# INTRODUCCIÓN.

## Utilización de PHP.

Con PHP, integrar la actividad dinámica en páginas web es una cuestión muy sencilla:

```
<?php
echo "Hoy es " . date("l") . ". ";
?>
Aquí las últimas noticias.
```

O, también, del modo:

```
Hoy es <?php echo date("l"); ?>. Aquí las últimas noticias.
```

## Utilización de MySQL.

Con PHP podemos hacer llamadas directamente a MySQL sin tener que acceder a la línea de comandos de MySQL.

```
INSERT INTO users VALUES('Ledo', 'Alberto', 'alberto@s2a.es');

SELECT surname, firstame FROM users WHERE email='alberto@s2a.es';
```

## Utilización de Javascript.

Como se ha comentado, javascript se usa cada vez más par ala comunciación asíncrona, el proceso de acceso al servidor web en segundo plano, no hay que volver a cargar la página web en su totalidad par amostrar un nuevo contenido. En lugar de eso, un allamada asíncrona puede incorporar y actualizar un solo elemento en una página web.

```
<script type="text/javascript">
      document.write("Hoy es " + Date());
</script>
```

## Utilización de CSS.

```
<style>
      p {text-align: justify;
         font-family: Helvetica;}
</style>
```

## Ejemplo de comunicación asíncrona.

Comprobar si existe un nombre de usuario en  el sitio cuando un usuario se está registrnado par acrear una nueva cuenta.

1. El servidor genera el código HTML para crear el formualrio web, en el que se pide la información necesaria, como el nombre de usuario.

2. Al mismo tiempo, el servidor añade código Javascript a HTML par amonitorizar el cuadro de entrada de nombre usuario y comprobar dos cosas: si se ha tecleado texto y si la entrada ha dejado de estar seleccioanda debido a que el usuario hizo clic en otro cuadro de entrada.

3. Una vez que se ha introducido el texto y se ha deseleccionado el campo, el código Javascrit pasa, en segundo palno, el nombre de usuario que se escribió, a un script PHP en el servidor web y espera uan respuesta.

4. El servidor web busca el nombre de usuario en la base de datos MySQL y responde a Javascrit en cuanto a si ese nombre ya existe.

5. Javascript entonces coloca una indicación al lado del cuadro de entrada del nombre de usuario par amostrar si el nombre está disponible, tal vez uan marc de verificación verde o un gráfico con una cruz roja.

6. Si el nombre de usuario no está disponible y el usuario sigue enviando el formulario, Javascrit interrumpe la presentación y vuelve a hacer hincapié en que el usuario necesita elegir otro nombre de usuario.

**INTRODUCCIÓN A PHP. LA SINTAXIS.**

PHP es un lenguaje bastante sencillo, que tiene sus orígenes en C y Perl, pero se parece más a Java.

```php
<?php
/* Este es un comenatrio
   de múltiples líneas */
// Este es un comentrio de una línea.
echo "Hello, world" // Imprime: Hello, world.
?>
```

Si solo existe código PHP en un archivo, se puede omitir el cierre '?>'. Esto puede ser una buena práctica, ya que asegurará que no haya exceso de espacios en blanco que se filtren de nuestros archivos PHP, especialmente importante cuando estamos escribiendo código orientado a objetos.

Los comentarios /* ... */ y //, son una excelente manera de eliminar temporalmente una línea de código de un programa que está dando errores. Sirven también para documentar el código.

**El símbolo $.**

En PHP, se debe colocar '$' delante de todas las variables, independientemente de que sean números, cadenas o matrices.

```php
<?php
$mycounter = 1;
$mystring = "Hello";
$myarray = array("Uno", "Dos", "Tres");
?>
```

**Variables.**

Strings.

```php
<?php
$username = "Eduardo Pusent";
$username = 'Elsa Pusent';
echo $username // Imprime: Elsa Punset
?>
```

Las comillas, simles o dobles, indican que estamos ante en una cadena. Veamos un ejemplo de código.

```php
<?php
$username = "Eduardo Punset";
echo $username;
echo "\n";
$current_user = $username;
echo $current_user;
?>
```

El resultado de este código es que debería aparecer dos veces el nombre Eduardo Punset, el primero como resultado del comando echo $username y el segundo como resultado del comando echo $current_user.

Arrays.

El código para crear una matriz está compuesto por el constructor array().

```php
<?php
$team = array('Pedro', 'Juan', 'Miguel');
echo $team[0]; // Imprime: Pedro
echo $team[2]; // Imprime: Miguel
?>
```

Array de dos dimensiones.

```php
<?php
$numbers = array(array('1', '2', '3'),
                 array('4', '5', '6'),
                 array('7', '8', '9'));

echo $numbers[1][2]; // Imprime: 6
?>
```

Los índices de las matrices(punteros de los elementos dentro de una matriz) comienzan en cero, no en uno, por lo que [1] en el comando anterior se refiere a la segunda de la tres matrices, y [2] se refiere a la tercera posición dentro de esa matriz.

**Operadores.**

Operadores de asignación.

El operaodr += suma el valor de la derecha al valor de la varibale de la izquierda, en lugar de reemplazar toralmente el valor de la izquierda, por ejemplo:

```php
<?php
$count = 5;
$count += 1;
echo $count; // Imprime: 6
?>
```

Por lo tanto, `$count += 1` es igual a la declaración de asignación más familiar `$count = count + 1`.

Los operadores de asignación son:

```
=     $j = 15     -->   $j = 15
+=    $j += 5     -->   $j = $j + 5
-=    $j -= 5     -->   $j = $j - 5
*=    $j *= 5     -->   $j = $j * 5
/=    $j /= 5     -->   $j = $j / 5
.=    $j .= $k    -->   $j = $j . $k (Concatenación)
%=    $j %= 4     -->   $j = $j % 4
```

Operadores de comparación.

```
==    es igual a         $j == 4
===   es idéntico a      $j === "978"      (Devuelve true o false)
!==   no es idéntico a   $j !== "1.2e3"    (Devuelve treu o false)
```

Hay que observar la diferencia entre = e ==, el primero es un operador de asiganción y el segundo un operador relacional.

El operador de comparación de triple signo igual '===' no solo verifica si las variables tienen el mismo valor, sino que también confirma que son del mismo tipo antes de devolver true o false.

Operadores lógicos.

| | | |
|---|---|---|
| && | And (y) | $j == 3 && $k == 2 |
| and | Baja prioridad and (y) | $j == 3 and $s == 2 |
| \|\| | Or (o) | $j < 5 \|\| $j > 10 |
| or | Baja prioridad or (o) | $j < 5 or $j > 10 |
| ! | Not (no) | !($j == $k) |
| xor | Or exclusivo | $j xor $k |

Observar que && se puede intercambiar en geenral con and; lo mismo es cierto para || y or. Sin embargo, debido a que and y or tienen un aprioridad menor, debemos evitar usarlos excepto cuando son la única opción, como en la siguiente declaración, en la cual debemos usar el operador or, (|| no se puede usar para forzar la ejecución de una segunda declaración si la priemra falla):

```
<?php
$site = "www.s2a.es";
$html = file_get_contents($site) or die ("No es posible descargar desde $site");
echo $html;
?>
```

**Asignación de valores a variables.**

El siguiente código le dice a PHP que primero aumente el valor de $x (++$x) en 1 y luego pruebe si tiene el valor 10 y, si lo tiene que presente el resultado.

```
<?php
$x = 9;
$y = 5;
if (++$x == 10) echo $x; // Imprime: 10
else echo $y;
?>
```

También podemos incrementar o decrementar la variable después de hacer la comprobación de valor, por ejemplo:

```
<?php
$x = 10;
$y = 5;
if ($x-- == 10) echo $x; //Imprime: 9 (si 'x == 10' restar 1 y luego 'echo $x').
else echo $y;
?>
```

En resumen, una variable se incrementa o decrementa antes de la prueba si el oeprador se coloca antes de la variable, mientras que la variable se incrementa o decrementa después de la prueba si el oerador está situado después de la variable.

**Concatenación de cadenas de caracteres.**

Se usa el oeprador '.' para concatenar cadenas

```
<?php
$msgs = 5;
```

```
echo "Tienes " . $msgs . " mensajes"; // Imprime: Tienes 5 mensajes.
?>
```

Del mismo modo que se puede añadir un valor a un avariable numeŕica con el operador '+=', se puede añadir una cadena a otra mediante '.=', así:

```
<?php
$messages .= $message;
?>
```

En este caso se agrega el valor de $message al valor de $messages, es decir ahora, el contenido de $messages es el que era más el contenido de $message.

**Tipos de cadenas.**

PHP admite dos tipos de cadenas que se denotan por el tipo de comillas que usemos. Si deseamos asignar una cadena literal y preservar el contenido exacto, debemos utilizar comillas simples. Si en el siguiente ejemplo hubiéramos usado comillas dobles, PHP habría intentando evaluar $varible como una variable, como se ve en el último echo.

```
<?php
$info = 'El prefijo de las variables es $ como aquí: $variable';
echo $info;
echo "\n";
$count = 1000;
echo "Eta semana $count personas han visto tu perfil"
?>
```

El código anterior imprime lo siguiente:

```
El prefijo de las variables es $ como aquí: $variable
Eta semana 1000 personas han visto tu perfil
```

La línea de código,

```
echo "Eta semana $count personas han visto tu perfil"
```

ofrece una opción más simple para la concatenación en la que no se necesita usar un punto, o cerrar y reabrir comillas para agregar una cadena a otra. Esto se llama sustitución de variables.

**Caracteres de escape.**

A veces una cadena contiene caracteres con significados especiales que se podrían interpretar de forma incorrecta.

Errror de síntaxis. Aquí, la segunda comilla le dice a PHP que se ha alcanzado el final de la cadena:

```
<?php
echo 'My spelling's atroshus'; /* PHP Parse error: syntax error, unexpected 's'
                                  (T_STRING), expecting ';' or ',' */
?>
```

Corrección. Añadimos una barra invertida para decirle a PHP que trate el caracter de manera literal:

```php
<?php
echo 'My spelling\'s atroshus';
?>
```

IMPORTANTE: Dentro de las comillas simples, solo el apóstrofo de escape (\') y la propia barra invertida (\\) se reconocen como caracteres de escape.

**Comandos de varias líneas.**

Declaración echo de una cadena de varias líneas:

```php
<?php
$author = "Steve Ballmer";

echo "Developers, developers,
developers!
- $author";
?>
```

Asignación de una cadena con varias líneas:

```php
<?php
$author = "Steve Ballmer";

$text = "Developers, developers,
developers!
- $author";
?>
```

PHP también puede tratar una secuencia de varias líneas mediante el operador '<<<', al que normalmente nos referimos como here-document o heredoc, como una forma de especificar un literal de cadena, que preserva los saltos de línea y otros espacios en blanco (incluida la sangría) en el texto. Por ejemplo:

```php
<?php
$author = "Steve Ballmer";

echo <<< _END
¡Developers, developers,
developers!
- $author.
_END;
?>
```

Este código le dice a PHP que presente todo lo que hay entre las dos eqtiquetas _END como si fuese una cadena con comillas dobles (aunque las comillas en un herodoc no necesitan caracteres de escape). Esto significa que es posible, por ejemplo, que un desarrollador escriba secciones enteras de HTML directamente en código PHP y luego reemplace partes dinámicas específicas con variables PHP.

Es importante recordar que el cierre _END debe aparecer justo al comienzo de una nuev alínea y debe ser lo único que haya en esa línea, ni siquiera se permite un comentario (ni siquiera un solo espacio).

El uso de <<<_END ... _END, el contsructor heredoc, no hay que añadir caracteres \n para ebviar un salto de línea. Además se pueden usar las comillas simples y dobles que se quieran dentro de un heredoc, sin tener que anteponer la barra ivertida. En lugar de _END se puede usar cualquier otra etiqueta.

IMPORTNATE: Si cargamos estos ejemplos de varias líneas en un archivo, no se mostrarán varias líneas, ya que todos los navegadores tratan los saltos de línea como si fueran espacios. Sin embargo, si utilizamos la característica View Source del navegador, veremos que los saltos de línea están correctamente situados y que PHP los matiene.

**Tipificación de variables.**

PHP es un lengujae débilmente tipado. Es decir, las variables no se tienen que declarar antes de utilizarlas, además PHP siempre convierte las avriables al tipo requerido por su contexto cuando se accede a ellas. Por ejemplo, poemos cear un núemro de varios dígitos y extraer el énesimo dígito del mismo simplemente suponiendo que es uan cadena.

```
<?php
$number = 12345 * 67890;
echo substr($number, 3, 1);
?>
```

En el momento de la asignación, $number es una variable numérica. Pero en la segunda línea, se hace una llamada a la función substr de PHP, que pide que se devuelva un carácter de $number, comenzando en la cuarta posición. Para hacer esto, PHP convierte $number en una cadena de nueve caracteres, de modo que substr puede acceder a ella y devolver el carácter. Lo mismo se aplica a la conversión de una cadena en un número.

```
<?php
$pi = "3.1415927"; // $pi es un literal de cadena
$radius = 5;
echo $pi * ($radius * $radius); // Imprime: 78.5398175
?>
```

En la práctica, esto significa que no hay que preocuparse demasiado por los tipos de variables. Simplemente asignamos valores que tengan sentido para nosotros, y PHP los convertirá si es necesario. Después, si queremos extraer valores, solo tenemos que pedirlos, por ejemplo, con una declaración echo.

**Constantes.**

Las constantes no deben ir precedidas de $ y solo pueden ser definidas con la función define. Generalmente, se considera una buena práctica usar solo letras mayúsculas para nombres de variables constantes.

```
<?php
Define ("ROOT_LOCATION", "/var/html/www");
$directory = ROOT_LOCATION;
?>
```

Como se ve en el código anterior, por ejemplo, siempre que necesitemos ejecutar código PHP en un servidor diferente con una configuración de carpetas distinta, solo tendremos que cambiar una línea de código.

Constantes predefinidas.

Conocidas como cnstantes mágicas, resultan útiles las siguientes. Para evitar colisiones de nombres, los nombre de constantes mágicas siempre tienen dos guiones bajos al principio y dos al final.

__LINE__

Número de la línea del archivo actual.

__FILE__

Ruta completa y nombre de archivo. Si se usa dentro de include, devuelve el nombre del archivo incluido.

__DIR__

Directoiro del archivo. Si se usa dentro de include, se devuelve el directorio del archivo incluido.

__FUNCTION__

Devuelve el nombre de la función.

__CLASS__

Devuelve el nombre de la clase.

__METHOD__

Devuelve el nombre del método de la clase.

__NAMESPACE__

Esta constante se define en la compilación. Devuelve el nombre del actual espacio de nombres.

Estas variables constantes encuentran un uso práctico en la depuración, cuando se necesita insertar un alíne de código par aver si el flujo del porgrama llega hasta ella.

```
<?php
echo "Esta es la línea" . __LINE__ . "del fichero" . __FILE__;
?>
```

Ejemplos no válidos:

```
<?php
echo <<< _END
Esta es la línea $__LINE__ del fichero $__FILE__ /* Error: Undefined variable:
                                                    __FILE__ */
_END;
?>
```

```
<?php
echo <<< _END
Esta es la línea __LINE__ del fichero __FILE__
_END;
?>
```

**Diferencia entre los comadnos echo y print.**

Existe una alternativa a echo, el comando print. Ambos son bastante similares, pero print es un constructor parecido a una función que admite un solo parámetro y tiene un valor de retorno (que siempre es 1), mientras que echo es puramente un constructor del lenguaje PHP. Dado que ambos son constructores, ninguno requiere paréntesis.

Debido a que echo no se implementa como una función, no se puede usar como parte de una expresión más compleja, mientras que print sí. Aquí hay un ejemplo para determinar si el valor de una variable es TRUE o FALSE, usando print, algo que no se podría realizar de la misma manera con echo.

```php
<?php
$b = 1;
$b ? print "TRUE" : print "FALSE";
?>
```

Se recomienda, no obstante, el uso de echo hasta que el trabajo en el desarrollo PHP pida la necesidad de usar print.

**Ámbito de aplicación de las variables.**

**Variables locales.**

Las variables locales son avriables que se crean dentro de una función y a las que solo se puede acceder mediante esa función. A menos que se haya declarado una variable de otro modo, su alcance se limita al ámbito local; ya se trate de una función o de código fuera de cualquier función, dependiendo de si se creó por primera vez o si se accedió a ella, desde el interior de una función o desde fuera de la misma.

Las variables creadas dentro de una función son locales a esa función, y a las variables creadas fuera de cualquier fucnión solo se puede acceder por un código que no sea una función.

Ejemplo de variable local:

```php
<?php
function longdate($timestamp) {
      $temp = date("l F jS Y", $timestamp);
      return "El día es $temp";
}

echo longdate(1); //Imprime: El día es Thursday January 1st 1970
?>
```

Variable local inacesible:

```php
<?php
$temp = "El día es ";
echo longdate(time());

function longdate($timestamp) { //Imprime: Saturday May 27th 2023
      return $temp . date("l F jS Y", $timestamp); /*PHP Warning: Undefined
                                                     variable $temp*/
}
?>
```

Debido a que $temp no se ha creado dentro de la función longdate no se le ha pasado como parámetro, longdate no puede acceder a ella. Por tanto, este fragmento de código solo produce la fecha, no el texto precedente.

