideran su máximo rival, el ASP de Microsoft.

Durante este tutorial se intentará formar al usuario para que sea capaz de escribir sencillas aplicaciones web con este lenguaje, haciendo uso además uno de un complemento que se ha convertido en indispensable: un Sistema de Gestión de Bases de Datos (SGBD). Para ello, se comenzará estudiando los fundamentos básicos del lenguaje -siempre mirando a la web- para, en la parte final, abordar las posibilidades que ofrecen PHP y un SGBD, en este caso PostgreSQL (http://www.postgresql.org/).

De todas maneras, si se desea más información acerca de otros usos de PHP, se aconseja echar un vistazo a proyectos como PHP-GTK (http://gtk.php.net/), que permite combinar este lenguaje con la archiconocida y veterana biblioteca GTK (http://www.gtk.org/).

2. ¿Qué es PHP?

Antes de profundizar en la sintaxis y semántica de PHP, conviene explicar de qué estamos está hablando exactamente. Así que, para empezar, tratemos de definir qué es PHP.

> PHP (acrónimo recursivo de "PHP: Hypertext Preprocessor") es un lenguaje de script de propósito general concebido originalmente para el desarrollo web y que ofrece la posibilidad de mezclarlo con código HTML.

Aunque la definición es bastante clara hay algunas cosas que conviene comentar. Para empezar, ¿qué es un lenguaje de *script*? De un tiempo a esta parte se llama así a aquellos lenguajes -normalmente interpretados- que pueden utilizarse para reemplazar a los ficheros de proceso por lotes y a los *scripts*[1] de los diferentes intérpretes de comandos -bash, sh, csh, etc.- Perl es quizás el ejemplo más extendido. Y es que, aunque PHP fue pensado originalmente para el desarrollo web, su uso se ha propagado a otras áreas, existiendo incluso una extensión que permite combinar este lenguaje con la archiconocida biblioteca GTK (http://www.gtk.org/) (PHP-GTK (http://gtk.php.net/)).

El otro aspecto reseñable de la definición presentada anteriormente es la posibilidad que ofrece de PHP de embeber su código en una página HTML. Veamos un pequeño ejemplo, si bien sobre este particular se volverá a incidir con posterioridad.

Ejemplo 1. Código PHP embebido en HTML

```
<HTML>
  <HEAD><TITLE>Ejemplo: hola mundo<TITLE>
    <BODY>
      <? echo "Ejemplo: esto es código PHP" ?>
  </BODY>
</HTML>
```

Como puede verse, en medio de todas las marcas HTML se encuentra una línea de código PHP. El intérprete sustituirá dicha línea por la cadena *Ejemplo: esto es código PHP* y devolverá el resultado. Hay que señalar que esta interpretación, cuando se habla de entornos web, se lleva a cabo en el servidor, es decir, el cliente no verá nunca el código PHP utilizado para generar una página, ya que es el servidor quien se encargada de procesarlo. Se trata de un comportamiento al más puro estilo CGI.

3. Preparación del espacio de trabajo

Cuando se sigue cualquier tutorial, es buena idea tener un espacio de trabajo en el que se puedan ir probando los ejemplos y realizando nuevos experimentos. Esta sección estará dedicada a echar un rápido vistazo a lo qué necesitaremos y dónde podemos conseguirlo. Sin embargo, aquí no se dará ninguna guía acerca de cómo instalar PHP o un servidor web, ya que para eso existe la documentación adecuada en el seno de cada proyecto.

Como se menionó con anterioridad, este tutorial está enfocado al desarrollo web con PHP -aunque la mayoría de las ideas expuestas podrán aplicarse a otros campos-. Por lo tanto, esto es lo que vamos a necesitar:

- Un servidor web, como por ejemplo Apache (http://httpd.apache.org/). El proceso de instalación de un software de esta naturaleza queda totalmente fuera del alcance de este tutorial.
- Un intérprete de PHP que, obviamente, puede descargarse de la web del proyecto: http://www.php.net/.

PHP puede ser instalado de dos modos:

- Como módulo del servidor web. Según la documentación de PHP, estos son los servidores soportados: Apache, Apache 2.0, Caudium, fhttpd, IIS/PWS, Netscape, iPlanet, SunONE, OmniHTTPd Server, Oreilly Website Pro, Sambar y Xitami.
- Como CGI. De esta manera se crea un intérprete que puede usarse tanto para el procesamiento CGI como para la ejecución de *scripts* que no tengan nada que ver con la web.

Obviamente, si se va a usar PHP para escribir *scripts* fuera de entornos web, se optará por el segundo modo. Sin embargo, cuando se trate de desarrollo web se puede escoger cualquiera de las dos modalidades.

Por otro lado, si el servidor web escogido no dispone de soporte para PHP habrá que considerar usar la opción CGI.

4. El programa "Hola mundo" en PHP

El programa Hola mundo se ha convertido en todo un clásico a la hora de comenzar el aprendizaje de un nuevo lenguaje de programación y, como no, será nuestro punto de partida para comenzar a estudiar la sintaxis y semántica de PHP. Así, en esta sección nos dedicaremos al análisis de un sencillo Hola mundo para PHP, con el fin de comprender los primeros -y más sencillos- fundamentos de este lenguaje.

A continuación crearemos un archivo (con extensión php) con el siguiente contenido:

Ejemplo 2. Hola mundo

```
<HTML>
  <HEAD><TITLE>Hola mundo<TITLE>
    <BODY>
      <? echo "Hola mundo" ?>
  </BODY>
</HTML>
```

Podemos ver como el código PHP queda delimitado por las marcas <? y ?>. Sin embargo, no es éste el único modo en que puede indicarse qué porción de código ha de ser interpretada como PHP. Pueden usarse las parejas <?php ?>, <script language="php"> </script> y <% %> -ala ASP-.

El otro aspecto a tener en cuenta es la ausencia, en este caso, del punto y coma que delimitarán las líneas en PHP (;). El motivo es que cuando se trate de una sóla línea no será necesario hacer uso de este. En el siguiente caso sí que es necesario (por sencillez obviaremos el código HTML):

Ejemplo 3. Usando ";" para separar líneas

```
<?
  session_register("a");
  $a = $a + 1;
  echo "Hola mundo (por $a vez)";
?>
```

Pero volvamos al Hola mundo. Ya hemos visto cómo se delimita el código PHP dentro del HTML y cómo se separan las líneas de código. Ahora, sólo nos resta conocer cuál es el efecto del código escrito:

Ejemplo 4. Resultado del programa "Hola mundo"

```
<HTML>
  <HEAD><TITLE>Hola mundo<TITLE>
    <BODY>
    Hola mundo
  </BODY>
</HTML>
```

Como puede observarse, la sentencia `echo` simplemente "imprime" su argumento (en este caso la cadena *Hola mundo*. Hay que tener en cuenta que `echo` no es exactamente una función, sino una sentencia del lenguaje, motivo por el cuál no es necesario el uso de paréntesis como más adelante veremos que ocurre con las funciones.

Este código, interpretado en el servidor -todo el énfasis que se haga en este punto es poco-, es enviado al navegador del cliente que lo interpretará de la manera oportuna.

5. Trabajando con variables

Hasta ahora no hemos hecho más que introducirnos en el mundo de PHP. Será a partir de ahora cuando comencemos a ver ejemplos más prácticos que el Hola mundo. Como punto de partida vamos a estudiar las variables en PHP: su sintaxis, su modo de empleo, ycuriosidades como los nombres de variables y funciones variables.

5.1. Sintaxis

En PHP las variables se representan utilizando un signo de dólar ($) seguido por el nombre de la variable. Cabe destacar que los nombres de las variables en PHP son sensibles a mayúsculas, por lo que `$edad` y `$Edad` son variables diferentes. En el Ejemplo 5 `$a` y `$A` son, por lo tanto, variables diferentes, con valores 8 y 13 respectivamente.

Ejemplo 5. Asignación valores a variables

```
<?php
  /* $a y $A son distintas variables */
  $a = 8;         // asigna el valor 8 a la variable $a
  $A = 13;        // asigna el valor 13 a la variable $A
  $sum = $a + $A; // suma $a y $A y guarda el resultado en $sum
  $dif = $a - $A; // resta $
  echo "La suma de $a y $A es $sum. Su resta es $dif.";
?>
```

¿Cuál será la salida del ejemplo? Como puede observarse el código se encuentra exhaustivamente comentado, así que no debe presentar dificultad alguna que la salida será:

```
La suma de 8 y 13 es 21. Su resta es -5.
```

5.2. Paso por valor y por referencia

PHP4 ofrece dos formas de asignar valores a las variables:

Por valor

Cuando se asigna una exprexión a una variable, el valor íntegro de dicha expresión se copia en la variable de destino. Esto provoca que, por ejemplo, si se copia el valor de la variable `$A` en `$B`, la modificación de esta última no afecte a la primera.

Por referencia

En este caso, si se asigna el valor de otra variable `$A` a una `$B`, la última se convertirá en un alias de la primera. Esto significa que la modificación del valor de cualquiera de ellas afectará a la otra. Este comportamiento resulta especialmente útil para implementar variables de entrada/salida en funciones.

Ejemplo 6. Paso de valor por referencia

```
<?
  $foo = "Igual";
  $bar = &$foo; // Se usa el signo & para indicar referencia
  echo "$foo = $bar";
```

```
?>
```

Queda como ejercicio averiguar cuál sería el resultado de ejecutar el código del ejemplo Ejemplo 6. Si se ha seguido la explicación, no ha de resultar nada complicado. Además, siempre puede hacerse la prueba.

5.3. Variables variables

PHP permite usar como nombre de una variable el contenido de una variable. Dicho de otro modo, es posible tener variables cuyo nombre es establecido dinámicamente -en tiempo de ejecución-. Para ello, basta con anteponer otro signo dólar ($) antes de la variable. Veamos un ejemplo para que quede más claro:

Ejemplo 7. Nombre de variable variable

```
<?
  $var  = "mivariable";
  $$var = "Contenido";
  echo "$$var";
?>
```

Por supuesto, la última sentencia imprimirá el valor de la variable `$mivariable`, que es `Contenido`.

Pero la cosa no queda aquí: PHP no sólo permite nombres de variables variables, sino que además puede hacerse lo mismo con las funciones. Así, los códigos del ejemplo anterior y el siguiente darán el mismo resultado.

Ejemplo 8. Nombre de funciones variables

```
<?
  $var  = "mivariable";
  $$var = "Contenido";
  $func = "echo";
  $func "$$var";
?>
```

6. Tipos de datos

Aunque los tipos de datos están directamente relacionados con las variables, parecen los suficientemente importantes para dedicarles una sección a parte. A continuación se estudiarán los diferentes tipos de datos que ofrece PHP, viendo algunos ejemplos de su utilización.

A este respecto hay que indicar que PHP es un lenguaje débilmente tipado: el intérprete decide el tipo de una variable en el momento de utilizarla, aunque permite que el programador indiqué cómo se debe considerar dicha variable. Así, si asignamos por ejemplo un entero a una variable, el intérprete asumirá que se trata de una variable de tipo entero. Pero la debilidad del tipado de PHP es tan elevada que el usuario puede decidir que esa variable, en principio entera, sea a partir de un momento determinado una cadena.

A priori, PHP soporta los siguientes tipos de datos:

- Enteros (int)
- Números en coma flotante (float)
- Cadenas
- Lógicos (bool)
- Vectores (arrays)
- Objetos (object)

A lo largo de esta sección se irán tratando todos y cada uno de estos tipos, a excepción de los objetos, a los que hemos reservado una sección a parte debido a que son algo más que simples tipos de datos.

6.1. Enteros

Los tipos numéricos, así como las cadenas, no requieren una explicación muy detallada de cuál es su finalidad. Como se vio anteriormente, PHP cuenta con dos tipos numéricos: uno destinado a los números enteros y otro a los reales. Veamos ahora cómo funcionan los primeros.

En PHP pueden representarse los enteros usando tres sistemas diferentes:

- Decimal. Este es el sistema, compuesto de 10 dígitos (0-9), que normalmente usamos los seres humanos. En PHP se escribe el número deseado, sin más. Así queda interpretado como decimal.
- Octal. Utiliza sólo 8 dígitos (0-7). Se antepone un cero al resto del número.
- Hexadecimal. Utiliza 16 dígitos. ¿Cuáles? Pues muy sencillo: del 0 al 9 y las seis primeras letras del alfabeto (A-F). Se indica que se debe intepretar como número hexadecimal usando el prefijo 0x.

En cualquiera de los casos puede anteponerse el signo menos (-) para indicar que se trata de un número negativo. Observemos el ejemplo Ejemplo 9 para que quede algo más claro.

Ejemplo 9. Uso de los números enteros en PHP

```
$epos = 2;    // Entero positivo
$eneg = -42;  // Entero negativo
$oct  = 012;  // Octal (equivale al 10 decimal)
$hex  = 0x21  // Hexadecimal (equivale al -33 decimal)
```

6.2. Reales

Los números en coma flotante se utilizan para representar reales. En principio, su sintaxis resulta bastante sencilla, existiendo dos posibilidades, tal y como se muestra en el ejemplo Ejemplo 10.

Ejemplo 10. Uso de los números enteros en PHP

```
$pi  = 3.1416; // Punto que separa parte entera y decimal
$npi = -3.1416;
$num = 1.1e3;   // Punto (opcional) y exponente base diez
$num = 1.1e-2;
```

Obviamente, se usará una u otra opción según conveniencia. Como puede observarse, puede anteponerse un signo menos (-) para indicar la negatividad del número o su exponente.

6.3. Cadenas de texto

Las cadenas de texto pueden especificarse usando dos tipos de delimitadores: comillas simples (’) o dobles ("). En caso de que se usen comillas dobles, las variables que se encuentren dentro de la cadena serán sustituidas por su valor. Además, pueden utilizarse caracteres de escape al estilo C (\n para el retorno de carro, \t para el tabulador, etc.).

Si por el contrario se usan comillas simples, no se hará interpretación alguna de las variables o los caracteres de escape. Veamos unos ejemplos.