Existen soluciones al ejemplo anterior. Por ejemplo, referirse a $temp dentro de su ámbito local como aquí:

```php
<?php
$temp = "El día es ";
```

```
echo $temp . longdate(time());

function longdate($timestamp) {
      return date("l F jS Y", $timestamp);
}
?>
```

También se puede pasar $temp como argumento a longdate() para solucionar el problema:

```
<?php
$temp = "El día es ";
echo $temp . longdate($temp, time());

function longdate($text, $timestamp) {
      return $text . date("l F jS Y", $timestamp);
}
?>
```

**Variable globales.**

Hay casos en los que se necesita que una variable tenga alcance global, porque se desea que cualquier parte de código pueda acceder a ella. Para ello se antepone a la variable la palabra clave global, por ejemplo:

```
      global $is_logged_in;
```

**Variables estáticas.**

Cuando queramos que una variable sea local, pero queremos mantener su valor la próxima vez que se llame a la función, por ejemplo para un contador que rastree cuántas veces se llama a la función, declaramos las variable como estática:

```
<?php
function test() {
      static $count = 0;
      echo $count;
      $count++;
?>
```

IMPORTANTE: Aquí, la próxima vez que se llame a la función, como ya se ha declarado $count en la primera llamada, se salta la priemra línea de la función. A continuación, muestra el valor incrementado previamente de $count, antes de que la variable vuelva a incrementarse.

A las varibles estáticas no se les puede asiganr el valro de una expresión, por ejemplo, las siguientes declaraciones no serían válidas:

```
<?php
static $int = 1+2; // No permitido.
static $sqrt = sqrt(144); // No permitido.
?>
```

**Variables superglobales.**

Las variables superglobales las proporciona el entorno PHP. Son globales dentro del programa accesibles desde todas partes. Estas variables superglobales contienen información útil sobre el programa en ejecución y su entorno. Están estructuradas como matrices asociativas. Son las siguientes:

```
$GLOBALS
$_SERVER
$_GET
$_POST
$_COOKIE
$_SESSION
$_REQUEST
$_ENV
```

Ejemplo de uso:

```
$came_from = $_SERVER['HTTP_REFERER'];
```

Aquí se devuelve la URL de la página que remitió al usuario a la página web en la que se encuentra.

**$GLOBALS.**

En PHP, la variable superglobal $GLOBALS es un arreglo asociativo que contiene todas las variables globales disponibles en el ámbito de ejecución del script. Cada elemento en el arreglo $GLOBALS es una referencia a una variable global, donde el nombre de la variable es la clave del arreglo y su valor es el valor de la variable.

La ventaja de utilizar $GLOBALS es que permite acceder y manipular variables globales desde cualquier lugar dentro del script, incluso dentro de funciones o métodos. Esto evita tener que pasar las variables como argumentos en funciones o declararlas como globales dentro de cada función en la que se necesitan.

Aquí hay un ejemplo para ilustrar cómo se utiliza $GLOBALS:

```php
<?php
$globalVariable = 10;

function testFunction() {
    $localVariable = 20;

    // Acceder a la variable global $globalVariable utilizando $GLOBALS
    echo $GLOBALS['globalVariable']; // Imprime 10
    echo "\n";

    // Modificar el valor de la variable global $globalVariable
    $GLOBALS['globalVariable'] = 30;

    // Acceder a la variable local $localVariable
    echo $localVariable; // Imprime 20
}

testFunction();

// Acceder a la variable global modificada fuera de la función
echo $globalVariable; // Imprime 30
?>
```

En este ejemplo, $GLOBALS['globalVariable'] se utiliza para acceder a la variable global $globalVariable desde dentro de la función testFunction(). También se muestra cómo se puede modificar el valor de la variable global utilizando $GLOBALS. Fuera de la función, se puede acceder a la variable global modificada directamente como $globalVariable.

Es importante tener en cuenta que el uso de variables globales puede dificultar el mantenimiento del código y hacerlo menos legible. Por lo tanto, se recomienda utilizarlas con precaución y considerar alternativas como pasar variables como argumentos de función o utilizar variables locales.

## EXPRESIONES Y CONTROL DE FLUJO EN PHP.

### TRUE y FALSE.

El valor generado por una expresión, puede ser un número una cadena o un valor booleano.

```php
<?php
echo "a: [" . TRUE . "]<br>"; // Imprime: a: [1] <br>
echo "b: [" . FALSE . "]<br>"; // Imprime: b: [] <br>
echo "c: [" . (1==0) . "]<br>"; // Imprime: c: [] <br>
echo "d: [" . (1==1) . "]<br>"; // Imprime: d: [1] <br>
?>
```

En PHP, TRUE y FALSE son constantes predefinidas. TRUE tiene el valor de '1', FALSE tiene el valor de NULL, otra constante predefinida que no significa nada. En otros lenguajes FALSE puede definirse como 0 o incluso como -1.

### Literales y variables.

```php
<?php
$myvar = "Variable";

echo "a: " . 73 . " <br>"; // Imprime: a: 73 <br> [Literal numérico]
echo "b: " . "Hola" . " <br>"; // Imprime: b: Hola <br> [Literal cadena]
echo "c: " . $myvar . " <br>"; // Imprime: c: Variable <br> [Variable cadena]
?>
```

### Operador de ejecución y de concatenación.

El operador comillas, ajecuta el contenido de las comillas, por ejemplo:

```php
<?php
echo `ls -lha`;
?>
```

Salida:

```
total 20K
drwxr-xr-x 2 n7rc n7rc 4,0K may 25 09:59 .
drwxr-xr-x 3 n7rc n7rc 4,0K may 25 07:50 ..
-rw-r--r-- 1 n7rc n7rc  104 may 25 02:19 00_hello_world.php
-rw-r--r-- 1 n7rc n7rc  449 may 27 03:32 01_varibales.php
-rw-r--r-- 1 n7rc n7rc 3,7K may 27 16:23 02_operadores.php
```

El operador '.' es un operador de concatenación:

```php
<?php
$var1 = "Hola, ";
$var2 = "bienvenido a estos apuntes.";

echo $var1 . $var2; // Imprime: Hola, bienvenido a estos apuntes.
?>
```

### La declaración if.

El contenido de la condición if puede ser cualquier expresión PHP válida, incluidas pruebas de igualdad, expresiones de comparación, pruebas para 0 y NULL, e incluso funciones integradas o alguna función que se escriba.

Las acciones a llevas a cabo cuando una condición if es TRUE se colocan generalmente entre llaves '{}'. Se puede ignorar el uso de llaves si solo se tiene que ejecutar una sola declaración.

```php
<?php
$bank_balance = 1500;

if ($bank_balance < 1000) { // Se evalúa como FALSE.
      $money = 1000;
      $bank_balance += $money;
}

echo $bank_balance;
?>
```

**La declaración elseif.**

Hay ocasiones en las que queremos que ocurran una serie de posibilidades diferentes, según una secuencia de condiciones. Podemos lograrlo con la expresión 'elseif', tenemos así una construcción completa if...elseif...else. Se pueden insertar tantos elseif como se quieran.

```php
<?php
if ($page == "Home") echo "Has seleccionado Home";
elseif ($page == "About") echo "Has seleccionado About";
elseif ($page == "Login") echo "Has seleccionado Login";
else echo "Selección no reconocida";
?>
```

**Otros bucles y expresiones de ontrol de flujo.**

La declaración switch, el operador ternario '?', los bucles while y do-while, el bucle for y las palabras reservadas 'break' y 'continue' operan y tienen una sintaxis similar a la usado en lenguaje C.

**Conversión implícita y explícita.**

En el siguiente código, las entradas a la división son enteros; por defecto, PHP convierte la salida a punto flotante para obtener un valor preciso. A esto se le llama conversión implícita:

```php
<?php
$a = 56;
$b = 12;
$c = $a / $b;

echo $c; // Imprime: 4.6666666666667.
?>
```

¿Y si hubiéramos querido que $c fuese un número entero, entonces hacemos lo siguiente:

```php
<?php
$a = 56;
$b = 12;
$c = (int) ($a / $b);

echo $c; // Imprime: 4.
?>
```

El código anterior realiza una conversión explícita. Se pueden convertir explícitamente variables y literales a los tipos que se muestran a continuación:

| | |
|---|---|
| (int) o (integer) | Convierte a entero. |
| (bool) o (boolean) | Convierte a booleano. |
| (float) o (double) o (real) | Convierte a un número en punto flotante. |
| (string) | Convierte a una cadena. |
| (array) | Convierte a una matriz. |
| (object) | Convierte a un objeto. |

**FUNCIONES EN PHP.**

La sintaxis general de una función es la siguiente:

```
function nombre_función([parámetros[, ...]]) {
      //sentencias
}
```

Los nombres de las funciones no distinguen entre mayúsculas y minúsculas por lo que todas las cadenas que se muestran a continuación pueden referirse a la función imprimir: PRINT, print y PrInT.

Ejemplo de creación de una función genérica:

```php
<?php
echo fix_names("ALBERTO", "juan", "PeDRo"); // Imprime: Alberto Juan Pedro

function fix_names($n1, $n2, $n3) {
      $n1 = ucfirst(strtolower($n1));
      $n2 = ucfirst(strtolower($n2));
      $n3 = ucfirst(strtolower($n3));
return $n1 . " " . $n2 . " " . $n3;
}
?>
```

Ejemplo de creación de una función que devuelve un array:

```php
<?php
$names = fix_names("ALBERTO", "juan", "PeDRo");
echo $names[0] . " " . $names[2] . " " . $names[1]; // Imprime: Alberto Pedro
Juan

function fix_names($n1, $n2, $n3) {
      $n1 = ucfirst(strtolower($n1));
      $n2 = ucfirst(strtolower($n2));
      $n3 = ucfirst(strtolower($n3));
return array($n1, $n2, $n3);
}
?>
```

Este método tiene la ventaja de mantener los tres nombres separados, en lugar de concatenarlos formando una sola cadena, por lo que se puede hacer referencia a cualquier nombre sin estar obligado a extraer ninguno de los otros dos nombres.

**Paso de argumentos por referencia.**

La referencia a una variable se denota del siguiente modo con el operador '&':

```
$myvar2 = &$myvar1
```

```php
<?php
$myvar1 = "Díaz";
$myvar2 = &$myvar1;

echo $myvar2; // Imprime: Díaz
?>
```

En PHP está obsoleto el paso de argumentos a una función por referencia, es decir, por ejemplo 'increment(&$myvar);' está obsoleto. Sin embargo, dentro de

la definición de función se puede continuar accediendo a los argumentos por referencia.

Se puede reescribir el ejemplo anterior para pasar referencias a todos los parámetros, y entonces la función puede modificarlos directamente.

```
<?php
$a1 = "ALBERTO";
$a2 = "juan";
$a3 = "PeDRo";

echo $a1 . " " . $a2 . " " . $a3; // Imprime: ALBERTO juan PeDRo
fix_names($a1, $a2, $a3);
echo $a1 . " " . $a2 . " " . $a3; // Imprime: Alberto Juan Pedro

function fix_names(&$n1, &$n2, &$n3) { // Paso de argumentos por referecnia.
      $n1 = ucfirst(strtolower($n1));
      $n2 = ucfirst(strtolower($n2));
      $n3 = ucfirst(strtolower($n3));
}
?>
```

IMPORTANTE: En lugar de pasar las cadenas directamente a la función, primero se asignan a variables y se imprimen, para ver sus valores "antes". Luego se llama a la función; dentro de la función se ha colocado el "operador" '&' delante de cada parámetro que se debe pasar por referencia. Entonces las variables $n1, $n2, y $n3 se adjuntan a "hilos" que conducen a los valores de $a1, $a2 y $a3; es decir, hay un grupo de valores, pero se permite el acceso a estos valores a dos conjuntos de nombres de variables. Por lo tanto la función fix_names() solo tiene que asignar nuevos valores a $n1, $n2 y $n3 para actualizar los valores de $a1, $a2 y $a3. Como se puede observar, ambas declaraciones de echo utilizan solo los valores de $a1, $a2 y $a3.

**Devolución en variables globales.**

La mejor manera de que una función tenga acceso a una variable creada externamente es declarando que tiene acceso global desde dentro de la función. La palabra clave 'global' segida del nombre de la variable le da a cualquier parte del código acceso completo a ella.

```
<?php
$a1 = "ALBERTO";
$a2 = "juan";
$a3 = "PeDRo";

echo $a1 . " " . $a2 . " " . $a3; // Imprime: ALBERTO juan PeDRo
fix_names($a1, $a2, $a3);
echo $a1 . " " . $a2 . " " . $a3; // Imprime: Alberto Juan Pedro

function fix_names() {
      global $a1; $a1 = ucfirst(strtolower($a1));
      global $a2; $a2 = ucfirst(strtolower($a2));
      global $a3; $a3 = ucfirst(strtolower($a3));
}
?>
```

Ahora no hay que pasar parámetros a la función, y esta no tiene que aceptarlos. IMPORTANTE: Una vez declaradas, estas variables conservan el acceso global y están disponibles para el resto del porgrama, incluídas sus fucniones.

**Inclusión y requisión de archivos.**

La declaración include le dice a PHP que obtenga un archivo particular y cargue todo su contenido. Es como pegar el archivo a incluir en el archivo en curso, en el punto donde se haya insertado include.

```
inlcude "library.php";
```

Cada vez que se emite la directiva include se vuelve a incliur el archivo solicitado, incluso si ya se ha insertado. Supongamos que se incluye library.php, pero también se incluye otra biblioteca que a su vez incluye también library.php; se ha incluído, sin querer, dos veces library.php. Este emitirá mensajes de error, dado que se está intentando definir la misma constante o función varias veces. En este caso hay que usar, en su lugar, include_once, así se ignorará cualquier otro intento de incluir el mismo archivo.

```
include_once "library.php";
```

En general es mejor el uso de include_once que el uso de include.

Con el uso de include e include_once la ejecución del porgrama continúa incluso si no se encuentra el archivo solicitado. Cuando sea absolutamente imprescindible incluir un archivo se ha de usar require, y por las mismas razones que el uso de include_once, se aconseja el uso de require_once:

```
require "functions.php";
require_once "functions.php";
```

**phpinfo() y phpversion.**

```php
<?php
echo phpversion(); // Imprime: 8.2.5
?>

<?php
echo phpinfo; // Imprime: Información importante sobre PHP y sus módulos.
?>
```

**OBJETOS EN PHP.**

Al crear un porgrma para utilizar objetos, se necesita diseñar una combinación de datos y código llamada clase. Cada nuevo objeto basado en esta clase se llama instancia de la clase.

A los datos asociados a un objeto se les denomina propiedades; las funciones que utiliza se denominan métodos. Así pues, al definir una clase se deben proporcionar los nombres de sus propiedades y el código para sus métodos.

IMPORTANTE: Cuando se crea un objeto, es mejor utilizar la encapsulación o escribir una clase de tal forma que solo se pueden utilizar sus métodos par amanipular sus propiedades. En otras palabras, negar el acceso directo de código externo a sus datos. Los métodos que se proporcionan se conocen como interfaz del objeto.

**Declaración de clases.**

Antes de poder utilizar un objeto se debe definir una clase con la palabra reservada class. La definición de una clase contiene el nombre de la clase (que distingue entre mayúsculas y minúsculas), sus propiedades y sus métodos.

```php
<?php
$object = new User;
print_r($object);

class User {
      public $name, $password;

      function save_user() {
            echo "El código para salvar el usuario viene aquí.";
      }
}
?>
```

print_r pide PHP que imprima información sobre una variable, de forma que la pueden interpretar las personas (la r significa humman-readable). En este caso el nuevo objeto muestra lo siguiente (salida por consola):

```
User Object
(
      [name] =>
      [password] =>
)
```

o en un navegador, que se vería del modo:

```
User Object ( [name] => [password] => )
```

La salida dice que $object es un objeto definido por el usuario que tiene las propiedades name y password.

**Creación de objetos.**

```php
<?php
$object = new User;
$temp = new User('name', 'password');
?>
```

Existen dos formas de crear un nuevo objeto. Al crear el objeto $object, sencillamente asiganmos un objeto a la clase User. En la segunda variante, $temp, pasamos parámetros a la llamada.

El código (toda esta explicación se detallará más adelante):

```
$object = new User('name', 'password');
```

en PHP crea una nueva instancia de la clase User y asigna esa instancia a la variable $object.

En este caso, se está utilizando el constructor de la clase User para inicializar la instancia con dos argumentos: 'name' y 'password'. El constructor es un método especial dentro de una clase que se ejecuta automáticamente cuando se crea un nuevo objeto de esa clase.

El código asume que hay una clase llamada User definida en el código PHP o incluida desde otro archivo. La clase User puede tener propiedades y métodos que definen el comportamiento y las características de un usuario en el sistema. Al pasar 'name' y 'password' como argumentos al constructor, se establecen los valores iniciales para esos atributos de la instancia de la clase User.

Aquí hay un ejemplo simplificado de cómo se podría definir la clase User y su constructor:

```php
<?php
class User {
    private $name;
    private $password;

    public function __construct($name, $password) {
        $this->name = $name;
        $this->password = $password;
    }

    // Otros métodos y propiedades de la clase User...
    // Getter y setter para acceder a los atributos name y password

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getPassword() {
        return $this->password;
    }

    public function setPassword($password) {
        $this->password = $password;
    }
}
?>
```

En este ejemplo, la clase User tiene dos atributos privados ($name y $password) y un constructor que asigna valores a estos atributos. También se proporcionan métodos getter y setter para acceder y modificar los valores de los atributos respectivamente.

Al ejecutar $object = new User('name', 'password');, se crea una nueva instancia de la clase User con los valores 'name' y 'password', y se asigna esa instancia a la variable $object. A partir de ese momento, se puede acceder a los métodos y propiedades de la instancia a través de la variable $object, como $object->getName() para obtener el nombre o $object->setPassword('newPassword') para modificar la contraseña.

**Acceso a objetos.**

```php
<?php
$object = new User;
print_r($object);
echo "\n";

$object->name = "Juan";
$object->password = "qwerty";
print_r($object);
echo "\n";

$object->save_user();

class User {
      public $name, $password;
      function save_user() {
            echo "El código para salvar el usuario viene aquí.";
      }
}
?>
```

Como se puede ver en la sintaxis para acceder a la propiedad de un objeto se usa:

```
$object->propiedad
```

del mismo modo, para llamar a un método se usa:

```
$object->método
```

Salida:

```
User Object
(
      [name] =>
      [password] =>
)

User Object
(
      [name] => Juan
      [password] => qwerty
)

El código para salvar el usuario viene aquí.
```

**Clonación de objetos.**

Una vez creado un objeto, se pasa por referencia cuando se pasa como parámetro. Es decir, hacer la asignación de objetos no copia los objetos en su totalidad.

```php
<?php
$object1 = new User();
$object1->name = "Sonia";
$object2 = $object1;
$object2->name = "María";
echo "object1 name = " . $object1->name . "\n";
echo "object2 name = " . $object2->name;
class User {
      public $name;
}
?>
```

Se crea el objeto $object1 y se le asigna el valor Sonia a la propiedad name. Luego se crea $object2, se le asigna el valor de $object1 y el valor María a la propiedad name de $object2, así esté código tiene como salida:

```
object1 name = María
object2 name = María
```

Cambiar la propiedad name de $object2 también la cambia en $object1, es decir tanto $object2 como $object1 se refieren al mismo objeto.

Se puede utilizar el operador clone para cambiar la propiedad name de $object2 sin cambiar la propiedad de $object1, como sigue:

```php
<?php
$object1 = new User();
$object1->name = "Sonia";
$object2 = clone $object1;
$object2->name = "María";
echo "object1 name = " . $object1->name . "\n";
echo "object2 name = " . $object2->name;
class User {
      public $name;
}
?>
```

La salida de este código es:

```
object1 name = Sonia
object2 name = María
```

**Constructores.**

Al crear un objeto nuevo, se puede pasar una lista de argumentos a la clase a la que se llama, como en [*] del siguiente ejemplo. Para ello se utiliza la función con nombre __construct, el constructor de la clase:

```php
<?php
$object = new User('Santiago', 'qwerty'); // [*]
print_r ($object);


class User {
    private $name;
    private $password;

    public function __construct($name, $password) {
        $this->name = $name;
```

```
        $this->password = $password;
    }

    // Otros métodos y propiedades de la clase User...
    // Getter y setter para acceder a los atributos name y password

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getPassword() {
        return $this->password;
    }

    public function setPassword($password) {
        $this->password = $password;
    }
}
?>
```

En resumen (sobre el constructor):

En PHP, un constructor es un método especial dentro de una clase que se ejecuta automáticamente cuando se crea una nueva instancia (objeto) de esa clase. Su objetivo principal es inicializar el objeto y configurar cualquier estado inicial que pueda necesitar.

Un constructor se define utilizando el nombre reservado __construct(). Cuando se crea un nuevo objeto utilizando la palabra clave new, PHP busca automáticamente el método __construct() dentro de la clase correspondiente y lo ejecuta.

Aquí hay un ejemplo básico de una clase User con un constructor:

```
<?php
class User {
    private $name;
    private $email;

    public function __construct($name, $email) {
        $this->name = $name;
        $this->email = $email;
    }
    // Otros métodos y propiedades de la clase User...
}
?>
```

En este ejemplo, la clase User tiene dos propiedades privadas: $name y $email. El constructor __construct() acepta dos parámetros, $name y $email, que se utilizan para inicializar las propiedades correspondientes de la instancia de User.

Cuando se crea un nuevo objeto de la clase User, se pueden pasar los valores deseados al constructor para establecer el estado inicial del objeto:

```
<?php
$user = new User('admin', 'admin@s2a.es');
?>
```

En este caso, se crea un nuevo objeto $user de la clase User y se pasan los valores 'admin' y 'admin@s2a.es' al constructor. El constructor asigna esos valores a las propiedades $name y $email del objeto $user.

El constructor se ejecuta automáticamente en el momento de la creación del objeto, y se puede utilizar para realizar cualquier configuración inicial necesaria, como la asignación de valores predeterminados, la conexión a una base de datos u otras tareas relacionadas con la inicialización del objeto.

Es importante tener en cuenta que en PHP también se puede tener un constructor vacío, es decir, un constructor sin ningún parámetro. En caso de no definir un constructor explícitamente, PHP proporciona uno implícito vacío por defecto.

**Palabra reservada this.**

En PHP, la palabra reservada this se refiere a la instancia actual de la clase en la que se está utilizando. Se utiliza dentro de una clase para acceder a las propiedades de esa clase.

La palabra this actúa como un pronombre que apunta al objeto actual. Permite acceder y manipular las propiedades y métodos de la clase dentro de sus propios métodos.

Aquí tienes un ejemplo para ilustrar su uso:

```php
<?php
class Person {
    private $name;

    public function setName($name) {
        $this->name = $name;
    }

    public function sayHello() {
        echo "Hola, mi nombre es " . $this->name;
    }
}

$person = new Person();
$person->setName('Santiago Auserón');
$person->sayHello();
?>
```

En este ejemplo, la clase Person tiene una propiedad privada $name y dos métodos. El método setName() establece el valor de la propiedad $name utilizando la palabra this para hacer referencia a la instancia actual de la clase. El método sayHello() muestra un mensaje que incluye el valor de la propiedad $name utilizando $this->name.

Cuando se crea un nuevo objeto de la clase Person y se llama a los métodos correspondientes, la palabra this se refiere a la instancia actual del objeto. En este caso, $this->name se refiere a la propiedad $name del objeto $person.

Es importante tener en cuenta que la palabra this solo puede ser utilizada dentro del contexto de una clase. Fuera de una clase, no tiene ningún significado y generará un error si se intenta utilizar.

**Destructores.**

También existe la posibilidad de crear métodos destructores. Esta opción utilidad cuando el código ha hecho la última referencia a un objeto o cuando un script llega al final. El destructor puede hacer labores de limpieza, como liberar la conexión a una base de datos o a algún otro recurso que se haya reservado dentro de la clase. Si se ha reservado un recurso dentro de una clase, hay que liberarlo. Muchos problemas en el conjunto del sistema los causan programas que ocupan recursos y se olvida liberarlos.

En PHP, un método destructor es un método especial dentro de una clase que se ejecuta automáticamente cuando un objeto de esa clase se destruye o se elimina explícitamente. Su propósito principal es liberar recursos o realizar tareas de limpieza antes de que el objeto sea eliminado de la memoria.

El método destructor se define utilizando el nombre reservado __destruct(). Cuando se destruye un objeto, ya sea mediante la liberación explícita con unset() o cuando ya no hay referencias a él, PHP busca automáticamente el método __destruct() dentro de la clase correspondiente y lo ejecuta.

Aquí hay un ejemplo básico de una clase File con un método destructor:

```php
<?php
class File {
    private $handle;

    public function __construct($filename) {
        $this->handle = fopen($filename, 'r');
    }

    public function __destruct() {
        fclose($this->handle);
    }
    // Otros métodos y propiedades de la clase File...
}
?>
```

En este ejemplo, la clase File tiene una propiedad privada $handle, que representa el descriptor de archivo abierto. En el constructor __construct(), se abre el archivo especificado y se asigna el descriptor a la propiedad $handle.

El método destructor __destruct() se encarga de cerrar el archivo utilizando fclose($this->handle). Esto garantiza que el archivo se cierre correctamente antes de que el objeto File sea destruido.

Cuando se destruye un objeto File, ya sea explícitamente con unset($file) o cuando el objeto ya no tiene referencias, el método __destruct() se ejecuta automáticamente, lo que cierra el archivo asociado.

Es importante tener en cuenta que el método destructor no se invoca directamente por el programador, sino que es manejado automáticamente por el sistema cuando el objeto se destruye. Además, PHP permite tener solo un método destructor por clase.

**Ámbito de las propiedades y métodos.**

PHP proporciona tres palabras clave para controlar el campo de acción de las propiedades y métodos (miembros):

Public

Los miembros con clave pública se pueden referenciar en cualquier lugar, incluso en otras clases e instancias de objeto. Este es el valor por defecto.

Protected

Estos miembros se pueden referenciar solo por los métodos de clase del objeto y los de cualquier subclase.

Privated

Estos miembros se pueden referenciar solo por métodos dentro de la misma clase, no por subclases.

¿Cómo decidir que se necesita usar?

Usar public cuando el código externo debe acceder a ese miembro y la extensión de clases también debe heredarlo.

Usar protected cuando el código externo no debe acceder al miembro, pero la extensión de clases debe heredarlo.

Usar private cuando el código externo no debe acceder al miembro y la extensión de clases tampoco debe heredarlo.

Ejemplo de uso:

```
<?php
class Example {
      public $age = 47;
      protected $useracount;

      private function admin() {
            // Sentencias
      }
}
?>
```

**Operador '::' y miembros estáticos.**

En PHP, el operador '::', conocido como operador de resolución de ámbito o operador de doble dos puntos, se utiliza para acceder a constantes, propiedades y métodos estáticos de una clase, sin necesidad de crear una instancia (objeto) de la misma.

Un **miembro estático** es una propiedad o método que pertenece a la clase en sí misma, en lugar de pertenecer a una instancia específica de la clase. Esto significa que el miembro estático es compartido por todas las instancias de la clase y se puede acceder a él sin necesidad de crear un objeto.

Para declarar un miembro estático en una clase, se utiliza la palabra clave static antes de la declaración. Los miembros estáticos se asocian con la clase en la que se definen y se accede a ellos utilizando el operador de resolución de ámbito ::.

Aquí tienes un ejemplo que muestra cómo se declaran y utilizan miembros estáticos en PHP:

```php
<?php
class MyClass {
    public static $staticProperty = 'Hello';

    public static function staticMethod() {
        return 'World';
    }
}

echo MyClass::$staticProperty; // Imprime "Hello"
echo "\n";

echo MyClass::staticMethod(); // Imprime "World"
echo "\n";
?>
```

En este ejemplo, la clase MyClass tiene un miembro estático llamado $staticProperty y un método estático llamado staticMethod(). Puedes acceder al miembro estático utilizando la sintaxis Clase::$staticProperty y al método estático utilizando la sintaxis Clase::staticMethod() sin necesidad de crear una instancia de la clase.

Los miembros estáticos son útiles cuando se desea compartir datos o funcionalidades comunes entre todas las instancias de una clase. Algunos casos de uso comunes para los miembros estáticos incluyen:

- Variables de contador o contadores globales utilizados por todas las instancias de una clase.
- Métodos utilitarios que no requieren el estado de una instancia en particular.
- Constantes compartidas por todas las instancias de una clase.

Es importante tener en cuenta que los miembros estáticos no tienen acceso a las propiedades y métodos no estáticos de la clase, ya que estos pertenecen a una instancia específica. Del mismo modo, los miembros no estáticos no pueden acceder directamente a los miembros estáticos de la clase.

Los métodos estáticos son útiles para realizar acciones relacionadas con la clase misma, pero no con instancias específicas de la clase.

En resumen: La declaración de miembros de una clase como static los hace accesibles sin una instanciación de la clase. A una propiedad declarada como estática no puede acceder directamente una instancia de clase, pero un método estático sí puede hacerlo.

El **operador ::** se utiliza en la siguiente sintaxis:

```
Clase::miMetodo();
Clase::$miPropiedad;
Clase::MI_CONSTANTE;
```

Aquí tienes una descripción de cada uso del operador :::

Acceder a un método estático: Permite llamar a un método declarado como estático en una clase sin necesidad de crear una instancia de la clase. Se utiliza la sintaxis Clase::miMetodo();.

Acceder a una propiedad estática: Permite acceder a una propiedad declarada como estática en una clase sin necesidad de crear una instancia de la clase. Se utiliza la sintaxis Clase::$miPropiedad;.

Acceder a una constante: Permite acceder a una constante declarada dentro de una clase. Las constantes son miembros de clase que tienen un valor constante y no pueden ser modificados. Se utiliza la sintaxis Clase::MI_CONSTANTE;.

Aquí un ejemplo para ilustrar el uso del operador :::

```php
<?php
class MyClass {
    public static $myProperty = 'Hello,';
    const MY_CONSTANT = 'World';

    public static function myMethod() {
        echo self::$myProperty . ' ' . self::MY_CONSTANT;
    }
}

echo MyClass::$myProperty; // Imprime "Hello"
echo "\n";
echo MyClass::MY_CONSTANT; // Imprime "World"
echo "\n";
MyClass::myMethod(); // Imprime "Hello World"
?>
```

En este ejemplo, la clase MyClass tiene una propiedad estática $myProperty, una constante MY_CONSTANT y un método estático myMethod(). El operador :: se utiliza para acceder a la propiedad estática y a la constante, así como para llamar al método estático sin crear una instancia de la clase.

Es importante tener en cuenta que el operador :: solo se puede utilizar con elementos estáticos de una clase, es decir, aquellos elementos que pertenecen a la clase en sí y no a una instancia específica del objeto.

**Declaración de constantes y palabra reservada self.**

De la misma manera que se pueden crear constantes globales con la función define, se pueden definir constantes dentro de clases. Las constantes se pueden referenciar directamente mediante el uso de la palabra clave self y el operador ::. Como anteriormente hemos explicado, el siguiente código llama a la clase directamente utilizando el operador ::, sin crear antes una instancia del mismo.

```php
<?php
Translate::lookup(); // lookup() es un método estático y se puede llamar así.

class Translate {
    const ESPAÑOL = 0;
    const INGLES = 1;
    // ...

    static function lookup() {
        echo self::ESPAÑOL;
    }
}
?>
```

Mediante la palabra reservada self se accede directamente a una porpiedad estática o una constante dentro de una clase.

**Herencia.**

Con la palabra clave extends podemos crear una subclase de otra. Se puede tomar una clase similar a la que se necesita, extenderla a una subclase y solo modificar las partes que son diferentes. Una subclase amplia una clase añadiendo propiedades y/o métodos adicionales. Por ejemplo:

```php
<?php
$object = new Subscriber();
$object->name  = "Sonia";
$object->password  = "qwerty";
$object->phone  = "012 345 678";
$object->email  = "sonia@s2a.es";
$object->display();

class User {
    public $name, $password;

    public function save_user() {
        // Código de save_user().
    }
}
class Subscriber extends User { // Definición de subclase.
    public $phone, $email;

    public function display() {
        echo "Nombre: " . $this->name . "\n";
        echo "Password: " . $this->password . "\n";
        echo "Teléfono: " . $this->phone . "\n";
        echo "email: " . $this->email . "\n";
    }

}
?>
```

**Palabra reservada 'parent'.**

La palabra reservada parent se utiliza dentro de una clase para hacer referencia a la clase padre o superclase de la clase actual. Se utiliza para acceder a los métodos o propiedades de la clase padre desde la clase hija o subclase.

La palabra parent se utiliza en la siguiente sintaxis:

Acceder a un método o propiedad de la clase padre:

```
parent::metodo()
parent::$propiedad.
```

Aquí hay un ejemplo para ilustrar el uso de la palabra reservada parent:

```php
<?php
class ParentClass {
    protected $property = 'Hola desde parent';

    public function greet() {
        echo $this->property;
    }
}

class ChildClass extends ParentClass {
    protected $property = 'Hola desde child';
```

```
    public function greet() {
        echo $this->property;
        echo "\n";
        echo parent::$property;
    }
}

$child = new ChildClass();
$child->greet();
?>
```

En este ejemplo, hay una clase padre llamada ParentClass y una clase hija llamada ChildClass. Ambas clases tienen una propiedad protegida llamada $property. La clase padre tiene un método greet() que muestra el valor de la propiedad $property de la clase padre, mientras que la clase hija también tiene su propia implementación del método greet() que muestra el valor de la propiedad $property de la clase hija y luego utiliza parent::$property para acceder al valor de la propiedad de la clase padre.