Ejemplo 11. Uso de comillas dobles

```
$a = 3;
$b = 4;
$x = $a*$b;   // Multiplica $a y $b
echo "$a multiplicado por $b es igual a $x";
```

¿Qué imprimirá la sentencia `echo` del Ejemplo 11? Como se están usando comillas dobles, se interpretarán las variables. Así que el resultado será:

```
3 multiplicado por 4 es igual a 12
```

Supongamos ahora que hubiésemos usado comillas (’) simples en lugar de dobles. ¿Cuál hubiera sido el resultado? Obviamente, no se habría interpretado nada y se hubiera obtenido:

```
$a multiplicado por $b es igual a $x
```

Una curiosidad: ¿cómo podemos especificar que unas comillas pertenecen a una cadena? Es decir, supongamos por un momento que se quiere representar la cadena *su nombre es "Tux"*. ¿Cómo podría hacerse? En principio puede hacerse tanto con comillas simples como dobles. El ejemplo Ejemplo 12 contiene la solución.

Ejemplo 12. Representación de comillas

```
echo 'mi nombre es "Tux"';    // Usando comillas simples.
                              // A la inversa funciona también.
echo "mi nombre es \"Tux\"";  // Escapando las comillas dobles
```

Para terminar con los fundamentos de las cadenas, es obligatorio explicar cómo concatenarlas. En PHP el operador de concatenación de cadenas es el punto (.). Combinado con el signo igual (=) permite añadir otra cadena a una contenida en una variable.

Ejemplo 13. Concatenación de cadenas

```
/* Asignamos cadenas a dos variables $str1 y $str2 */
$str1 = "Nunca supe que dominaba el piano...";
$str2 = "hasta que instalé EMACS";

/* Impresión de la concatenación de cadenas usando el operador '.' */
echo $str1.$str2;

/* Un pequeño truco: concatenación sin usar el operador '.' */
echo "$str1$str2";

/* Adición de $str2 a $str1 */
$str1 .= $str2;     // Alternativa: $str1 = $str1.$str2;
```

6.4. Vectores

Como en todos los lenguajes, los vectores permiten agrupar un conjunto de valores. En PHP tienen una especial relevancia, ya que, sin contar con los objetos, se trata del único tipo compuesto. Aquí no existen los struct de C o las listas de Python.

Al estilo de Perl y otros lenguajes de *script* existen dos clases de vectores:

Escalares

: Se trata del vector convencional. Cada posición del vector se identifica unívocamente por un número.

Asociativos (hash)

Se trata de un concepto algo más evolucionado de vector. Para indexarlo se usan cadenas en lugar de números. Así podemos tener el valor 24 en la posición "edad".

Existen dos construcciones del lenguaje para crear vectores: `array()` y `list()`. Sin embargo vamos a empezar viendo un ejemplo bastante más sencillo que aprovecha el hecho de que en PHP puedan crearse vectores simplemente asignándoles valores, si necesidad de utilizar ninguna de las funciones anteriores.

Ejemplo 14. Fundamentos de los vectores

```
/* Vectores escalares */
$a[0] = "Hola";   // Vector escalar. Valor "Hola" en la posición 0.
$a[1] = "mundo";  // Valor "mundo" en la posición 1.
$a[]  = "!";      // Valor "!" en la posición 2. Si se omite el
                  // índice se toma el anterior + 1.

/* Vectores asociativos */
$b["nombre"] = "Silvia";
$b["edad"]   = 25;

/* Vectores escalares usando array */
$c = array("Hola", "mundo", "!");

/* Vectores asociativos usando array */
$d = array("nombre" => "Silvia", "edad" => 25);
```

Fijándonos en el ejemplo Ejemplo 14 nos daremos cuenta de algo bastante curioso: no es necesario que todos los valores de un vector sean del mismo tipo. Y, si no queda claro, probemos a ejecutar el código del ejemplo Ejemplo 15. En este código hay cosas que aún no se han visto... ya habrá tiempo.

Ejemplo 15. Distintos tipos de datos en un mismo vector

```
$b["nombre"] = "Silvia";
$b["edad"]   = 25;

/* El siguiente bucle imprime los tipos de cada
   uno de los componentes del vector $b */
foreach (array_keys($b) as $item) {
  echo gettype($b[$item]) . "\n";
}
```

Una característica de los vectores en PHP -y en muchos otros lenguajes- es que no están limitados a una sola dimensión. Los vectores multidimensionales permiten representar matrices y otras estructuras más complejas. La forma de indicar más de una dimensión es, simplemente, añadir más de un índice. Además, pueden mezclarse vectores asociativos y escalares (Ejemplo 16).

Ejemplo 16. Vectores multidimensionales

```
<?
$b[0]["nombre"] = "Silvia";
$b[1]["nombre"] = "Isabel";
$b[0]["edad"]   = 25;
$b[1]["edad"]   = 20;

/* De nuevo, se recomienda probar este código
   para ver el resultado */
foreach ($b as $person) {
  foreach (array_keys($person) as $field) {
    echo "$field:  $person[$field]<br>";
  }
}
?>
```

Ya sólo queda comentar un pequeño detalle respecto a los vectores: si se pretende imprimir el valor de un vector asociativo usando como índice un literal (por ejemplo $persona["nombre"]) nótese que existirá un problema con las comillas. La forma de solucionarlo es utilizar llaves de esta forma: {$persona["nombre"]}. Esta es una acotación necesaria importante, ya que mucha gente puede encontrarse con este problema. Aunque siempre está la opción de la concatenación.

7. Estructuras de control

Llegados a este punto deberíamos saber cómo se utilizan las variables y los diferentes tipos de datos de que se dispone. Aún falta la programación orientada a objetos, pero eso lo dejaremos para más adelante.

Así que en principio podríamos escribir pequeños *scripts* "lineales" que hicieran cosas muy sencillitas. ¿Y por qué lineales? Porque a excepción del `foreach` que hemos visto en algunos ejemplos no se ha nombrado ninguna otra estructura de control (sentencias que permiten cambiar el curso del programa). Éste es el momento de presentarlas, para poder empezar a ver ya algunos ejemplos más provechosos.

7.1. Bucles

Los bucles (repeticiones de sentencias) constituyen una de las estructuras de control más socorridas en el mundo de la programación. PHP cuenta, en principio, con cuatro tipos de bucles, a saber:

- `while`
- `do .. while`
- `for`
- `foreach`

Vamos a estudiar brevemente las características y peculiaridades de cada uno de ellos.

7.1.1. while

```
while (condición)
  sentencias;
```

Es la implementación del bucle con condición inicial. Se evalúa la condicion y si se cumple se ejecuta el grupo de sentencias. Al terminar se vuelve a evaluar la condición y, si vuelve a cumplirse, se entra de nuevo en la ejecución de las sentencias. Se sale del bucle cuando una de las comprobaciones de la condición falla.

Ejemplo 17. Bucle `while`

```
/* El siguiente bucle imprime los número de 0 a 99 */
$i = 0;
while ($i < 100)
  echo $i++;

/* Buscando un valor en un vector. Obviamente, hay funciones para
   hacer esto, pero se trata de un simple ejemplo. */
$i = 0;
while ($a[$i] != $tofind]) {
  $i++;
}
```

En el `while` y en todas las estructuras de control las llaves son opcionales siempre que exista una sóla sentencia. Cuando haya más de una se convertirán en obligatorias.

7.1.2. do..while

```
do
  sentencias;
while (condición);
```

El funcionamiento de este bucle resulta bastante parecido al de Sección 7.1.1. La diferencia estriba en que la condición se escribe y evalúa al final del bucle. Es decir, las sentencias se ejecutan, al menos, una vez.

Ejemplo 18. Bucle `do..while`

```
/* El siguiente bucle imprime los número de 0 a 99 */
$i = 0;
do {
  echo $i++;
} while ($i < 100);
```

7.1.3. for

```
for (expresion1; expresion2; expresion3) sentencias;
```

Probablemente sea este el núcleo más "poderoso". Su funcionamiento, al estilo del `for` del lenguaje C, es el siguiente:

- La primera expresión una vez al principio y de forma incondicional.
- Al principio de cada iteración se evalúa la expresión 2 y sólo si es verdadera se continúa iterando.
- Al final de cada iteración, y de forma incondicional, se ejecuta la expresión 3.

Las tres expresiones utilizadas en este bucle pueden ser asignaciones o condiciones. Incluso puede tratarse de varias expresiones separadas por comas. Los ejemplo, nuevamente, ilustrarán estos conceptos.

Ejemplo 19. Bucle `do..while`

```
/* Un ejemplo sencillo inicializando un
vector de $n elementos al valor 0 */
for ($i = 0; $i < $n; $i++) {
  $a[$i] = 0;
}

/* El bucle puede complicarse cuanto queramos */
for ($i = 0, $j = 4; $i < $n, $j < $p; $i++, $j += 4) {
  ...
}

/* He aquí un bucle infinito: suele combinarse con algún 'break' */
for (;;) {
  ...
}
```

7.1.4. foreach

```
foreach ($vector as $var) {
 sentencias;
}

foreach ($vector as $clave=>$valor) {
 sentencias;
}
```

PHP dispone de una estructura de control que permite iterar de forma sumamente cómoda sobre un vector: `foreach`. Como puede observarse, dispone de dos sintaxis posibles.

- En el primer caso va recorriéndose el vector poniendo el valor de la posición actual en la variable `$var` y ejecutando las sentencias.
- En el caso de la segunda sintaxis no sólo se coloca el valor de la posición actual en `$var`, sino que además -en el caso de vectores asociativos- se pone su clave en `$clave`.

Ejemplo 20. Bucle `foreach`

```
/* Recorriendo un vector */
foreach ($a as $item) {
  echo $item;
}

/* Recorriendo un vector incluyendo sus índices */
foreach (array_keys($a) as $key) {
echo "$key: $a[$key] <br>";
}

/* Usando la segunda sintaxis para hacer lo mismo */
foreach ($a as $key=>$valor) {
  echo "$key: $valor <br>";
}
```

7.2. Alternativas

Estas estructuras desvían el flujo de ejecución del programa en función de una condición. PHP incluye las dos construcciones más usuales en el mundo de la programación:

- `if`
- `switch .. while`

7.2.1. if, else, elseif

```
if (condicion)
  sentencias;
[elseif (condicion)]
  sentencias;
[else]
  sentencias;
```

Esta es la estructura condicional más conocida y extendida. Se encuentra presente, de una u otra forma, en la mayoría de los lenguajes de programación.

En esta construcción, la palabra reservada *if* aparece una sóla vez; *elseif* puede aparecer cuantas veces se desee; y *else* puede hacerlo, a o sumo, una sola vez.

La interpretación es la siguiente:

- Si la condición de *if* es verdadera, se ejecuta el primer grupo de sentencias y se sale de la estructura sin pasar por el resto. Si esta condición es falsa, se pasa al primer *elseif* y, si este no existe, al *else*. Obviamente, si tampoco existe el *else* se sale de la estructura.

  El *elseif* se comporta igual que el *if*. Si se cumple la condición ejecuta su grupo de sentencias, y si no pasa el siguiente *elseif* (si existiera) o al *else*.

  El grupo de sentencias del *else* es la alternativa a todos los anteriores, es decir, sólo se ejecutará si ninguna de las condiciones anteriores se ha cumplido.

Ejemplo 21. `if`

```
if ($a == $b)
{
  $a = 0;
  $b = 1;
} else if ($a < $b)
  $a = 1;
} else {
  $b = 1;
}
```

Al igual que en los bucles, sólo es necesario el uso de llaves cuando el grupo de sentencias tiene más de una sentencia. Sin embargo, es una buena práctica ponerla siempre.

7.2.2. switch

```
switch (expresion) {
  case valor1:
    sentencias;
    [break;]
  case valor2:
    sentencias;
    [break;]
  case valor3:
    sentencias;
    [break;]
  ...
  default:
    sentencias;
}
```

En esta estructura se evalúa la expresión y se compara, por orden, con los valores. Si coincide, se ejecuta el grupo de sentencias correspondiente. Si no es así se pasa al siguiente valor. Puede usarse el `break` para salir del switch.

Ejemplo 22. `if`

```
switch ($op) {
  case 0:
    $i = 4;
    break;
  case 1: /* equivale a ($op == 1 || $op == 2) */
  case 2:
    $i = 0;
    break;
  default:
    $i++;
}
```

Es de dominio público que el comportamiento de la estructura `switch` puede emularse utilizando `if`. Así, el ejemplo Ejemplo 22 anterior podría escribirse también como se muestra en Ejemplo 23.

Ejemplo 23. Simulando un `switch` con un `if`

```
if ($op) {
  $i = 4;
} else if (($op == 1) || ($op == 2)) {
  $i = 0;
} else {
  $i++;
}
```

8. Funciones

Como todo lenguaje de programación que se precie, PHP permite que el usuario defina sus propias funciones. En esta sección vamos a estudiar cuál es la sintaxis para definir nuevas funciones, de qué modo han de pasarse los parámetros y cómo devolver valores.

La sintaxis a la hora de definir las nuevas funciones resulta bastante sencilla: simplemente hay que poner la palabra reservada `function` seguida del nombre de la función y, a continuación, entre paréntesis los nombres de los parámetros separados por comas. Después de la definición de los parámetros ya sólo resta poner el cuerpo de la función, delimitado por sendas llaves ({}).

Ejemplo 24. Pseudo código de definición de una función

```
function nombre($arg1, $arg2, ...)
{
  // Cuerpo de la función
  echo "función inútil";
  return false;
}
```

Como vemos, en la definición de funciones PHP se cuenta con la particularidad de que no se especifica el tipo del valor devuelto por esta ni el de ninguno de sus argumentos. Esto se debe al débil tipado del lenguaje, ya visto con anterioridad.

Por defecto, los valores de los parámetros son pasados "por valor", es decir, se realiza una copia en estos parámetros y su modificación no afecta a la variable que se pasó. Este comportamiento puede variarse si se aplica lo visto en la Sección 5.2. Así, basta con anteponer el signo & a cualquiera de los argumentos -en su declaración- para que se consideren de entrada/salida.

El último detalle que comentaremos de las funciones es el modo en que se devuelven valores. Una función puede devolver cualquier expresión y, para ello, se utiliza la palabra reservada `return`.

9. Orientación a objetos

Como lenguaje moderno que es, PHP tiene soporte para la programación orientada a objetos. Se dará por supuesto que el lector conoce ya los conceptos de clase, objeto, atributo, método, etc. quedando fuera del alcance de este tutorial explicar sus fundamentos.