Cuando se crea un objeto de la clase ChildClass y se llama al método greet(), se muestra el valor de la propiedad $property de la clase hija y luego se accede al valor de la propiedad de la clase padre utilizando parent::$property.

La palabra parent es útil cuando se necesita acceder a métodos o propiedades de la clase padre desde la clase hija. Esto permite heredar y extender el comportamiento de la clase padre en la clase hija, agregando o modificando funcionalidades según sea necesario.

Es importante tener en cuenta que la palabra parent solo puede ser utilizada dentro del contexto de una clase hija que hereda de una clase padre. Fuera de ese contexto, no tiene ningún significado y generará un error si se intenta utilizar.

**Constructores de subclases.**

Cuando se usan métodos constructores de subclases, PHP no llama automáticamente al método constructor de la clase padre. Para ejecutar el código de inicialización completo (constructor de la calse padre + constructor de la subclase), el método constructor de la subclase siempre debe llamar a los constructores padre. Por ejemplo:

```
<?php
$object = new Tiger();

echo "Los tigres tienen ...\n";
echo "Pelo: " . $object->fur . "\n";
echo "Rayas: " . $object->stripes . "\n";

class Wildcat {
    public $fur;

    function __constructor() {
        $this->fur = "TRUE";
    }
}
class Tiger extends Wildcat { // Definición de subclase.
    public $stripes;
```

```
    function __constructor() {
        parent::__constructor(); // Llamáda al constructor de la clase padre.
        $this->stripes = "TRUE";
    }
}
?>
```

**Palabra clave final.**

La palabra clave final se utiliza para indicar que una clase, método o propiedad no puede ser heredado, sobreescrito o reemplazado por clases hijas o subclases.

La palabra clave final se coloca antes de la declaración de la clase, método o propiedad para marcarla como final. Esto evita que otras clases extiendan la clase final, sobre escriban los métodos finales o modifiquen las propiedades finales.

Aquí hay algunos ejemplos para ilustrar el uso de la palabra clave final:

Clase final:

```
<?php
final class FinalClass {
    // Contenido de la clase final
}
?>
```

En este ejemplo, la clase FinalClass se declara como final. Esto significa que ninguna otra clase puede heredar de ella.

Método final:

```
<?php
class ParentClass {
    final public function finalMethod() {
        // Contenido del método final
    }
}

class ChildClass extends ParentClass {
      public function finalMethod() {
      // Error: No se puede sobre escribir un método final
       }
}
?>
```

En este ejemplo, el método `finalMethod()` de la clase `ParentClass` se declara como final. Esto significa que ninguna subclase puede sobre escribir ese método.

Propiedad final:

```
<?php
class ParentClass {
    final public $finalProperty = 'Valor final';
}
?>
```

En este ejemplo, la propiedad `$finalProperty` de la clase `ParentClass` se declara como final. Esto significa que ninguna subclase puede modificar el valor de esa propiedad.

El uso de la palabra clave final permite definir componentes que no pueden ser modificados en clases hijas o subclases, brindando mayor control y evitando posibles modificaciones o extensiones inesperadas.

Es importante tener en cuenta que la palabra clave final se puede aplicar tanto a clases como a métodos y propiedades, pero no se puede aplicar a funciones o constantes a nivel global. Además, es importante considerar cuidadosamente el uso de final, ya que puede limitar la flexibilidad y capacidad de extensión de tu código.

**MATRICES (ARRAYS) EN PHP.**

Una matriz se puede definir del siguiente modo:

```
<?php
$paper[] = "Copiadora";
$paper[] = "Inyección de tinta";
$paper[] = "Láser";
$paper[] = "Fotográfico";

print_r($paper);
?>
```

El resultado del código anterior es:

```
Array
(
    [0] => Copiadora
    [1] => Inyección de tinta
    [2] => Láser
    [3] => Fotográfico
)
```

También podemos definir la matriz como sigue:

```
<?php
$paper[3] = "Copiadora";
$paper[2] = "Inyección de tinta";
$paper[1] = "Láser";
$paper[0] = "Fotográfico";

print_r($paper);
?>
```

Con el mismo resultado, pero invertido:

```
Array
(
    [0] => Fotográfico
    [1] => Láser
    [2] => Inyección de tinta
    [3] => Copiadora
)
```

Para un sitio web en producción no se suele utilizar la función print_r, que es más para labores de depuración. El siguiente código muestra como imprimir los elementos de la matriz:

```
<?php
$paper[] = "Copiadora";
$paper[] = "Inyección de tinta";
$paper[] = "Láser";
$paper[] = "Fotográfico";

for ($j = 0; $j < 4; ++$j)
    echo "$j: $paper[$j] \n";
?>
```

Con ese código se obtiene como resultado el siguiente:

```
0: Copiadora
1: Inyección de tinta
2: Láser
3: Fotográfico
```

**Matrices asociativas (clave-valor).**

Hacer un seguimiento de los elementos de una matriz por índices funciona bien, pero con las matrices asociativas se puede hacer referencia a los elementos de la matriz por nombres en lugar de hacerlo por números. La siguiente es uan matriz asociativa:

```php
<?php
$paper['copiadora'] = "Copiadora";
$paper['inyeccion'] = "Inyección de tinta";
$paper['laser'] = "Láser";
$paper['foto'] = "Fotográfico";
echo $paper['laser'];
?>
```

Cada elemento yiene ahora un nombre único que s epuede utilizar para referecniarlo en otro lugar, como en la declaracióne cho anterior.

Cuando se utilizan matrices asociativas, en lugar de matrices numéricas, el código para referirse a sus eleemntos es fácil de escribir y de depurar.

**Asigación mediante la palabra clave array.**

```php
<?php
$p1 = array("Copiadora", "Inyección", "Láser", "Fotográfica");
echo "Elemento de \$p1: " . $p1[2] . "\n";

$p2 = array('copiadora' => "Copiadora y multipropósito",
            'inyeccion' => "Impresora de inyección",
            'laser' => "Impresora láser",
            'foto' => "Papel fotográfico");
echo "Elemento de \$p2: " . $p2['inyeccion'] . "\n";
?>
```

```
Elemento de $p1: Láser
Elemento de $p2: Impresora de inyección
```

$p1 y $p2 son diferentes tipos de matrices, la primera es de indexación numeŕica, la segunda una matriz asociativa de clave-valor.

**Bucle 'foreach...as'.**

Uso de foreach...as con una matriz de indexación numérica:

```php
<?php
$paper = array("Copiadora", "Inyección", "Láser", "Fotográfica");
$j = 0;

foreach($paper as $item){
    echo "$j: $item\n";
    ++$j;
}
?>
```

Cuando PHP encuentra la instrucción foreach, toma el primer elemanto de la matriz y lo coloca en la variable que sigue a la palabra clave as, en este caso $item; cada vez que el flujo de control vuelve a foreach, el siguiente elemento de la martriz se coloca en la variable que sigue a la palabra as. Una vez que se han utilizado todos los valores, termina la ejecuciónd el bucle. La salida del código anterior es:

```
0: Copiadora
1: Inyección
2: Láser
3: Fotográfica
```

Uso de foreach...as con una matriz asociativa:

```php
<?php
$paper = array('copiadora' => "Copiadora y multipropósito",
               'inyeccion' => "Impresora de inyección",
               'laser' => "Impresora láser",
               'foto' => "Papel fotográfico");
foreach($paper as $item => $description)
    echo "$item: $description\n";
?>
```

Salida:

```
copiadora: Copiadora y multipropósito
inyeccion: Impresora de inyección
laser: Impresora láser
foto: Papel fotográfico
```

**Recorrido de matrices de varias dimensiones.**

```php
<?php
$products = array(
    'paper' => array(
        'copier' => "Copiadora",
        'inkjet' => "Impresora de inyección",
        'laser' => "Impresora láser",
        'photo' => "Papel fotográfico"),

    'pens' => array(
        'ball'=> "Bolígrafos",
        'hilite' => "Florescentes",
        'marker' => "Marcadores"),
    'misc' => array(
        'tape'=> "Cinta adhesiva",
        'glue'=> "Adhesivos",
        'clips' => "Clips")
);

foreach($products as $section => $items)
    foreach($items as $key => $value)
        echo "$section:\t$key\t($value)\n";

echo "Acceso a un elemento individual:\n";
echo $products['misc']['glue'];
?>
```

La salida de este programa es la siguiente:

```
paper:      copier      (Copiadora)
paper:      inkjet      (Impresora de inyección)
paper:      laser       (Impresora láser)
paper:      photo       (Papel fotográfico)
pens:       ball        (Bolígrafos)
pens:       hilite      (Fluorescentes)
pens:       marker      (Marcadores)
misc:       tape        (Cinta adhesiva)
misc:       glue        (Adhesivos)
misc:       clips       (Clips)
Acceso a un elemento individual:
Adhesivos
```

**PHP EN LA PRÁCTICA.**

**Uso de printf.**

Con el uso de printf, en lugar de con echo y print, podemos controlar el formato de la salida al permitir añadir caracteres especiales de formato en una cadena. Para cada carácter de formato, printf espera que se pase un argumento, y se producirá la presentación usando ese formato. Es similar al comando printf del lenguaje C. Por ejemplo, el siguiente ejemplo utiliza el especificador de conversión %d para mostrar el valor 3 (arg, -argumento-) en decimal:

```
<?php
printf("Hay %d productos en tu cesta.", 3);
?>
```

Los especificadores de conversión permitidos son los siguientes:

| | |
|---|---|
| % | Muestra el carácter % (no se requiere argumento). |
| b | Muestra arg como un entero binario. |
| c | Muestra el carácter ASCII para arg. |
| d | Muestra arg como entero decimal. |
| e | Muestra arg en notación científica. |
| f | Muestra arg en punto flotante. |
| o | Muestra arg como entero octal. |
| s | Muestra arg como una cadena. |
| u | Muestra arg como un decimal sin signo. |
| x | Muestra arg en hexadecimal, en minúsculas. |
| X | Muestra arg en hexadecimal, en mayúsculas. |

Siempre que se pasen el correspondiente núemro de argumentos, se pueden incluir tantos especificadores como se desee.

Un ejemplo práctico del uso de printf define los colores en HTML, usando valores decimales para su correspondiente codificación en hexadecial, por ejemplo:

```
<?php
printf("<spam style='color:#%X%X%X'>Hola</spam>", 65, 127, 245);
?>
```

La salida de este ejemplo es la siguiente:

```
<spam style='color:#417FF5'>Hola</spam>
```

Como en el lenguaje C, podemos indicar en el especificador la precisión decimal del resultado mostardo, por ejemplo, para indicar una precisiónd e dos decimales , indicamos .2 entre el signo % y el especificador de converisón.

```
<?php
printf("El resultado es: %.2f", 123.42 / 12);//Imprime: El resultado es: 10.29
?>
```

También se puede indicar si se desea rellenar la salida con ceros o espacios anteponiendo al especificador determinados valores, por ejemplo:

```
<?php
printf("El resultado es: %15.2f", 123.42 / 12);//Campo de longitud 15.
?>

<?php
printf("El resultado es: %015f", 123.42 / 12);//El resultado es: 00000010.285000
?>
```

También podemos rellenar cadenas y/o establecer una longitud deseada como podemos hacer con los números. Por ejemplo:

```php
<?php
printf("%'#10s", Hola); // Imprime: ######Hola
?>
```

**Uso de sprintf.**

Puede ocurrir que no requiramos el resultado de uan conversión en la salida estándar, o en el navegador, pero necesitemos usar el resulatdo de la conversión en otra parte del código. La función sprintf, devuelve una cadena formateada como resultado en lugar de imprimir directamente los datos. Esto significa que puedes guardar el resultado formateado en una variable para usarlo más tarde o manipularlo según sea necesario. Ejemplos:

```php
<?php
$hexstring = sprintf("%X%X%X", 65, 127, 245); // $hexstring = 417FF56
echo $hexstring;
?>
```

```php
<?php
$out = sprintf("El resultado es: %.2f", 123.42 / 12);
echo $out; //Imprime: El resultado es: 10.29
?>
```

**Funciones de fecha y hora.**

PHP usa marcas de tiempo estándar de Unix, el núemro de segundos desde el comienzo del 1 de enero de 1970.

```php
<?php
echo time(); // Imprime: 1685350298
?>
```

Debido a que el valor se almacena en segundos, para obteenr, por ejemplo, la indicación de fecha y hora para este instante de la próxima semana, se utilizaría la seiguiente sentencia:

```
echo time() + (7*24*60*60);
```

Si se desea crear un aindicación de fecha y hora para una fecha determinada se usa la función mktime(). Por ejemplo, para indicar fecha y hora en formato Unix de las 00:00 del 1 de enero del año 2030:

```php
<?php
echo mktime(0, 0, 0, 1, 1, 2030); // Imprime: 1893452400
?>
```

**Función date().**

Date() se usa para visualizar la fecha, su formato es el siguiente:

```
date($format, $timestamp);
```

El parámetro format debe ser una cadena que contenga especificadores de formato, a continuación un ejemplo; y $timestamp debe ser una indicación Unix de fecha y hora, por ejemplo, para imprimir la hora y fecha actual:

```php
<?php
echo date("l F jS, Y - g:ia", time());//Imprime: Monday May 29th, 2023 - 11:06am
?>
```

Concultar https://www.php.net/manual/es/function.date.php para ver una lista completa de los especificadores de formato que pueden ser usados.

**Constantes de fecha.**

Hay un número de constantes útiles para usar con la función date(), para devolver la fecha en formatos específicos. Por ejemplo, date(DATE_RSS) devuelve la fecha y hora en formato válido para un feed RSS. Algunas de las constantes más utilizadas son las siguientes:

```php
<?php
echo date(DATE_ATOM, time());    // 2023-05-29T11:27:05+02:00Monday
echo date(DATE_COOKIE, time()); // 29-May-2023 11:27:05 CESTMon
echo date(DATE_RSS, time());     // 29 May 2023 11:27:05
echo date(DATE_W3C, time());     // +02002023-05-29T11:27:05+02:00
?>
```

**Validación de fecha chackdate().**

checkdate() davuelve TRUE si la fecha es válida o FALSE si no lo es:

```php
<?php
$month = 9;
$day = 31;
$year = 2023; // Una fecha inválida.

if(checkdate($month, $day, $year)) echo "La fecha es válida";
else echo "La fecha es inválida";
?>
```

**Manejo de archivos.**

**Verificación de la existencia de un archivo.**

```php
<?php
if(file_exists("testfile.txt")) echo "El archivo existe.";
else echo "Archivo no encontardo.";
?>
```

**Creación de archivos.**

```php
<?php
$fh = fopen("testfile.txt", 'w') or die("Error al crear el archivo");

$text = <<<TEST
Línea 1
Línea 2
Línea 3
TEST;
fwrite($fh, $text) or die("Imposible escribir en el archivo");
fclose($fh);
echo "El archivo 'testfile.txt' fue creado y escrito."
?>
```

Existen varios parámetros de modo que se pueden utilizar, tal y como se ha usado 'w', son los siguientes:

'r'

Abre solo para lectura; colocal el puntero al principio del archivo. Devuelve FALSE si el archivo no existe.

'r+'

Abre para lectura y escritura; colocal el puntero al principio del archivo. Devuelve FALSE si el archivo no existe.

'w'

Abre solo para escritura; coloca el puntero al principio del archivo y trunca la longuitud del archivo a cero. Si el archivo no existe intenta crearlo.

'w+'

Abre para lectura y escritura; colocal el puntero al principio del archivo y trunca la longuitud del archivo a cero. Si el archivo no existe intenta crearlo.

'a'

Abre solo para escritura; coloca el puntero al final del archivo. Si el archivo no existe intenta crearlo.

'a+'

Abre para lectura y escritura; coloca el puntero al final del archivo. Si el archivo no existe intenta crearlo.

**Lectura de archivos.**

Leer parte de un archivo.

```php
<?php
$fh = fopen("testfile.txt", 'r') or die("El archivo no existe");

$line = fgets($fh);
$chars = fread($fh, 6);
fclose($fh);
echo $line; // Imprime: Línea 1
echo $chars; // Imprime; Línea (6 caracteres)
?>
```

fgest() devuelve una línea del archivo y fread() lee el grupo de caracteres indicado en su segundo argumento. La función fread se utiliza normalmente con datos binarios

Leer un archivo completo: get_file_contents().

```php
<?php
$filename = 'testfile.txt';
$fh = fopen($filename, 'r') or die("El archivo no existe");

$filesize = filesize($filename);
$file = fread($fh, $filesize);
```

```
fclose($fh);

echo $file; // Imprime el contenido completo del archivo testfile.txt
echo file_get_contents($filename); // Lee también el archivo completo.
?>
```

file_get_contents() no necesita abrir y cerra el archivo, puede leer su contenido directamente. Otra forma de usar file_get_contents() es para recuperar archivos o información de Internet, como en el siguiente ejemplo:

```
<?php
$url = 'https://www.s2a.es';
$page = file_get_contents($url);

echo $page;
?>
```

**Copia, renombrado y eliminación de archivos.**

```
<?php
if(!copy('testfile.txt', 'testfile2.txt'))
    echo "No se ha podido copiar el archivo";
else echo "Archivo copiado";
?>

<?php
if(!rename('testfile2.txt', 'testfile2.new'))
    echo "No se ha podido renombrar el archivo";
else echo "Archivo renombrado";
?>

<?php
if(!unlink('testfile2.new'))
    echo "No se ha podido borrar el archivo";
else echo "Archivo borrado";
?>
```

En los ejemplos se utiliza el operador ! (NOT) para evitar un intento fallido. Colocado delante de una expresión aplica el operador NOT a la misma (si no puedes copiar, renombrar o borrar, ...).

**Carga de archivos a un servidor.**

```
<?php
echo <<<_END
    <html><head><title>PHP Form Upload</title></head><body>
    <form method='post' action='upload.php' enctype='multipart/form-data'>
    Select File: <input type='file' name='filename' size='10'>
    <input type='submit' value='Upload'>
    </form>
_END;

if ($_FILES) {
    $name = $_FILES['filename']['name'];
    move_uploaded_file($_FILES['filename']['tmp_name'], $name);
    echo "Uploaded image '$name'<br><img src='$name'>";
}
echo "</body></html>";
?>
```

Examinemos el programa anterior. La primera sección es código HTML para presentar el formulario de selección y subida de archivos, que selecciona el método POST de envío de formularios, establece el destino de los datos enviados al programa e indica al nevegador web qu elos datos enviados se deben codificar a través del tipo de contenido de "multipart/form-data".

"multipart/form-data" es un tipo de codificación de datos utilizado en formularios HTML para enviar archivos y datos binarios a través de una solicitud HTTP. Se utiliza cuando se necesita enviar archivos u otros datos binarios, como imágenes, archivos de audio, videos, etc.

La codificación "multipart/form-data" divide los datos en varias partes y las envía en una estructura de contenido multipart. Cada parte contiene una sección de datos con su propio encabezado y contenido. Cada parte puede contener tanto datos de texto como datos binarios, y se identifica mediante un identificador único (boundary) que se especifica en el encabezado de la solicitud.

En PHP, los archivos enviados a través de "multipart/form-data" se pueden acceder a través de la variable global $_FILES. Todos los archivos cargados se colocan en la matriz asociativa del sistema $_FILES. Por lo tanto, es suficiente una rápida comprobación para ver si $_FILES contiene algo y así determinar si el usuario ha subido un archivo. Esto se hace con la sentencia

```
if($_FILES)
```

La primera vez que el usuario visita la página, antes de subir el archivo, $_FILES está vacía, por lo que el programa se salta este bloque de código. Cuando el usuario sube un archivo, el programa se ejecuta otra vez

```
action='upload.php'
```

entonces descubre un elemento en la matriz $_FILES. Una vez que el programa se da cuenta que se ha cargado un archivo, el nombre del mismo se recupera y se coloca en la variable $name

```
$name = $_FILES['filename']['name'];
```

Ahora todo lo que se necesita hacer es mover el archivo desde la ubicación temporal en la que lo colocó PHP a una ubicación permanente. Hacemos esto con la función move_uploaded_file(), le pasamos el nombre original del archivo y, con ese nombre, se guarda en el directorio en uso

```
move_uploaded_file($_FILES['filename']['tmp_name'], $name);
```

**Uso de $_FILES.**

Cuando se carga un archivo se almacenan cinco valores en la matriz $_FILES, como se muestra a continuación (donde file es el nombre del campo de carga del archivo suministrado por el formulario de envío):

$_FILES[[‘file][‘name’] Nombre del archivo cargado.
$_FILES[[‘file][‘type’] Tipo de contenido del archivo (p.ej., image/jpeg).
$_FILES[[‘file][‘size’] Tamaño del archivo en bytes.
$_FILES[[‘file][‘tmp_name’] Nombre del archivo temporal en el servidor.
$_FILES[[‘file][‘error’] Código de error resultante de la carga.

A los tipos de contenido se les solía conocer como tipos MIME (Multipurpose Internet Mail Extension), pero debido a que más tarde se extendió su uso a todo Internet, ahora se los llama a menudo Internet Media Types.

**Llamadas al sistema.**

A veces PHP no tendrá alguna función que se necesite para realizar una acción determianda, pero el sistema operativo que esta ejecutando PHP puede lleavrla a cabo. En tales casos, su puede utilizar la función exec() para hacer el trabajo.

```php
<?php
//$cmd = "dir"; // Windows
$cmd = "ls -lha"; // Linux
exec(escapeshellcmd($cmd), $output, $status);
if ($status) echo "Exec command failed";
else {
    echo "<pre>\n";
    foreach($output as $line) echo htmlspecialchars("$line\n");
    echo "</pre>";
}
?>
```

Salida (contenido del directorio en el que se encuentra el programa):

```
total 260K
drwxrwxrwx  2 n7rc     n7rc     4.0K May 29 19:10 .
drwxrwxrwx 11 root     root     4.0K May 29 06:17 ..
-rw-r--r--  1 www-data www-data 237K May 29 18:20 apuntes_PHP_v1.0.pdf
-rw-r--r--  1 n7rc     n7rc      280 May 29 19:13 exec.php
-rwxrwxrwx  1 n7rc     n7rc       74 May 29 06:17 index.php
-rwxrwxrwx  1 n7rc     n7rc      480 May 29 17:54 upload.php
```

En el programa se llama a la función htmlspecialchars para convertir cualquier caracter especial devuelto por el sistema en caracteres que HTML pueda entender y mostrar correctamente.

exec() acepta tres argumentos:

- El propio comando, ($cmd)
- Una matriz en la que el sistema colocará la salida del comando, ($output).
- Una variable para contener el estado devuelto de la llamada, ($status).

La función escapeshellcmd() desinfecta la cadena de comandos e impide la ejecución de comandos arbitrarios. Es un buen hábito utilizarla siempre que se emite una llamada a exec().

**¿Qué diferencia existe en PHP entre usar exec('ls') y las comillas simples `ls` para ejecutar un comando del sistema?**

Función exec():

La función exec() se utiliza para ejecutar un comando en el sistema y devuelve la última línea de la salida generada por el comando. El resultado se puede asignar a una variable para su posterior procesamiento.

```
$var = exec('ls -lha');
```

Comillas simples `ls` (comillas invertidas o backticks):

Las comillas simples `ls` (comillas invertidas o backticks) se utilizan como una forma abreviada para ejecutar comandos en el sistema operativo y capturar la salida completa generada por el comando.

```
$var = `ls -lha`;
```

En este caso, `ls -lha` ejecutaría el comando "ls -lha" en el sistema operativo y capturaría toda la salida generada por el comando. El resultado se puede asignar a una variable como en el ejemplo.

La principal diferencia entre exec('ls') y `ls` radica en cómo se maneja la salida del comando:

exec('ls') devuelve solo la última línea de salida generada por el comando, lo cual es útil si solo necesitas esa información específica.

`ls` captura y devuelve la salida completa generada por el comando, lo que te permite acceder a toda la salida generada por el comando en forma de una cadena, que luego puedes procesar o analizar según sea necesario.

```php
<?php
echo `ls -lha`;
?>
```

Salida:

> total 260K drwxrwxrwx 2 n7rc n7rc 4.0K May 29 19:10 . drwxrwxrwx 11 root root 4.0K May 29 06:17 .. -rw-r--r-- 1 www-data www-data 237K May 29 18:20 apuntes_PHP_v1.0.pdf -rw-r--r-- 1 n7rc n7rc 285 May 29 19:23 exec.php -rwxrwxrwx 1 n7rc n7rc 74 May 29 06:17 index.php -rwxrwxrwx 1 n7rc n7rc 480 May 29 17:54 upload.php

Es importante tener en cuenta que al utilizar la ejecución de comandos del sistema en PHP, debes tener precaución y asegurarte de que los comandos ejecutados sean seguros y confiables. Además, ten en cuenta que el uso de la ejecución de comandos del sistema puede depender de la configuración del servidor y puede estar restringido por motivos de seguridad. Las funciones de llamadas al sistema suelen estar deshabilitadas en los servidores web compartidos, suponen un riesgo para la seguridad. Siempre se debe intentar resolver los problemas dentro de PHP, si se puede, e ir al sistema solo si es estrictamente necesario.

**EJEMPLO DE UN PROGRAMA EN PHP.**

Mostraremos un ejemplo de PHP, un carrusel de imágenes autónomo. El código es autoexplicativo. Se incorporan los archivos CSS y el script Javascript para hacer el programa funcional. El programa requiere de las bibliotecas jquery y bootstrap. El código está escrito por procedimientos y no contempla POO.

El programa muestra información y noticias mediante un carrusel de imágenes. Para que una imagen entre en cola de reproducción el nombre de la misma debe comenzar por la fecha de vencimiento de la noticia (yyyymmdd*.*). Una vez vencida la fecha saldrá de cola de reproducción y la imagen será eliminada. Optimizado para mostrar en monitor/TV sin intervención del usuario.

[config.inc.php]

```php
<?php
/* Configuración */

// Título de la página.
$title = 'Carrusel Imágenes';

// Datos de la barra de cabecera (activar/desactivar).
$header = True; // True/False activa/desactiva la barra.
$titleHeader = 'Título cabecera';
$subTitleHeader = 'Subtítulo cabecera';
$chanelName = 'Nombre de canal';
$subTitleChanel = 'Subtítulo canal';

$pool = 'images'; // carpeta origen de las imágenes del slider.

$widthImage = 1600; // dimensión imagen en píxeles.
$heightImage = 900; // dimensión imagen en píxeles.
$fadeIn = 5000; // tiempo de entrada de la imagen (mseg).
$fadeOut = 5000; // tiempo de salida de la imagen (mseg).
$timeFocus = 60; // tiempo de permanencia en foco de la imagen (seg).
?>
```

[functions.inc.php]

```php
<?php

/*getPool() obtiene las imágenes y ubicación de la carpeta de reproducción de un Slider.*/
function getPool($path){

  if(substr($path, -1) != "/") $path .= "/";// añade "/" al final de la carpeta.
  $dir = opendir($path);
  $files = array();

  while ($current = readdir($dir)){
    if( $current != "." && $current != "..") {
      if(is_dir($path.$current)) {
        getPool($path.$current);
      } else {
        $files[] = $current;
      }
    }
  }
  return array($path, $files);
}
```

```
/*getCarousel() obtiene la cola de reproducción incorporando las imagenes al
Slider de forma aleatoria. Si la fecha en el nombre del archivo es menor a la
fecha actual saca la imagen de cola de reproducción. Si el nombre del archivo no
comienza por una fecha válida no entra en cola de reproducción.*/
function getCarousel($path){
  global $path;
  global $files;
  global $widthImage;
  global $heightImage;

  shuffle($files); // desordena el array de forma aleatoria.

  for($i=0; $i<count( $files ); $i++){
      if (is_numeric(substr($files[$i],0,8)) & substr($files[$i],0,8) >=
          Date("Ymd")){
      echo '<div style="width: ' . $widthImage . 'px; height: ' . $heightImage .
           'px;
            max-height: ' . $heightImage . 'px;">
            <img style="width: ' . $widthImage . 'px; height: ' . $heightImage .
           'px
            " src="' . $path . $files[$i] . '" />
            </div>';
    }
  }
}

/*getIndicators() obtiene el número de imágenes en la cola de reproducción
generando un elemento <li> para cada uno de ellos dentro de la clase css
"indicartors".*/
function getIndicators($path){
  global $files;
  //Se imprime un solo elemento de la clase "active".
  echo '<li class="active"><em></em></li>';

  //Se imprime la lista de elementos (<li> - 1).
  for($i=0; $i<(count( $files ) - 1); $i++){
      if (is_numeric(substr($files[$i],0,8)) & substr($files[$i],0,8) >=
          Date("Ymd")){
      echo '<li><em></em></li>';
    }
  }
}

/*printHeader() imprime la barra de cabecera si $header está definido a True.*/
function printHeader($header){
  global $header;
  global $titleHeader;
  global $subTitleHeader;
  global $chanelName;
  global $subTitleChanel;

  if ($header === True){
      echo '<nav style="margin: 0px; height: 3%; width: 100%;" class="navbar
navbar-default">
            <div style="margin: 0px; height: 3%;"class="container-fluid">
              <div class="navbar-header">
                <a class="navbar-brand">' . $titleHeader . '</a>
                <a class="navbar-brand">:::: ' . $subTitleHeader . ' ::::</a>
              </div>
              <div class="collapse navbar-collapse" id="bs-example-navbar-
               collapse-1">
```

```
                <ul class="nav navbar-nav">
                <li class="active"><a>Canal: #' . $chanelName . '</a></li>
                <li class="active"><a>:::: ' . $subTitleChanel . ' ::::</a></li>
                </ul>
              </div>
            </div>
          </nav>';
  }else{
    return;
  }
}

/*refreshTime() devuelva el tiempo de refresco en el navegador. El tiempo de
refresco es múltiplo de las imágenes en cola de reproducción.*/
function refreshTime($path){
  global $files;
  global $timeFocus;
  $imagesIn = array();

  for($i=0; $i<count( $files ); $i++){
    if (substr($files[$i],0,8) >= Date("Ymd")){
      array_push($imagesIn, $files[$i]);
    }
  }
  $refreshTime = count($imagesIn) * $timeFocus;
  return ($refreshTime);
}

/*delFiles() elimina los ficheros que inician su nombre con fecha menor a la
fecha del sistema.*/
function delFiles($path){
  global $path;
  global $files;

  for($i=0; $i<count( $files ); $i++){
      if (is_numeric(substr($files[$i],0,8)) & substr($files[$i],0,8) <
          Date("Ymd")){
      unlink ($path . $files[$i]);
    }
  }
  return;
}
?>
```

[index.php]

```
<?php

/*
Muestra información y noticias mediante un carrusel de imágenes. Para que una
imágen entre en cola de reproducción el nombre de la misma debe comenzar por la
fecha de vencimiento de la noticia (yyyymmdd*.*). Una vez vencida la fecha
saldrá de cola de reproducción y la imagen será eliminada.
*/

require_once ('config.inc.php');
require_once ('functions.inc.php');

$path  = getPool($pool)[0]; // Ubicación de las imágenes.
$files = getPool($pool)[1]; // Elementos del carrusel.
?>
```

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="utf-8">
  <title><?php echo ($title); ?></title>
  <meta http-equiv="refresh" content="<?php echo (refreshTime($pool)); ?>">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="css/bootstrap.min.css" rel="stylesheet">
  <link href="css/style.css" rel="StyleSheet" type="text/css">
  <script src="js/jquery-3.5.0.min.js"></script>
  <script src="js/bootstrap.min.js"></script>
  <script type="text/javascript">
    //Carrusel/Indicadores
    $(function(){
      $('#carousel div:gt(0)').hide();
      setInterval(function(){
            // control carrusel
            $('#carousel div:first-child').fadeOut(<?php echo ($fadeOut); ?>)
            .next('div').fadeIn(<?php echo ($fadeIn); ?>)
            .end().appendTo('#carousel');
            // control indicadores
            $('.indicators li.active').removeClass('active').next()
            .add('.indicators li:first').last().addClass('active');
      }, <?php echo ($timeFocus * 1000); ?>); // milisegundos.
    });
  </script>
</head>
<body>

<?php printHeader($header) ?>

<div style="margin-top: 3%; margin-bottom: 1%;">
  <div align="center" class="col-md-12 col-sm-12  col-xs-12" style="width: 100%;
       height: 100%;">
    <div id="carousel">
      <?php getCarousel($pool); ?>
    </div>
  </div>
</div>

<div  align="center" class="col-md-12 col-sm-12   col-xs-12" style="width: 90%;
      height: 100%;">
  <ul class="indicators">
    <?php getIndicators($pool); ?>
  </ul>
</div>

</body>
</html>

<?php delFiles($pool);?>

[/css/style.css]

body {
  background-color: #cbccca;
  max-width: 100%;
}
```

```
#carrousel {
  position: relative;
  overflow: hidden;
  width:1600px;
  max-height:900px;
  border: 0px;
  border-radius: 50px;
  margin: 0 auto;
  padding: 0;
}

/*indicators*/

*,
*:after,
*:before {
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}

.indicators {
  display: block;
  height: 1.2em;
  list-style-type: none;
  margin: 2% 3% auto 2%;
  width: 100%;
}

.indicators > li {
  -moz-border-radius: 0.05em / 0.05em;
  -webkit-border-radius: 0.05em 0.05em;
  border-radius: 0.05em / 0.05em;
  -moz-transition: background-color 0.15s linear;
  -o-transition: background-color 0.15s linear;
  -webkit-transition: background-color 0.15s linear;
  transition: background-color 0.15s linear;
  cursor: pointer;
  float: left;
  height: 100%;
  margin-left: 0.3em;
  position: relative;
  vertical-align: top;
  width: 1.8em;
}