Antes de empezar a explicar cómo funciona el soporte para objetos en PHP, hay que destacar que se preparan grandes cambios al respecto para la versión 5 del lenguaje, ya que, siendo objetivos, hoy por hoy es un poco pobre. Para más información es aconsejable visitar la siguiente URL:
http://www.php.net/zend-engine-2.php (http://www.php.net/zend-engine-2.php)

9.1. Definición de clases

En Ejemplo 25 se muestra la estructura que tiene la definición de una clase en el lenguaje PHP. Para indicar que se trata de una clase se utiliza la palabra reservada `class` seguida del nombre de la clase. A continuación se colocan todos los atributos (precedidos por la palabra reservada `var`) y las diferentes funciones miembro que formarán parte de la clase.

Ejemplo 25. Definición de una clase

```
<?php
```

```
  class vector2D {
    var $x, $y;

    /* El constructor lleva el nombre de la clase */
    function vector2D($x = 0, $y = 0) {
      $this->$x = $x;
      $this->$y = $y;
    }

    function setX($val) {
      $this->$x = $val;
    }

    function setY($val) {
      $this->$y = $val;
    }

    function toArray()
    {
      return array($this->$x, $this->$y);
    }

    function toHash($x, $y)
    {
      return array("x" => $this->$x, "y" => $this->$y);
    }
  }
?>
```

Una vez definida la clase simplemente hay que instanciar tantos objetos como necesitemos. Para ello disponemos de la construcción `new`, tal y como se muestra en el Ejemplo 26.

Ejemplo 26. Instanciando un objeto

```
$vector = new vector2D(3,-1);
$array  = vector2D->toHash();
echo "X = {$array["x"]} Y = {$array["y"]}";
```

9.2. Herencia

Una de las pocas cosas que soporta la orientación a objetos en PHP es la herencia. La palabra clave `extends` expresa esta relación.

Ejemplo 27. Herencia

```
class vector3D extends vector2D {
  var $z;

  function vector3D($x = 0, $y = 0, $z = 0)
  {
    $this->$x = $x;
    $this->$y = $y;
    $this->$z = $z;
  }

  function setZ($val)
  {
    $this->$z = $val;
  }

  function toArray()
  {
    return array($this->$x, $this->$y, $this->$z);
  }

  function toHash($x, $y)
  {
    return array("x" => $this->$x,
                 "y" => $this->$y,
                 "z" => $this->$z);
  }
}
```

10. Programación web

Como ya se mencionó con anterioridad, la web es el entorno para el que PHP fue pensado. Así que, al respecto, cuenta con algunas comodidades sobre otros lenguajes como pudieran ser Perl o C. En este capítulo vamos a analizar cómo escribir sencillos programas para la web. Una vez entendidos estos conceptos se pasará a explicar cómo funciona el acceso a bases de datos en PHP, para poder construir aplicaciones web más complejas.

10.1. Variables superglobales

Como punto de partida para la programación web vamos a conocer lo que en PHP se conoce como variables *autoglobales* o *superglobales*.

A partir de la versión 4.1.0 PHP ofrece un conjunto de vectores predefinidos que contiene variables del servidor -si se trata de un entorno web-, el entorno y las entradas del usuario. Estos vectores están

automáticamente accesibles -de ahí el nombre- desde cualquier punto del *script*. Hay que destacar que PHP no ofrece ningún mecanismo para que el usuario pueda definir variables de este tipo, teniendo que utilizar éste la construcción *global* para hacer accesible una variable global desde, por ejemplo, una función.

`$GLOBALS`

Contiene una referencia a cada variable que se encuentra actualmente accesible en el *script*, incluso las globales definidas por el usuario.

`$_SERVER`

Este vector asociativo contiene información acerca del servidor.

`$_GET`

Variables pasadas al *script* vía HTTP GET.

`$_POST`

Variables pasadas al *script* vía HTTP GET.

`$_COOKIE`

Variables pasadas al *script* vía *cookies* HTTP.

`$_FILES`

Variables que controlan la subida de ficheros mediante HTTP.

`$_ENV`

Variables de entorno.

`$_REQUEST`

Une el contenido de `$_GET`, `$_POST` y `$_COOKIE`.

`$_SESSION`

Variables registradas en la sesión actual.

Una vez presentadas estas variables, vamos a realizar un ejercicio para poner en práctica conceptos ya vistos con anterioridad al tiempo que nos familiarizamos con ellas.

Para empezar, vamos a escribir un *script* que muestre toda la información relativa al servidor y la presente en una tabla HTML. Para ello recorreremos el vector autoglobal `$_SERVER` utilizando la estructura `foreach`, al tiempo que vamos construyendo la tabla. El código, en el ejemplo Ejemplo 28.

Ejemplo 28. Variables globales

```
<?
  /* Cabecera de la tabla */
  $html = "<TABLE><TR><TH>Variable</TH><TH>Valor</TH></TR>\n";
  /* Se recorre el vector mostrando los pares clave/valor */
```

```
  foreach ($_SERVER as $key=>$val) {
    $html.= "<TR><TD>$key</TD><TD>$val</TD></TR>\n";
  }
  $html.= "</TABLE>\n";
  /* Se manda el resultado al navegador */
  echo $html;
?>
```

10.2. Formularios

Este apartado no está dedicado a aprender a programar formularios HTML (o XHTML). Para conseguir información acerca de eso, lo mejor es acudir a la web del World Wide Consortium (http://www.w3c.org/).

El objetivo de esta sección es desarrollar un sencillo formulario para que el lector tenga algo más de "rodaje" antes de llegar al plato fuerte: la programación PHP con bases de datos.

El ejercicio consistirá, simplemente, en mejorar el *script* que se vió en la sección anterior. Se presentará un formulario con todas las variables autoglobales disponibles de forma que el usuario pueda escoger una de ellas y visionar su contenido. Tomemos unos minutos para pensarlo y, entonces, analicemos la solución que se dará a continuación.

A excepción de alguna modificación, el código que muestra el contenido de una variable ya lo tenemos (ejemplo Ejemplo 28. Ahora simplemente es necesario escribir el formulario para, acto seguido, mezclarlo todo.

¿Cómo implementar el formulario? Parece claro que será necesario recorrer el vector `$_GLOBALS` extrayendo los nombres de las variables que se encuentran definidas. Nuevamente parece claro que lo más cómodo es utilizar la construcción `foreach`. El código, en el Ejemplo 29.

Ejemplo 29. Formulario. Petición de la variable a analizar

```
<?
  $html = "<FORM ACTION=\"{$_SERVER["PHP_SELF"]}\" METHOD=\"post\">";
  $html.= "<SELECT NAME=\"variable\">";
  /* array_keys devuelve las claves de un vector asociativo */
  foreach (array_keys($GLOBALS) as $var) {
    $html.= "<OPTION VALUE=\"$var\">$var\n";
  }
  $html.= "</SELECT>\n";
  $html.= "<INPUT NAME=\"accion\" TYPE=\"submit\" VALUE=\"Ver\">\n";
  $html.= "</FORM>\n";
```

Ya tenemos un *script* que construye una tabla tomando los valores de un vector (Ejemplo 28) y un formulario para seleccionar una de las variables globales (Ejemplo 29). Ahora simplemente hay que mezclar una y otra cosa y hacer un par de cambios:

- El recorrido por los valores (Ejemplo 28) debe realizarse ahora sobre la variable indicada por el usuario, cuyo nombre se recoge en `$_POST["variable"]`. Por tanto se usará el contenido de esa variable como nombre de la variable a recorrer, poniendo en práctica lo visto en la Sección 5.3.
- El otro cambio es un pequeño detalle que podría pasar desapercibido: la eliminación, en la primera línea del Ejemplo 28, del punto que sucede a `$html`.