.indicators > li:first-child {
  margin-left: 0;
}

.indicators > li > em {
  -moz-transition: all 0.4s linear;
  -o-transition: all 0.4s linear;
  -webkit-transition: all 0.4s linear;
  transition: all 0.4s linear;
  background-color: #a3a3a3;
  content: "";
  display: inline-block;
  height: 40%;
  position: absolute;
  right: 0;
```

```
  bottom: 0;
  left: 0;
}

.indicators li:hover > em {
  height: 100%;
}

.indicators > .active > em {
  background-color: #4d4d4d;
  height: 100%;
}
```

**INTRODUCCIÓN A MySQL.**

Una base de datos MySQL contiene una o más talas, cada una de las cuales contiene registros o filas. Dentro de estas filas hay varias columnas o campos que contienen los datos propiamente dichos.

Los principales términos inciales para familiarazarse con una base de datos son los siguientes:

Database (Base de datos)

Contenedor general de una colección de datos.

Table (Tabla)

Subcontenedor dentro de una base de datos que almacena los datso reales.

Row (Fila)

Regístro único dentro de una tabla, que puede contenar varios campos.

Column (Columna)

Nombre de una campo dentrod e una fila.

**Conexión a una base de datos MySQL.**

```
root@s2a:~# mysql -u root -p
Enter password:

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 7
Server version: 10.5.19-MariaDB-0+deb11u2 Debian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| moodle             |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0,004 sec)
```

**Comandos MySQL.**

| | |
|---|---|
| ALTER | Modificar una base de datos o una tabla. |
| BACKUP | Copia de seguridad de una tabla. |
| \c | Cancelar la entrada. |
| CREATE | Crear una base de datos. |
| DELETE | Eliminar una fila de una tabla. |
| DESCRIBE | Describir la columnas de una tabla. |
| DROP | Eliminar una base de datos o una tabla. |
| EXIT (Ctrl-c) | Salir |

| | |
|---|---|
| GRANT | Cambiar los privilegiso de un usuario. |
| HELP (\h, \?) | Mostrar ayuda. |
| INSERT | Insertar datos. |
| LOCK | Bloquear una(s) tabla(s). |
| QUIT (\q) | Salir. |
| RENAME | Cambiar el nombre de una tabla. |
| REVOKE | Eliminar privilegios. |
| SHOW | Listar los detalles de un objeto. |
| SOURCE | Ejecutar un archivo. |
| STATUS (\s) | Visualizar el estado en curso. |
| TRUNCATE | Vaciar una tabla. |
| UNLOCK | Desbloquear una(s) tabla(s). |
| UPDATE | Actualizar un registro existente. |
| USE | Uso de una base de datos. |

Los comandos y las palabras clave no distinguen entre mayúsculas y minúsculas, el estilo recomendado es usar mayśuculas. Los nombres de tablas distinguen entre mayúsculas y minúsculas en Linux, pero no en Windows, el estilo recomendado es usar minúsculas.

**Trabajo con bases de datos en MySQL.**

```
MariaDB [(none)]> CREATE DATABASE publications;
Query OK, 1 row affected (0,000 sec)

MariaDB [(none)]> USE publications;
Database changed
MariaDB [publications]>

MariaDB [publications]> GRANT ALL ON publications.* TO 'user'@'localhost'
IDENTIFIED BY 'qwerty';
Query OK, 0 rows affected (0,000 sec)
```

Con este último comando hemos permitido al usuario 'user' desde localhost el acceso completo a la base de datos publications usando la contraseña 'qwerty'; le hemos dado todos los privilegios.

Ahora pasamos a crear una tabla en la base de datos publications que seleccionamos anteriormente, con el comando `USE publications;`.

```
MariaDB [publications]> CREATE TABLE classics (
    -> author VARCHAR(128),
    -> title VARCHAR(128),
    -> type VARCHAR(16),
    -> year CHAR(4));
Query OK, 0 rows affected (0,020 sec)
```

Y verificamos que se ha creado correctamente:

```
MariaDB [publications]> DESCRIBE classics;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| author | varchar(128) | YES  |     | NULL    |       |
| title  | varchar(128) | YES  |     | NULL    |       |
| type   | varchar(16)  | YES  |     | NULL    |       |
| year   | char(4)      | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0,004 sec)
```

Analicemos cada uno de los encabezados de la salida anteiror:

| | |
|---|---|
| `Field` | Nombre de cada campo o columna dentro de una tabla. |
| `Type` | Tipo de datos que se almacena en cada campo. |
| `Null` | Si se permite que el campo contenga un valor NULL. |
| `Key` | Qué tipo de clave, si la hubiera, se ha aplicado. [*] |
| `Default` | Cuando se crea una nueva fila, si no se ha especificado ningún valor a cada campo, se asiganrá uno por defecto. |
| `Extra` | Información adicional, como por ejemplo, si un campo está configurado como autoincremento. |

[*] Las claves o índices en MySQL son una manera rápida de averiguar y buscar datos. En MySQL, una clave (key) se refiere a un campo o conjunto de campos que se utilizan para identificar y organizar los datos en una tabla. Las claves son fundamentales para el diseño y rendimiento de una base de datos, ya que permiten la búsqueda rápida y eficiente de registros.

En MySQL, existen varios tipos de claves que se pueden utilizar:

- Clave primaria (Primary Key): Es un tipo de clave única que identifica de forma única cada registro en una tabla. Una tabla puede tener solo una clave primaria y no puede tener valores duplicados o nulos. La clave primaria se utiliza comúnmente para establecer relaciones con otras tablas.

- Clave única (Unique Key): Es una clave que garantiza que no se puedan insertar valores duplicados en un campo o conjunto de campos específicos. A diferencia de la clave primaria, una tabla puede tener varias claves únicas.

- Clave externa (Foreign Key): Es un tipo de clave que establece una relación entre dos tablas. La clave externa en una tabla hace referencia a la clave primaria de otra tabla, lo que permite la vinculación y el mantenimiento de la integridad referencial entre las tablas.

- Clave índice (Index Key): Es una estructura de datos que mejora la velocidad de búsqueda en una tabla. Los índices se crean en uno o varios campos y permiten una búsqueda más rápida y eficiente de los registros en función de esos campos.
-

En resumen, las claves en MySQL son utilizadas para identificar y organizar los datos en una tabla, establecer relaciones entre tablas y mejorar la velocidad de búsqueda. Son una parte fundamental en el diseño y optimización de bases de datos.

**Tipos de datos.**

Tipo de datos CHAR.

El tipo `VARCHAR` es una cadena de caracteres de longitud variable en el que se especifica una longitud máxima, en el caso de nuestro ejemplo 128. Las cadenas de mayor longitud son truncadas al tamaño máximo expecificado.

El tipo CHAR especifica un tamaño para la cadena, si se coloca una cadena más pequeña se rellena con espacios; en nuestro ejemplo, al tratarse de una cadena para representar un año se ha indicado la longitud de 4.

Tipo de datos BINARY.

Almacenan cadenas de bytes que no tienen un juego de caracteres asociado. Por ejemplo se puede usar el tipo BINARY para almacenar una imagen GIF.

El tipo BINARY es como CHAR, pero contiene datos binarios.

El tipo VARBINARY es como VARCHAR, pero contiene datos binarios.

Tipo de datos TEXT.

Los datos correspondientes a caracteres también pueden almacenarse en uno de los campos TEXT. Las diferencias entre estos y los campos VARCHAR son mínimas; una de ellas es que los campos TEXT no pueden tener valores por defecto. Otra diferencia hace referencia a que MySQL indexa solo los primeros n caracteres de una columna TEXT. Esto significa que si se necesita buscar el contenido completo de un campo, VARCHAR es el tipo de datos mejor y más rápido que se puede usar. Sin embargo, si nunca se va a buscar más de un núemro determinado de caracteres principales en un campo, probablemente sea mejor opción utilizar TEXT. Existen varios tipos TEXT: TINYTEXT, TEXT, MEDIUMTEXT Y LONGTEXT.

Tipos de datos BLOB.

Los tipos BLOB (Binary Large Object), son como los BINARY pero para almacenar más de 65536 bytes. Los tipos BLOB no pueden contener valores por defecto. Existen los siguientes: TINYBLOB, BLOB, MEDIUMBLOB y LONGBLOB.

Tipos de datos INT.

Existen varios tipos numéricos: TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, BIGINT, FLOAT, DOUBLE/REAL. Se diferencian en el núemro de bytes usados y en los valores que pueden representar. Para especificar un tipo de datos sin signo, se utiliza el calificador UNSIGNED. Por ejemplo en la siguiente sentencia:

```
CREATE TABLE tablename (fieldname INT UNSIGNED);
```

Al crear un campo numérico, se puede pasar un número opcional como parámetro, pero este parámetro no indica el núemro de bytes de almacenamiento a utilizar, lo que realmente representa es la anchura del campo de visualización de los datos cuando se extraen. Se usa normalmente con el calificador ZEROFILL.

```
CREATE TABLE tablename (fieldname INT(4));
CREATE TABLE tablename (fieldname INT (4) ZEROFILL);
```

La última sentencia escrita hace que cualquier número con una anchura menor de cuatro caracteres se rellene con uno o más ceros, lo suficiente para que la anchura de visualización sea de cuatro caracteres.

Tipos de datos de FECHA y HORA.

```
DATETIME    Fecha y hora      '0000-00-00 00:00:00'
DATE        Fecha             '0000-00-00'
```

| | | |
|---|---|---|
| TIEMSTAMP | Fecha y hora | '0000-00-00 00:00:00' |
| TIME | Hora | '00:00:00' |
| YEAR | Año | '0000' |

Los tipos DATATIME y TIMESTAMP se muestran de la misma manera. La principal diferencia es que TIMESTAMP tiene un margen muy estrecho (1970-2037). TIMESTAMP es útil, sin embargo, porque puedes dejar que MySQL establezca el valor automáticamente "DEFAULT CURRENT_TIMESTAMP". Si no se especifica valor al añadir una fila, la hora en curso se insertará automáticamente. También es posible hacer que MySQL actualice una columna TIMESTAMP cada vez que cambies una fila. El tipo de dato TIMESTAMP es ampliamente utilizado en MySQL para almacenar y manipular valores de fecha y hora en aplicaciones y bases de datos. Proporciona funcionalidades para la gestión de fechas y horas, incluyendo comparaciones, cálculos y formatos de visualización.

**El atributo AUTO_INCREMENT.**

Una forma de que todas las filas de una tabla sean únicas es asegurarnos de ellos mediante un campo o columna AUTO_INCREMENT. Esto hace que MySQL establezca un valor único para esa slumna en cada fila. Realmente no tenemos control sobre el valor que tomará esta columna en cada fila, pero eso no importa si lo que se quiere es que se garantice un valor único.

```
MariaDB [publications]> ALTER TABLE classics ADD id INT UNSIGNED
    -> NOT NULL AUTO_INCREMENT KEY;
Query OK, 0 rows affected (0,017 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Se ha mostrado como añadir una nueva columna llamada id a la tabla classics con autoincremento. id es una columna de un entero sin signo, no nulo y es una clave primaria, esto es útil para buscar filas basadas en esa columna.

```
MariaDB [publications]> DESCRIBE classics;
+--------+------------------+------+-----+---------+----------------+
| Field  | Type             | Null | Key | Default | Extra          |
+--------+------------------+------+-----+---------+----------------+
| author | varchar(128)     | YES  |     | NULL    |                |
| title  | varchar(128)     | YES  |     | NULL    |                |
| type   | varchar(16)      | YES  |     | NULL    |                |
| year   | char(4)          | YES  |     | NULL    |                |
| id     | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
+--------+------------------+------+-----+---------+----------------+
5 rows in set (0,001 sec)
```

**Podemos borrar ahora la columna id del siguiente modo:**

```
MariaDB [publications]> ALTER TABLE classics DROP id;
Query OK, 0 rows affected (0,026 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Adicción de datos a una tabla.**

```
MariaDB [publications]> INSERT INTO classics (author, title, type, year)
    -> VALUES('Mark Twain', 'Las aventuras de Tom Sawyer', 'Ficción', '1876');
Query OK, 1 row affected (0,002 sec)
```

Hemos creado una entrada (fila) en la tabla classics. Veamos ahora el contenido de la tabla, mostrando todas sus filas:

```
MariaDB [publications]> SELECT * FROM classics;
+------------+----------------------------+----------+------+
| author     | title                      | type     | year |
+------------+----------------------------+----------+------+
| Mark Twain | Las aventuras de Tom Sawyer | Ficción  | 1876 |
+------------+----------------------------+----------+------+
1 row in set (0,000 sec)
```

**Cambio de nombre de una tabla.**

El cambio de nombre de una tabla como cualquier otro cambio en la estructura o en la metainformación de una tabla, se realiza mediante el comando ALTER. Así, por ejemplo:

```
MariaDB [publications]> ALTER TABLE classics RENAME books;
Query OK, 0 rows affected (0,004 sec)
```

**Modificación del tipo de datos de una columna.**

Modifiquemos la columna year de CHAR(4) a SMALLINT. También se usa el comando ALTER, en este caso con MODIFY.

```
MariaDB [publications]> ALTER TABLE classics MODIFY year SMALLINT;
Query OK, 1 row affected (0,024 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

**Adicción de una nueva columna.**

```
MariaDB [publications]> ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
Query OK, 0 rows affected (0,004 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Cambio de nombre de una columna.**

Cambiaremos el nombre de la columna type por el de category, lo haremos con ALTER TABLE (CHANGE) y se requiere que se especifique el tipo de datos, incluso si no hay intención de cambiarlo, y VARCHAR(16) era el tipo de datos especificado cuando esa columna se creó inicialmente como type.

```
MariaDB [publications]> ALTER TABLE classics CHANGE type category VARCHAR(16);
Query OK, 0 rows affected (0,004 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Eliminación de una columna.**

```
MariaDB [publications]> ALTER TABLE classics DROP pages;
Query OK, 0 rows affected (0,007 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Copia de seguridad de la base de datos.**

Para hacer una copia de seguridad usamos el comando mysqldump, es un comando par ser usado desde el sistema. Su uso es desde fuera de MySQL y se usa del siguiente modo:

```
mysqldump --user=user --password=qwerty publications > publications.sql
```

**Eliminación de una tabla.**

```
MariaDB [publications]> DROP TABLE classics;
Query OK, 0 rows affected (0,005 sec)

MariaDB [publications]> SHOW TABLES;
Empty set (0,000 sec)
```

**Restauración de copia de seguridad de la base de datos.**

Para restaurar una copia de seguridad usamos el comando mysql del modo:

```
mysql --user=user --password=qwerty publications < publications.sql
```

**Índices.**

Hasta ahora, del modo en el que están las cosas, la tabla classics funciona y permite búsquedas, pero al ir creciendo, las búsqueda cada vez serán más lentas. La manera de crear búsquedas rápidas es agregar un índice, ya sea al crear una tabla o en cualquier momento posterior. Existen diferentes tipos de índice: `INDEX` (índice normal), la `PRIMARY KEY`(Clave principal) o el índice `FULLTEXT`(Texto completo).

Es necesario decidir qué columnas requieren un índice y existen índices que se pueden combinar en varías columnas. Incluso se puede reducir el tamaño del índice al limitar el número de caracteres de cada columna que se debe indexar. Si imaginamos las búsquedas de la tabla classics, se hace evidente que se necesite buscar en todas las columnas (por ejemplo, si tuviese una columna pages en esta no sería necesario buscar). Creemos un índice para cada columna:

```
MariaDB [publications]> ALTER TABLE classics ADD INDEX(author(20));  ALTER TABLE
classics ADD INDEX(title(20));   ALTER TABLE classics ADD INDEX(category(4));
ALTER TABLE classics ADD INDEX(year);
Query OK, 0 rows affected (0,011 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0,008 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0,010 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0,009 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [publications]> DESCRIBE classics;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| author   | varchar(128) | YES  | MUL | NULL    |       |
| title    | varchar(128) | YES  | MUL | NULL    |       |
| category | varchar(16)  | YES  | MUL | NULL    |       |
| year     | smallint(6)  | YES  | MUL | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0,001 sec)
```

Se puede ver que ahora se muestra la clave `MULL` para cada columna. Esta clave significa que pueden suceder múltiples apariciones de un valor dentro de esa columna, que es exactamente lo que queremos, ya que los autores pueden aparecer muchas veces, el mismo título del libro lo podrían utilizar varios autores, etc.

Con el comando,

```
ALTER TABLE classics ADD INDEX(author(20));
```

se limita el índice a los primeros 20 caracteres, es decir cuando MySQL indexa el título:

```
Las aventuras de Tom Sawyer
```

solo almacenará el índice de los primeros 20 caracteres:

```
Las aventuras de Tom
```

Esto se hace para optimizar la velocidad de acceso. La longitud de 20 parece suficiente para asegurar la unicidad de la mayoría de las cadenas de las columnas author y title.

**Uso de CREATE INDEX.**

Los siguientes comandos son equivalentes. La diferencia entre ellos es que `CREATE INDEX` no su puede usar para crear una PRIMARY KEY:

```
ALTER TABLE classics ADD INDEX(author(20));
CREATE INDEX author ON classics (author(20));
```

**Adicción de índices durante la creación de tablas.**

```
MariaDB [publications]> CREATE TABLE books (
    -> author VARCHAR(128),
    -> title VARCHAR(128),
    -> category VARCHAR(16),
    -> year SMALLINT,
    -> INDEX(author(20)),
    -> INDEX(title(20)),
    -> INDEX(category(4)),
    -> INDEX(year));
Query OK, 0 rows affected (0,021 sec)

MariaDB [publications]> DESCRIBE books;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| author   | varchar(128) | YES  | MUL | NULL    |       |
| title    | varchar(128) | YES  | MUL | NULL    |       |
| category | varchar(16)  | YES  | MUL | NULL    |       |
| year     | smallint(6)  | YES  | MUL | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0,001 sec)
```

También es posible crear índices FULLTEXT, que buscan en columnas enteras de texto y permiten búsquedas muy rápidas, a diferencia de un índice normal. Los índices FULLTEXT se pueden crear solo para columnas CHAR, VARCHAR y TEXT.