Si se prueba el *script* se verá que aparecen variables globales con las que no se contaba: `$HTTP_POST_VARS` o `$HTTP_GET_VARS` son dos de los ejemplos. Decir al respecto que el uso de estas variables está desconsejado en favor de las más modernas `$_POST` y `$_GET`.

Ejemplo 30. *Script* que muestra los valores de la variable indicada por el usuario.

```
<?
  /*
   * variables.php
   */

  /* Formulario para la petición de variable a analizar */
  $html = "<FORM ACTION=\"{$_SERVER["PHP_SELF"]}\" METHOD=\"post\">";
  $html.= "<SELECT NAME=\"variable\">";
  /* array_keys devuelve un vector con las claves de un vector asociativo */
  foreach (array_keys($GLOBALS) as $var) {
    $html.= "<OPTION VALUE=\"$var\">$var\n";
  }
  $html.= "</SELECT>\n";
  $html.= "<INPUT NAME=\"accion\" TYPE=\"submit\" VALUE=\"Ver\">\n";
  $html.= "</FORM>\n";

  /* Tabla de valores de la variable */
  $html.= "<TABLE><TR><TH>Variable</TH><TH>Valor</TH></TR>\n";

  /* Se comprueba si es o no un vector */
  if (is_array($$_POST["variable"])) {
    foreach ($$_POST["variable"] as $key=>$val) {
      $html.= "<TR><TD>$key</TD><TD>$val</TD></TR>\n";
    }
  } else { /* Si no es un vector, la tabla se simplifica bastante */
    $html.= "<TR><TD>{$_POST["variable"]}</TD><TD>{$$_POST["variable"]}</TD></TR>\n";
  }
  $html.= "</TABLE>\n";
  echo $html;


?>
```

11. Programación con bases de datos

Uno de los aspectos más interesantes de PHP es la cantidad de bases de datos que soporta. Ésta es la relación de las 21 soportadas en el momento de escribir este tutorial:

- Adabas D
- dBase
- Empress
- FilePro (sólo lectura por el momento)
- Hyperwave
- IBM DB2
- Informix
- Ingres
- Interbase
- Frontbase
- mSQL
- Direct MS-SQL
- MySQL
- ODBC
- Oracle
- Ovrimos
- PostgreSQL
- Solid
- Sybase
- Velocis
- Unix dbm

Esta sección estará destinada a ver cómo trabaja PHP con bases de datos. En principio, se ha escogido PostgreSQL (http://www.postgresql.org/) para realizar las primeras pruebas. Más adelante se estudiará una interesante alternativa que permitirá escribir código compatible con múltiples SGBD[2].

Seguramente surgirá la pregunta: ¿por qué PostgreSQL? ¿Por qué no otro SGBD como MySQL? La razón es sencilla: existe mucha documentación en la red acerca de la programación con PHP y MySQL, así que no está de más tratar en este caso el desarrollo con otro de los grandes sistemas de BBDD libres[3].

El único handicap que pudiera tener PostgreSQL, es que sólo funciona sobre sistemas UNIX. Sin embargo, y a pesar de que seas usuario de otro sistema como Windows, siempre podrás leer esta sección,

ya que muchos de los fundamentos que así se expondrán son aplicables al resto de sistemas. A último remedio podrías plantearte probar otro sistema operativo.

11.1. PostgreSQL

11.1.1. Configuración

Obviamente, la configuración y puesta en marcha de PostgreSQL queda fuera del alcance de este tutorial. La documentación al respecto de este SGBD es realmente buena: consúltela.

En lo que al intérprete de PHP se refiere, será necesario modificar ligeramente su fichero de configuración `php.ini` para activar el soporte para el SGBD PostgreSQL . Se trata de añadir una línea que, por otro lado, puede que ya se encuentre en el fichero y no haya más que descomentarla: *extension=pgsql.so* en el caso de sistemas UNIX y *extension=pgsql.dll* en sistemas Windows -a pesar de que PostgreSQL sólo funcione sobre UNIX, siempre puedes conectar desde una máquina Windows-.

Nótese que este cambio tan sólo es necesario si se ha compilado el soporte para PostgreSQL como extensión. Si no sabe de qué estamos hablando, es probable que así sea y no debería haber problema. Huelga decir que si PHP se compiló sin soporte para PostgreSQL, nada de esto funcionará.

11.1.2. Conexión a PostgreSQL

Antes de poder realizar consultas, actualizaciones, inserciones, etc. es necesario, como primer paso, conectarse al servidor de bases de datos. En el caso de PostgreSQL se necesita una cuenta, es decir, un nombre de usuario y una contraseña que nos dé acceso al sistema. Por supuesto, también será necesario crear una base de datos a la que conectarnos.

Aunque se aconseja consultar la Guía del administrador (http://www.postgresql.org/docs/7.3/static/admin.html)[4] vamos a ver cómo podría crearse una nueva base de datos y un usuario para hacer algunas pruebas.

```
# su - postgres
$ createuser -P tutorial
Enter password for user "tutorial":
Enter it again:
Shall the new user be allowed to create databases? (y/n) n
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

Ya se tiene un nuevo usuario *tutorial* con la contraseña que hayamos escogido. Como puede verse, este usuario no será capaz de crear ni nuevas bases de datos ni usuarios. Ahora tan sólo resta crear una nueva base de datos con la que jugar.

```
$ createdb dbtutorial
CREATE DATABASE
```

Los detalles de seguridad como desde dónde se permite conectarse, si se permite hacerlo sin contraseña o si ésta se transmite o no cifrada, quedan bajo la elección y control del lector. Para más información, consultar el capítulo Client Authentication[5] de la documentación de PostgreSQL.

Una vez preparada una base de datos y una cuenta de usuario, podemos empezar a pensar en el modo de conectar PHP con el SGBD. El lenguaje nos ofrece una serie de funciones[6] funciones para interactuar con esta base de datos. Empezaremos echando un vistazo a la primera que utilizaremos: `pg_connect`.