```
MariaDB [publications]> ALTER TABLE classics ADD FULLTEXT (author, title);
Query OK, 0 rows affected (0,071 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Claves principales.**

Vamos a añadir un campo y los datos correspondientes para el isbn de los libros de la base de datos:

```
ALTER TABLE classics ADD isbn CHAR(13);
UPDATE classics SET isbn='9780123456789' WHERE year='1876';
```

Completamos la información y nos queda una base de datos del siguiente modo:

```
MariaDB [publications]> SELECT author, title, isbn FROM classics;
+--------------------+----------------------------+---------------+
| author             | title                      | isbn          |
+--------------------+----------------------------+---------------+
| Willian Shakespeare | Romeo y Julieta           | 9780123456588 |
| Charles Darwin     | El origen de las especies  | 9780123456589 |
| Jane Austen        | Orgullo y prejuicio        | 9780123456734 |
| Mark Twain         | Las aventuras de Tom Sawyer | 9780123456789 |
+--------------------+----------------------------+---------------+
4 rows in set (0,000 sec)
```

Ahora podemos establecer una clave primaria, PRIMARY KEY, para ello usaremos el isbn, como sigue:

```
MariaDB [publications]> ALTER TABLE classics ADD PRIMARY KEY (isbn);
Query OK, 0 rows affected (0,071 sec)

MariaDB [publications]> DESCRIBE classics;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| author   | varchar(128) | YES  | MUL | NULL    |       |
| title    | varchar(128) | YES  | MUL | NULL    |       |
| category | varchar(16)  | YES  | MUL | NULL    |       |
| year     | smallint(6)  | YES  | MUL | NULL    |       |
| isbn     | char(13)     | NO   | PRI | NULL    |       |
+----------+--------------+------+-----+---------+-------+
5 rows in set (0,001 sec)
```

Podríamos haber creado la clave principal en el momento de crear la tabla.

**Adicción clave primaria durante la creación de tablas.**

```
MariaDB [publications]> CREATE TABLE books2 (
    -> author VARCHAR(128),
    -> title VARCHAR(128),
    -> category VARCHAR(16),
    -> year SMALLINT,
    -> isbn CHAR(13),
    -> INDEX(author(20)),
    -> INDEX(title(20)),
    -> INDEX(category(4)),
    -> INDEX(year)
    -> PRIMARY KEY (isbn));
Query OK, 0 rows affected (0,021 sec)
```

## Consultas de bases de datos MySQL.

### SELECT.

El comando `SELECT` se usa para extraer datos de una tabla. Su sintaxis básica es la siguiente:

```
SELECT objetivo FROM nombre_de_table;
```

`objetivo` puede ser un *, que significa cada columna, o se pueden seleccionar solo ciertas columnas, por ejemplo:

```
MariaDB [publications]> SELECT author, title FROM classics;
+------------+----------------------------+
| author     | title                      |
+------------+----------------------------+
| Mark Twain | Las aventuras de Tom Sawyer |
+------------+----------------------------+
1 row in set (0,000 sec)

MariaDB [publications]> SELECT COUNT(*) FROM classics; /* Cuenta las columnas */
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0,000 sec)
```

### SELECT DISTINCT.

El calificador `DISTINCT` (distinto) o `DISTINCTROW` permite eliminar varias entradas cuando contienen los mismos datos, así si hubiera más de una entrada para el autor Mark Twain, con su uso solo se mostraría una:

```
MariaDB [publications]> SELECT DISTINCT author FROM classics;
+------------+
| author     |
+------------+
| Mark Twain |
+------------+
1 row in set (0,000 sec)

MariaDB [publications]> SELECT DISTINCT author,title FROM classics;
+------------+----------------------------+
| author     | title                      |
+------------+----------------------------+
| Mark Twain | Las aventuras de Tom Sawyer |
+------------+----------------------------+
1 row in set (0,000 sec)
```

### DELETE.

`DELETE` se usa para eliminar filas. Se permite acotar las filas a eliminar mediante los calificadores `WHERE` y `LIMIT`.

```
MariaDB [publications]> DELETE FROM classics
    -> WHERE title='Las aventuras de Tom Sawyer';
Query OK, 1 row affected (0,002 sec)

MariaDB [publications]> SELECT * FROM classics;
```

```
Empty set (0,000 sec)
```

La palabra clave WHERE es importantísima en SQL.

**WHERE.**

La palabra clave `WHERE` permite limitar las consultas devolviendo solo aquellas en las que una determinada expresión es verdadera.

```
SELECT author, title FROM classics WHERE author='Mark Twain';
```

`LIKE` permite hacer búsquedas en partes de cadenas. Este calificador se usa con el carácter % antes o después de algún texto, significando cualquier cosa antes o cualquier cosa después, por ejemplo:

```
SELECT author, title FROM classics WHERE author LIKE 'Mark%';
```

**LIMIT.**

El calificador `LIMIT` (Límite) permite seleccionar cuántas líneas se deben devolver en una consulta, y en qué lugar de la tabla hay que empezar a devolverlas. Cuando se pasa un solo parámetros se comienza al principio de los resultados y devuelve el número de líneas especificado con el parámetro. Si se le pasan dos parámetros el primero indica el desplazamiento desde el inicio de los resultados y el segundo indica cuántas líneas se deben devolver. En los siguientes ejemplos, el primero devuelve las tres primeras filas de la tabla y el segundo devuelve dos filas empezando en la posición 1 (se salta la primera fila, posición 0).

```
SELECT author, title FROM classics LIMIT 3;
SELECT author, title FROM classics LIMIT 1,2;
```

**UPDATE...SET.**

`UPDATE...SET` permite actualizar en contenido de un campo. Por ejemplo:

```
UPDATE classics SET author = 'Mark Twain (Samuel Langhorne Clemens)'
  WHERE author = 'Mark Twain';
```

**ORDER BY.**

`ORDER BY` ordena los resultados devueltos por una o más consultas en orden ascendente o descendente.

```
SELECT author, title FROM classics ORDER BY author;
SELECT author, title FROM classics ORDER BY title DESC;
```

La primera consulta devuelve los resultados en orden alfabético ascendente (por defecto), la segunda en orden descendente. Se pueden ordenar los resultados de más de una columna:

```
SELECT author, title, year FROM classics ORDER BY author ASC, year DESC;
```

**GROUP BY.**

De forma similar a ORDER BY se pueden agrupar los resultados con `GROUP BY`, que sirve para recuperar información sobre un grupo de datos.

```
MariaDB [publications]> SELECT category, COUNT(author) FROM classics
    -> GROUP BY category;
+-------------+---------------+
| category    | COUNT(author) |
+-------------+---------------+
| Ficción     |             2 |
| No Ficción  |             1 |
| Teatro      |             1 |
+-------------+---------------+
3 rows in set (0,000 sec)
```

En la consulta anterior hemos preguntado a MySQL cuántas publicaciones hay de cada categoría en la tabla classics, y además hemos pedido recuento de autores para cada categoría.

**Unión de tablas.**

Es bastante habitual mantener varias tablas dentro de una base de datos, cada una con un tipo de información diferente. Vamos a crear una tabla para clientes que han comprado alguna de las publicaciones que ya tenemos en nuestra base de datos:

```
MariaDB [publications]> CREATE TABLE customers (
    -> name VARCHAR(128),
    -> isbn VARCHAR(13),
    -> PRIMARY KEY (isbn));
Query OK, 0 rows affected (0,009 sec)
```

Y completamos la información:

```
MariaDB [publications]> INSERT INTO customers (name, isbn)
    -> VALUES('Juan Lozano', '9780123456588');
Query OK, 1 row affected (0,001 sec)

MariaDB [publications]> SELECT * FROM customers;
+-----------------+---------------+
| name            | isbn          |
+-----------------+---------------+
| Juan Lozano     | 9780123456588 |
| Alberto García  | 9780123456589 |
| Sonia Alvarado  | 9780123456789 |
+-----------------+---------------+
3 rows in set (0,000 sec)
```

Ahora podemos usar la columna isbn que tiene identificadores únicos para unir las dos tablas en una única consulta:

```
MariaDB [publications]> SELECT name, author, title FROM customers, classics
    -> WHERE customers.isbn = classics.isbn;
+-----------------+---------------------+-----------------------------+
| name            | author              | title                       |
+-----------------+---------------------+-----------------------------+
| Juan Lozano     | Willian Shakespeare | Romeo y Julieta             |
| Alberto García  | Charles Darwin      | El origen de las especies   |
| Sonia Alvarado  | Mark Twain          | Las aventuras de Tom Sawyer |
+-----------------+---------------------+-----------------------------+
3 rows in set (0,002 sec)
```

Esta consulta ha enlazado claramente las tablas para mostrar las publicaciones compradas de la tabla classics por las personas de la tabla customers.

Podemos hacer lo mismo que en la consulta anterior con la clausula NATURAL JOIN. Este tipo de unión utiliza dos tablas y automáticamente une las columnas que tienen el mismo nombre.

```
SELECT name, author, title FROM customers NATURAL JOIN classics;
```

Si se desea especificar la columna en la que unir dos tablas, se usa el constructor JOIN...ON, de la siguiente manera, para obtener idénticos resultados a los del ejemplo anterior:

```
MariaDB [publications]> SELECT name, author, title FROM customers
    -> JOIN classics ON customers.isbn = classics.isbn;
+-----------------+---------------------+-----------------------------+
| name            | author              | title                       |
+-----------------+---------------------+-----------------------------+
| Juan Lozano     | Willian Shakespeare | Romeo y Julieta             |
| Alberto García  | Charles Darwin      | El origen de las especies   |
| Sonia Alvarado  | Mark Twain          | Las aventuras de Tom Sawyer |
+-----------------+---------------------+-----------------------------+
3 rows in set (0,000 sec)
```

**Uso de alias (AS).**

Los alias pueden ser especialmente útiles cuando se tienen consultas largas que hacen referencia a los mismos nombres de tabla muchas veces.

```
MariaDB [publications]> SELECT name, author, title FROM
    -> customers AS cust, classics AS class WHERE cust.isbn = class.isbn;
+-----------------+---------------------+-----------------------------+
| name            | author              | title                       |
+-----------------+---------------------+-----------------------------+
| Juan Lozano     | Willian Shakespeare | Romeo y Julieta             |
| Alberto García  | Charles Darwin      | El origen de las especies   |
| Sonia Alvarado  | Mark Twain          | Las aventuras de Tom Sawyer |
+-----------------+---------------------+-----------------------------+
3 rows in set (0,000 sec)
```

También se puede usar AS para renombrar una columna (ya sea unido tablas o no), nos referimos a esto:

```
SELECT name AS customer FROM customers ORDER BY customer;
```

```
+-----------------+
| customer        |
+-----------------+
| Alberto García  |
| Juan Lozano     |
| Sonia Alvarado  |
+-----------------+
3 rows in set (0,000 sec)
```

**Uso de operadores lógicos AND, OR y NOT.**

Es posible usar los operadores lógicos AND, OR y NOT en las consultas WHERE de MySQL para limitar aún más las selecciones. Se pueden mezclar y usar del modo en que se necesiten. Por ejemplo:

```
SELECT author, title FROM classics WHERE
  author LIKE 'Charles%' AND author NOT LIKE '%Darwin';
```

**Funciones en MySQL.**

MySQL tiene funciones integradas que reducen sustancialmente el tiempo necesario para hacer consultas complejas, así como su complejidad. Se pueden consultar todas las funciones disponibles así como mucha otra información en el manual de referencia de MySQL en https://dev.mysql.com/doc/refman/8.0/en/.

En MySQL, una función es un bloque de código que realiza una tarea específica y devuelve un resultado. Pueden ser predefinidas (también conocidas como funciones integradas) o definidas por el usuario. Las funciones se utilizan para simplificar tareas repetitivas, realizar cálculos, manipular datos y transformarlos de diversas formas.

Aquí tienes un ejemplo de una función de MySQL que calcula el promedio de una columna en una tabla:

```
SELECT AVG(columna) AS promedio FROM tabla;
```

En este ejemplo, la función AVG() se utiliza para calcular el promedio de los valores en la columna especificada de la tabla. La función AVG() es una función predefinida de MySQL que devuelve el promedio de los valores numéricos.

Algunas características importantes de las funciones en MySQL son:

- Nombre de la función: Las funciones en MySQL tienen un nombre específico que las identifica, como AVG(), SUM(), COUNT(), etc. Estos nombres son utilizados para invocar y utilizar las funciones en consultas SQL.

- Parámetros: Las funciones pueden tener cero o más parámetros. Los parámetros son valores que se pasan a la función para que realice su operación. Algunas funciones pueden requerir parámetros obligatorios, mientras que otros pueden tener parámetros opcionales.

- Valor de retorno: Las funciones en MySQL pueden devolver un valor como resultado de su operación. El valor de retorno puede ser de diferentes tipos de datos, como numéricos, cadenas, fechas, booleanos, etc.

- Funciones predefinidas y definidas por el usuario: MySQL proporciona muchas funciones predefinidas que se pueden utilizar directamente en consultas SQL. También es posible crear funciones definidas por el usuario utilizando el lenguaje de programación SQL de MySQL.

Las funciones en MySQL son una parte fundamental del lenguaje de consulta SQL y proporcionan una amplia gama de operaciones para manipular y transformar datos en una base de datos. Pueden ser utilizadas en consultas SELECT, INSERT, UPDATE, DELETE y otras sentencias SQL para realizar cálculos, agregaciones, manipulación de cadenas, fechas y mucho más.

**DOMINIO DE MySQL – DISEÑO DE BASES DE DATOS.**

A la hora de plantear el diseño de una base de datos es bueno hacerse las preguntas a cerca de qué consultas pudieran realizarse sobre la misma, por ejemplo, en la base de datos para una librería estas podrían ser algunas de las consultas:

¿Cuántos autores, libros y clientes habrá en la base de datos?
¿Qué autor escribió cierto libro?
¿Qué libros ha escrito un determinado autor?
¿Cuál es el libro más caro?
¿Cuál es el libro más vendido?
…

Tendremos que plantear un esquema de tablas válido. Cuando los datos tienen alguna relación entre sí podrían estar en la misma tabla y cuando apenas existe relación entre ellos deberían ir en tablas separadas. Para una librería se podría plantear un esquema de base de datos con tres tablas: Autores, Libros y Clientes.

**Claves principales: las claves de las bases de datos relacionales.**

En el diseño de una base de datos lo que interesa e los vínculos entre las tablas, siguiendo con nuestro esquema de tres tablas (Autores, libros y clientes) nos interesa saber quién escribió qué libro y quién lo compró. Las tablas están separadas, por ello es preciso crear vínculos entre ellas. Esto se consigue dando a cada autor, libro y cliente un identificador único y para este identificador usaremos la PRIMARY KEY (Clave principal). Para un libro será su isbn, para un cliente su DNI y para los autores puede ser una clave arbitraria mediante el uso de AUTO_INCREMENT.

Al proceso de separar los datos en tablas y crear claves principales se le denomina normalización. Existen tres esquemas separados de normalización llamados primera, segunda y tercera formas normales. Si se modifica una base de datos para satisfacer cada una de estas formas en su orden, se podrá tener la seguridad de que la base de datos está perfectamente equilibrada para conseguir un acceso rápido y un mínimo consumo de memoria y uso de espacio en disco.

**Primera forma normal.**

Para que una base de datos satisfaga la primera forma normal, debe cumplir res requisitos:

- No debe haber columnas repetidas que contengan el mismo tipo de datos.
- Todas las columnas deben contener un único valor.
- Debe haber una clave principal para identificar de manera única cada fila.

Desde una perspectiva más técnica:

La Primera Forma Normal (1NF) es la primera etapa de la normalización de una base de datos. Se define como una condición en la que los datos en una tabla están organizados de manera que no existen grupos repetitivos o campos multivaluados. El objetivo principal de la 1NF es eliminar la redundancia de datos y asegurar la integridad de los mismos.