```
resource pg_connect(string cadena_conexion);
```

Esta función permite al intérprete conectarse con el servidor de bases de datos. Recibe como argumento una cadena que expresa qué parámetros han de usarse en la conexión, y devuelve un recurso[7]

En la cadena de conexión pueden definirse los siguientes parámetros:

host
: La máquina donde escucha el servidor PostgreSQL[8]

port
: Puerto de conexión.

tty
: *TTY* a utilizar. Interesante a la hora de depurar.

options
: Opciones adicionales.

dbname
: Nombre de la base de datos a la que se quiere conectar.

user
: Nombre del usuario PostgreSQL con el que se realizará la conexión. En nuestro caso, *tutorial*.

pass
: Contraseña para la conexión.

Esta podría ser una invocación válida:

```
<?
$conn = pg_connect("dbname=dbtutorial user=tutorial password=chie3Iej");
if ($conn) {
  echo "<P>conexión OK</P>";
} else {
  echo "<P>conexión KO</P>";
}
?>
```

Este código tratará de conectar a la base de datos y mostrará un mensaje en función de que haya conseguido o no conectarse. En este último caso `$conn` almacenará el valor *false* en caso de que la conexión no haya resultado satisfactoria.

Nótese que en la cadena de conexión se pasan de forma literal el usuario y la contraseña. Obviamente, esto es una mala práctica de programación. Esos datos deberían estar almacenados, por ejemplo, en algún fichero para cargarlos en tiempo de ejecución, ya que si no cualquiera que tuviera acceso al *script* podría averiguar el usuario y contraseña de conexión. Además, estos datos han de estar guardados fuera del directorio web, para que en caso de que el servidor se viera comprometido fuera más complicado acceder a ellos.

11.1.3. Ejecución de sentencias SQL

Una vez establecida la conexión, pueden comenzar a intercambiarse peticiones y respuestas con el servidor. Lo más habitual, es que se lancen consultas SQL y se recojan sus resultados para su tratamiento. Una vez dicho esto, podemos ver que el proceso se divide en dos pasos:

1. Enviar la consulta a ejecutar (`pg_query`).
2. Recoger los resultados de la ejecución.

Para enviar una consulta se utiliza la función `pg_query`:

```
resource pg_query(resource conexión, string consulta);
```

El primer parámetro es opcional. Sin embargo, se aconseja especificarlo, ya que puede ser la causa de fallos difíciles de localizar. Así, si por ejemplo quisiéramos rescatar todos los registros de una hipotética tabla articulo que contuviera información acerca de los artículos de una publicación:

```
/* Establecimiento de la conexión */
$conn = pg_connect("dbname=dbtutorial user=tutorial password=chie3Iej")
or die("Error");

if ($conn) {
```

```
  /* Se envía la consulta */
  $result = pg_query($conn, "SELECT * FROM articulo");
}
```

En este código se ha introducido como novedad la construcción `or die`, presente en lenguajes como Perl. Su cometido es detener el *script* en caso de que la línea a la que se encuentra asociado -en este caso la invocación a la función `pg_connect`- produzca un error. Al detener el *script* imprime la cadena que se le pasó como argumento.

Llegados a este punto, el servidor de bases de datos ejecutará el código que le hemos pasado, pero ¿dónde están los resultados? La variable `$result` contendrá un recurso que permite al servidor de bases de datos identificar cuál ha sido el resultado de esa consulta. Así, para recuperarlos, no habrá más que invocar a alguna de las funciones destinadas al efecto indicando el recurso al que queremos acceder.

PHP ofrece multitud de funciones para rescatar los resultados generados por una consulta a una base de datos PostgreSQL. En principio, todas las funciones cuyo nombre empieza por `pg_fetch_` están destinadas a esto.

`pg_fetch_all`

Devuelve un vector que contiene todos los registros devueltos por la consulta.

`pg_fetch_array`

Devuelve la siguiente fila del resultado en forma de vector. El primer elemento será la primera columna, el segundo la segunda, etc. Devuelve falso si no encuentra más registros.

`pg_fetch_assoc`

Devuelve la siguiente fila del resultado en forma de vector asociativo. Como claves del vector usa los nombres de los campos. Devuelve falso si no encuentra más registros.

`pg_fetch_object`

Devuelve una fila del resultado en forma de objeto. Por cada columna de la fila existirá un atributo del mismo nombre.

`pg_fetch_result`

Devuelve un valor de un resultado. Se le especifica fila y columna.

`pg_fetch_row`

Devuelve una fila del resultado en forma de vector. El primer elemento será la primera columna, el segundo la segunda, etc. Devuelve falso si no encuentra más registros.

Veamos cómo podríamos recorrer el resultado de la consulta anterior utilizando diferentes funciones. En el recorrido se mostrará, por ejemplo, el valor de un supuesto campo titular.

```
/* sea $result el recurso que identifica la respuesta */
```

```
/* pg_fetch_object */
while ($row = pg_fetch_object($result)) {
  echo $row->titular;
}

/* pg_fetch_assoc */
while ($row = pg_fetch_assoc($result)) {
  echo $row["titular"];
}

/* pg_fetch_row: en este caso ha de conocerse qué número de columna
   corresponde al campo que se quiere mostrar. Este es el handicap
   de esta función que, sin embargo, resulta algo más rápida que las
   demás. */
while ($row = pg_fetch_row($result)) {
  echo $row[0];
}

/* pg_fetch_result: nos apoyamos en pg_num_rows para saber con
   cuántas filas se cuenta. pg_fetch_result es muy útil para otros
   casos, pero no es la mejor opción a la hora de recorrer
   resultados enteros de consultas. Nuevamente hay que conocer qué
   columna corresponde con el campo buscado: supongamos que es la
   primera (0) */
for ($i = 0; $i < pg_num_rows($result); $i++) {
  echo pg_fetch_result($i, 0, $result);
}
```

11.1.4. Una pequeña aplicación de ejemplo

Para ilustrar algunos de los conceptos que se han visto hasta ahora se va a construir una sencilla aplicación.

En principio, el objetivo será construir una aplicación que permita que un usuario envíe su nombre y su correo-e a una base de datos, como si de una sencilla firma se tratara. Se registrará la hora del envío y, además, se ha de mostrar una lista con los usuarios que ya han "firmado". Sencillo, ¿verdad?

Como base de datos utilizaremos *dbtutorial*, ya creada con anterioridad. Cómo se diseña la base de datos no es lo importante en este ejercicio, así que simplemente se construirá una tabla con los tres campos necesarios: nombre, correo y fecha. Éste podría ser el código SQL:

```
CREATE TABLE firma (
  nombre char(30) NOT NULL,
  correo varchar(100) NOT NULL,
  fecha date NOT NULL,
  PRIMARY KEY(nombre, correo));
```

```
);
```

Este código bien podemos teclearlo directamente en el intérprete de PostgreSQL o bien podemos meterlo en un fichero para luego hacer que el sistema lo interprete. Es más aconsejable, obviamente, la segunda opción, ya que así se conserva el código por si, más adelante, pudiera hacer falta.

Después de crear la tabla será necesario dar permisos al usuario que va a conectarse a la base de datos para que pueda rescatar e insertar datos. Así que al final añadimos esta línea:

```
GRANT INSERT, SELECT ON dbtutorial TO tutorial;
```

Con esto ya tenemos listo el código referente al SGBD. A continuación, habría que escribir la pequeña aplicación en PHP. Sería un ejercicio interesante que el lector lo intentara por si mismo. El código del Ejemplo 31 ilustra una posible solución al problema: ni la mejor ni la peor... simplemente una cualquiera.

Ejemplo 31. Firmas

```
<?
/*
 * Ejemplo de aplicación PHP con PostgreSQL.
 * Pide al usuario un nombre y una dirección de correo-e y los introduce,
 * junto a la fecha, en una base de datos. Además, muestra los datos ya
 * introducidos por otros usuarios.
 */

/* Este fichero contiene la configuración. Es bueno que esté separado y, a ser
 * posible, fuera del árbol de la web. */
include "/home/imo/config-gb.php";

/*
 * Conexión a la base de datos.
 */
$conn = pg_connect("dbname={$config["dbname"]} user={$config["user"]} password={$config["pa


/*
 * Si la acción es "firmar" ejecutará un código diferente. ¿Por qué un switch?
 * Pues porque es más cómodo si se quieren añadir nuevas acciones del mismo
 * estilo. La variable "accion" (enviada vía POST) decide qué acción se
 * realizará.
 */
switch ($_POST["accion"]) {

  /* firmar: introduce los datos en la BBDD */
  case "firmar":
    /* Construye la sentencia SQL para introducir los datos */
    $qry = "INSERT INTO firma (nombre, correo, fecha) VALUES
```

```
    ('{$_POST["nombre"]}','{$_POST["correo"]}', '".date("y-m-d")."')";
    /* Ejecuta la acción. La '@' evita que mysql_query suelte información en
     * caso de que se produzca un error */
    $ret = @mysql_query($qry)
           or die("<p>Error: no se pudo completar la operación</p>");
    break;

  default:
    /* Formulario */
    $html = "<FORM ACTION=\"{$_SERVER["PHP_SELF"]}\" METHOD=\"post\">\n";
    $html.= "<P><INPUT NAME=\"nombre\" TYPE=\"text\" MAXLENGTH=\"30\"
    SIZE=\"30\"> Nombre<BR>\n<INPUT NAME=\"email\" TYPE=\"text\"
    MAXLENGTH=\"100\" SIZE=\"30\"> Correo-e</P>\n";
    $html.= "<INPUT NAME=\"accion\" TYPE=\"submit\" VALUE=\"firmar\">\n";
    $html.= "</FORM>\n";

    /* Lista de firmas */
    $html.= "<TABLE><TR><TH>Nombre</TH><TH>Correo</TH><TH>Fecha</TH></TR>\n";

    /* Selecciona todas las firmas de la base de datos */
    $ret = @pg_query($conn, "SELECT * FROM firma");

    /* Recorre toda la base de datos, fila a fila, construyendo la tabla que
     * mostrará los datos al final.
     */
    while ($firma = pg_fetch_assoc($ret)) {
      $html.= "<TR><TD>{$firma["nombre"]}</TD><TD>{$firma["email"]}</TD><TD>{$firma["fecha"
    }
}

/* Se produce la salida por pantalla */
echo $html;
?>
```

11.2. Escribiendo código genérico

Uno de los requisitos deseables de cualquier pieza de software, es que sea tan portable y corra sobre tantos sistemas como sea posible. En el caso de PHP, no suele haber grandes problemas, ya que con el intérprete adecuado se puede hacer correr una aplicación en plataformas muy dispares.

Esta misma portabilidad sería también deseable en lo que al SGBD con que combinamos nuestra aplicación se refiere. No es necesario comentar las ventajas que pueden obtenerse de la posibilidad de escribir un código que pueda apoyarse, con un mínimo cambio, sobre una base de datos MySQL, PosgreSQL, Interbase, etc.

Amén de la solución de utilizar ODBC (lo que puede traer algunas complicaciones), existe la posibilidad de utilizar alguna *capa de abstracción* que unifique todas las llamadas a las funciones. Al respecto existe algunas opciones interesantes: no hay más que buscar en la base de datos del proyecto PEAR[9] para encontrar algunas cosas interesantes.

En este tutorial no se estudiará el uso de ninguna de estas alternativas, pero sí que se aconseja que el lector lo haga en aras de escribir un código más independiente del sistema en el que corra.

11.3. Seguridad

En un entorno inherentemente inseguro, es necesario tomar algunas precauciones para no sucumbir a los posibles ataques que pudiera sufrir nuestra aplicación. A continuación se van a dar unos pocos consejos para la seguridad en la programación con PHP y bases de datos. Conviene seguir todos estos consejos, ya que así se dificulta la labor de un posible atacante.

1. Nunca conectar a la base de datos como el "superusuario" o el propietario de la base de datos. En su lugar, utilizar un usuario creado a tal efecto y con los mínimos privilegios posibles.
2. Asegúrate de que las entradas del usuario son del tipo apropiado. Para ellos existen funciones como is_numeric() (http://www.php.net/manual/en/function.is-numeric.php) o ctype_digit(). (http://www.php.net/manual/en/function.ctype-digit.php) En caso de que sea necesario, puede cambiarse usando settype() (http://www.php.net/manual/en/function.settype.php) o los diferentes tipos de representación de sprintf() (http://www.php.net/manual/en/function.sprintf.php).
3. Utiliza las funciones addslashes() (http://www.php.net/manual/en/function.addslashes.php) o addcslashes() (http://www.php.net/manual/en/function.addcslashes.php) para tratar las entradas no numéricas del usuario.
4. No imprimas ninguna información específica de la base de datos, especialmente en lo que a estructuras se refiere. Echa a un vistazo a las funciones para el manejo de errores (http://www.php.net/manual/en/ref.errorfunc.php) de que dispone PHP.
5. Si es posible, guarda información acerca de las consultas que se hacen a la base de datos. No prevendrá un posible ataque, pero en caso de desastre será útil para averiguar qué pasó.

A. Operadores PHP

A lo largo del texto no se ha hecho énfasis ninguno en los operadores que ofrece PHP. En este apéndice se muestra una lista de algunos de ellos.

Tabla A-1. Algunos operadores PHP

Ejemplo	Nombre
Aritméticos	
$a + $b	Suma

Ejemplo	**Nombre**
$a - $b	Resta
$a * $b	Multiplicación
$a / $b	División
$a % $b	Módulo
A nivel de bits	
$a & $b	AND
$a | $b	OR
$a ^ $b	XOR
~ $a	NOT
$a << $b	Desplazamiento izquierda
$a >> $b	Desplazamiento derecha
Comparación	
$a == $b	Igual
$a === $b	Idéntico
$a != $b	Distinto
$a <> $b	Distinto
$a !== $b	No idéntico
$a < $b	Menor que
$a > $b	Mayor que
$a <= $b	Menor o igual que
$a >= $b	Mayor o igual que
Incremento/Decremento	
$a++	Preincremento
++$a	Postincremento
$a--	Predecremento
--$a	Postdecremento
Lógicos	
$a and $b	AND
$a or $b	OR
$a xor $b	XOR
! $a	NOT
$a && $b	AND
$a || $b	OR

Notas

1. La traducción en este ámbito sería *guión*.
2. Sistemas de Gestión de Bases de Datos

3. Para más información acerca del *Software Libre*, visitar la web de la Free Software Foundation (http://www.fsf.org/home.es.html)
4. Guía del Administrador de PostgreSQL 7.3: http://www.postgresql.org/docs/7.3/static/admin.html (http://www.postgresql.org/docs/7.3/static/admin.html)
5. Client Authentication (http://www.postgresql.org/docs/7.3/static/client-authentication.html)
6. Funciones PHP (http://www.php.net/manual/en/ref.pgsql.php) para PostgreSQL
7. Tipo especial de datos que identifica, en este caso, a la conexión.
8. Sólo para TCP/IP. Por lo general, no debe especificarse si el intérprete PHP y el servidor PostgreSQL corren en la misma máquina, a no ser que éste último esté atendiendo peticiones TCP/IP -algo que debe evitarse siempre que sea posible-.
9. PEAR (http://pear.php.net/): PHP Extension an Application Repository.