Las condiciones que deben cumplirse para que una tabla esté en 1NF son las siguientes:

- No deben haber campos multivaluados: Cada campo en una tabla debe contener un solo valor, es decir, no puede haber múltiples valores en un solo campo. Si existe la necesidad de almacenar múltiples valores, se debe crear una nueva tabla y establecer una relación entre ambas.

- Cada columna debe tener un nombre único: Los nombres de las columnas (atributos) en una tabla deben ser únicos. Esto permite identificar y acceder a cada columna de manera unívoca.

- Cada celda debe contener un solo valor: En una tabla en 1NF, cada celda (intersección entre una fila y una columna) debe contener un solo valor. No puede haber celdas que contengan múltiples valores o listas separadas por comas, por ejemplo.

- Cada registro debe ser único: Cada fila en una tabla debe ser única, lo que significa que no puede haber registros duplicados. Esto se logra al incluir una clave primaria que identifique de forma única cada registro.

Cumplir con la Primera Forma Normal es esencial antes de avanzar a las etapas posteriores de normalización, como la Segunda Forma Normal (2NF) y la Tercera Forma Normal (3NF). Al aplicar la 1NF, se mejora la estructura de la base de datos, se reduce la redundancia de datos y se facilita el mantenimiento y consulta de la información.

**Segunda forma normal.**

La primera forma normal trata con datos duplicados en varias columnas. La segunda forma normal trata sobre la redundancia en varias filas. Para alcanzar la segunda forma normal, las tablas deben cumplir antes la primera forma normal. Una vez conseguido esto, se alcanza la segunda forma normal identificando las columnas cuyos datos se repiten en diferentes lugares y luego los elimina de sus propias tablas.

Desde una perspectiva más técnica:

La Segunda Forma Normal (2NF) es una etapa de la normalización de una base de datos que busca eliminar la redundancia de datos relacionada con dependencias parciales. La 2NF se basa en la Primera Forma Normal (1NF) y establece condiciones adicionales para la estructura de una tabla.

La definición y condiciones de la Segunda Forma Normal son las siguientes:

- Cumplir con la Primera Forma Normal (1NF): Antes de aplicar la 2NF, la tabla debe estar en 1NF, lo que implica que los datos deben estar organizados sin grupos repetitivos o campos multivaluados.

- Eliminar dependencias parciales: La 2NF busca eliminar las dependencias parciales en una tabla. Una dependencia parcial ocurre cuando una columna no clave depende solo de una parte de la clave primaria, es decir, depende de una porción de los valores clave.

- Crear nuevas tablas si es necesario: Para eliminar las dependencias parciales, se pueden crear nuevas tablas que contengan los atributos dependientes y establecer relaciones entre las tablas mediante claves primarias y claves externas.

En resumen, la Segunda Forma Normal (2NF) establece que una tabla debe estar en 1NF y debe eliminar las dependencias parciales. Al aplicar la 2NF, se evita la

duplicación de datos y se mejora la estructura de la base de datos, lo que facilita la gestión y consulta de la información.

Es importante tener en cuenta que existen más formas normales, como la Tercera Forma Normal (3NF), la Cuarta Forma Normal (4NF) y otras posteriores, que se utilizan para lograr una mayor reducción de la redundancia y una mejor organización de los datos. Cada forma normal tiene condiciones específicas que deben cumplirse para normalizar una base de datos.

**Tercera forma normal.**

Una vez que se tenga una base de datos que cumpla con la primera y segunda formas normales, estará en bastante buena forma y puede que no haya que modificarla más. Sin embargo, si se quiere ser más estricto con ella, podemos asegurarnos de que cumple con la tercera forma normal. Esta forma requiere que los datos que no son estrictamente dependientes de la clave principal, pero que dependen de otro valor de la tabla, también deberían tratarse en tablas separadas, en función de la dependencia.

Desde una perspectiva más técnica:

La Tercera Forma Normal (3NF) es una etapa de la normalización de una base de datos que busca eliminar la redundancia de datos relacionada con dependencias transitivas. La 3NF se basa en la Segunda Forma Normal (2NF) y establece condiciones adicionales para la estructura de una tabla.

La definición y condiciones de la Tercera Forma Normal son las siguientes:

- Cumplir con la Segunda Forma Normal (2NF): Antes de aplicar la 3NF, la tabla debe estar en 2NF, lo que implica que los datos deben estar organizados sin dependencias parciales.

- Eliminar dependencias transitivas: La 3NF busca eliminar las dependencias transitivas en una tabla. Una dependencia transitiva ocurre cuando una columna no clave depende de otra columna no clave en lugar de depender directamente de la clave primaria.

- Mover atributos no clave a nuevas tablas: Si una columna no clave depende de otra columna no clave en lugar de depender directamente de la clave primaria, se debe mover esa columna a una nueva tabla. La nueva tabla tendrá su propia clave primaria y se establecerá una relación entre las tablas mediante claves primarias y claves externas.

La normalización en 3NF asegura que no haya dependencias transitivas entre las columnas no clave de una tabla, evitando la redundancia de datos y mejorando la integridad de la base de datos.

Es importante destacar que la normalización no se detiene en la Tercera Forma Normal. En algunas situaciones, puede ser necesario aplicar formas normales más avanzadas, como la Cuarta Forma Normal (4NF) o la Quinta Forma Normal (5NF), para abordar situaciones específicas de redundancia o dependencias complejas en los datos.

**Ejemplo de normalización.**

Vamos a partir de una tabla, presentada solo con fines didácticos, que contiene datos repetidos. Vamos a normalizarla siguiendo los pasos mencionados. Aplicaremos las reglas de normalización para subdividirla en otras tablas, eliminar datos duplicados y conseguir una base de datos optimizada. La tabla es la siguiente:

| Author 1 | Author 2 | Title | ISBN | Price | Customer name | Customer address | Purchase date |
|---|---|---|---|---|---|---|---|
| *David Sklar* | *Adam Trachtenberg* | *PHP Cookbook* | *0596101015* | *44.99* | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| Danny Goodman | | Dynamic HTML | 0596527403 | 59.99 | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| Hugh E. Williams | David Lane | PHP and MySQL | 0596005436 | 44.95 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| *David Sklar* | *Adam Trachtenberg* | *PHP Cookbook* | *0596101015* | *44.99* | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| Rasmus Lerdorf | Kevin Tatroe & Peter MacIntyre | Programming PHP | 0596006815 | 39.99 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

Condiciones de la 1FN:

Tanto la columna 1 como la columna 2 tienen datos repetidos, por lo tanto ya tendríamos una columna para llevarla a una tabla separada, ya que las columnas repetidas de `Author` violan la regla número 1 de la 1FN.

El último libro, Programming PHP, cuenta con tres autores, en la columna número dos se incorporan dos autores, y esto viola la segunda regla de la 1FN.

La regla número 3 de la 1FN se cumple, dado que la clave principal del ISBN está creada.

Eliminemos las columnas referentes a los autores (tabla temp1):

| Title | ISBN | Price | Customer name | Customer address | Purchase date |
|---|---|---|---|---|---|
| *PHP Cookbook* | *0596101015* | *44.99* | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| Dynamic HTML | 0596527403 | 59.99 | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| PHP and MySQL | 0596005436 | 44.95 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| *PHP Cookbook* | *0596101015* | *44.99* | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| Programming PHP | 0596006815 | 39.99 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

La nueva tabla de autores es pequeña y sencilla. Tabla `authors`:

| ISBN | Author |
|---|---|
| 0596101015 | David Sklar |
| 0596101015 | Adam Trachtenberg |
| 0596527403 | Danny Goodman |
| 0596005436 | Hugh E. Williams |
| 0596005436 | David Lane |
| 0596006815 | Rasmus Lerdorf |
| 0596006815 | Kevin Tatroe |
| 0596006815 | Peter MacIntyre |

Los autores adicionales a cada título tienen sus propias filas, como puede observarse. Al principio puede parecer una tabla extraña dado que no arroja

información sobre qué auto escribió qué libro, pero MySQL puede decirlo rápidamente.

IMPORTANTE: El `ISBN` será la PRIMARY KEY de la tabla `books` que se creará con posterioridad, pero no es clave principal en la tabla `authors` que acabamos de ver. En esta tabla el `ISBN` es solo una columna, que con el propósito de acelerar las búsquedas haremos que sea clave, pero no la clave principal. De hecho no puede ser la clave principal en esta tabla porque no es única para ella, puede haber dos autores que hayan trabajado en el mismo `ISBN`. Debido a que se usará para enlazar autores a libros en otra tabla, a esta columna se le llama **clave externa**.

Condiciones DE LA 2FN:

En la tabla temp1 se puede ver como hay clientes que compraron dos libros y por lo tanto sus datos están duplicados en dos filas diferentes y esto rompe la 2FN. Por lo tanto la columnas referentes a los clientes se necesitan llevar a su propia tabla. Así nos quedaríamos con la nueva tabla `books` que sigue:

| ISBN | Title | Price |
|---|---|---|
| 0596101015 | PHP Cookbook | 44.99 |
| 0596527403 | Dynamic HTML | 59.99 |
| 0596005436 | PHP and MySQL | 44.95 |
| 0596006815 | Programming PHP | 39.99 |

Ahora disponemos de una tabla eficiente y autónoma, que satisface los requisitos de la 1FN y la 2FN. Sin embargo en la nueva tabla `customers` aún hay detalles duplicados:

| ISBN | Customer name | Customer address | Purchase date |
|---|---|---|---|
| 0596101015 | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| 0596527403 | Darren Ryder | 4758 Emily Drive, Richmond, VA 23219 | Dec 19 2008 |
| 0596005436 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| 0596101015 | Darren Ryder | 4758 Emily Drive, Richmond, VA 23219 | Dec 19 2008 |
| 0596006815 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

Tenemos que dividir esta tabla para asegurarnos que los datos de cada cliente solo se han introducido una vez y dado que el `ISBN` no es ni puede ser utilizado como clave principal para identificar clientes (o autores) se debe crear una nueva clave.

La siguiente tabla es la forma de normalizar la tabla customers cumpliendo la primera y la segunda formas normales: Cada cliente tiene ahora un número de cliente único que es la clave principal de la tabla. La dirección se ha separado en columnas diferentes.

| CustNo | Name | Address | City | State | Zip |
|---|---|---|---|---|---|
| 1 | Emma Brown | 1565 Rainbow Road | Los Angeles | CA | 90014 |
| 2 | Darren Ryder | 4758 Emily Drive | Richmond | VA | 23219 |
| 3 | Earl B. Thurston | 862 Gregory Lane | Frankfort | KY | 40601 |
| 4 | David Miller | 3647 Cedar Lane | Waltham | MA | 02154 |

Y los datos de las compras se colocan ahora en una nueva tabla llamada `purchases`, de lo contrario habría múltiples entradas de los detalles de los clientes para cada libro comprado.

| CustNo | ISBN | Date |
|---|---|---|
| 1 | 0596101015 | Mar 03 2009 |
| 2 | 0596527403 | Dec 19 2008 |
| 2 | 0596101015 | Dec 19 2008 |
| 3 | 0596005436 | Jun 22 2009 |
| 4 | 0596006815 | Jan 16 2009 |

Ahora tenemos las siguientes tablas: `authors`, `books`, `customers` y `purchases`. Podemos enlazar cada tabla con cualquier otra usando las claves `CustNo` e `ISBN`.

Condiciones de la 3FN:

En este caso no sería necesario acudir a la 3FN, pero podríamos hacerlo con la tabla customers y seguir dividiéndola en otras subtablas. Además como ejemplo ilustrativo a quedado claro el propósito de la aplicación de la normalización. La decisión de usar la 3FN puede ser difícil de valorar y debería basarse en los datos que se puedan necesitar añadir en el futuro.

**Cuando no utilizar la normalización.**

Aunque la normalización es un principio fundamental en el diseño de bases de datos y generalmente se recomienda seguir las formas normales, hay casos en los que la normalización estricta puede no ser la mejor opción. A continuación, se presentan algunas situaciones en las que la normalización puede estar desaconsejada:

- Bases de datos de muy bajo nivel: En algunos casos, especialmente en bases de datos de muy bajo nivel o sistemas de tiempo real, la desnormalización puede ser preferible. Esto se debe a que la desnormalización puede mejorar el rendimiento al reducir la cantidad de tablas y las uniones necesarias en las consultas. Sin embargo, se debe hacer un análisis cuidadoso de los beneficios y las implicaciones en términos de integridad de los datos antes de optar por la desnormalización en estos casos.

- Bases de datos con requisitos de rendimiento extremo: En situaciones en las que se requiere un rendimiento extremadamente alto y el tiempo de respuesta de las consultas es crítico, puede ser necesario desnormalizar la base de datos. Esto implica duplicar algunos datos y evitar operaciones de unión complejas. Sin embargo, es importante tener en cuenta que la desnormalización puede aumentar la complejidad de mantenimiento y puede requerir estrategias adicionales para mantener la integridad de los datos.

- Bases de datos de almacenamiento de datos analíticos (OLAP): En entornos de almacenamiento de datos analíticos, donde se realizan consultas complejas y se necesitan informes ad hoc, a menudo se utiliza la desnormalización para mejorar el rendimiento y la capacidad de análisis. La desnormalización puede facilitar las operaciones de agregación y reducir el tiempo de respuesta en consultas analíticas complejas.

- Aplicaciones con requisitos específicos de integridad de datos: En ciertos casos, las aplicaciones pueden tener requisitos específicos de integridad de datos que dificultan el cumplimiento de las formas normales. Si los requisitos de negocio o las reglas de negocio hacen que sea difícil mantener la integridad de los datos siguiendo las formas normales, puede ser necesario realizar ciertas desviaciones o compromisos en la normalización.

En resumen, aunque la normalización es una práctica generalmente recomendada, existen situaciones en las que la desnormalización puede ser preferible debido a requisitos de rendimiento, necesidades de almacenamiento de datos analíticos o requisitos específicos de integridad de datos. Sin embargo, es importante sopesar cuidadosamente los beneficios y las implicaciones antes de decidir desviarse de las formas normales en el diseño de una base de datos.

**Relaciones.**

De MySQL se dice que es un sistema de bases de datos relacionales. Sus tablas no almacenas solo datos, sino también relaciones entre esos datos. Hay tres categorías de relaciones.

Relaciones 'uno a uno'.

Cada elemento tiene relación con un solo elemento de otro tipo. Esto es extremadamente raro. Por ejemplo, una dirección puede estar asociada a múltiples clientes, rompiendo esta relación uno a uno.

Relaciones 'uno a muchos'.

Las relaciones uno a muchos (o muchos a uno) ocurren cuando una fila en una tabla está relacionada a muchas filas de otra tabla. Por ejemplo, un cliente que realiza varias compras de libros.

Para representar una relación de uno a muchos en una base de datos relacional, se crea una tabla para el muchos y una tabla para el uno. La tabla para los muchos debe contener una columna que lista la clave principal d ella tabla para el uno. Por lo tanto, en el ejemplo de la sección anterior, la tabla purchases contiene una columna que lista la clave principal de la tabla customers, la columna CustNo.

Relaciones 'muchos a muchos'.

Muchas filas de una tabla están enlazadas a muchas filas de otra tabla. Para crear esta tercera relación se añade otra tabla que contenga la misma clave de cada una de las otras tablas. Esta tercera tabla no contiene nada más, ya que su único propósito es el de conectar las otras tablas. Con una tabla de este tipo (intermediaria) se puede recorrer toda la información de la base de datos a través de una serie de relaciones. En otras palabras: una relación muchos a muchos es una asociación entre dos tablas en la que múltiples registros de una tabla están relacionados con múltiples registros de otra tabla. Esta relación se establece mediante el uso de una tabla intermedia (también conocida como tabla de unión o tabla puente) que contiene claves externas que enlazan los registros de ambas tablas.

En resumen, en una relación muchos a muchos en MySQL se utiliza una tabla intermedia para enlazar dos tablas principales. Esto permite representar y gestionar eficientemente asociaciones complejas entre los registros de las tablas en una base de datos relacional.

**ACCESO A MySQL MEDIANTE PHP.**

PHP cuenta con funciones integradas para acceder a MySQL. La razón de usar PHP como interfaz de MySQL es para formatear los resultados de las consultas SQL en un formulario visible en una página web.

El proceso es sencillo. PHP se conecta a MySQL y selecciona la base de datos a utilizar. Prepara una cadena de consulta y realizar la consulta. Recupera los resultados para enviarlos por ejemplo a una página web. Finalmente se desconecta de MySQL.

[config.inc.php]

```php
<?php
$hn = 'localhost';
$db = 'publications';
$un = 'user';
$pw = 'qwerty';
?>
```

[index.php]

```php
<?php
require_once 'config.inc.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die ('Error fatal');
?>
```

En index.php se crea un objeto denominado $conn al llamar a una nueva instancia del método mysqli() y pasa todos los valores recuperados del archivo config.inc.php. Logramos la comprobación de errores mediante la referencia a la propiedad `$conn→connect_error`.

El operador → indica que el elemento de la derecha es una propiedad o método del objeto de la izquierda. En este caso, si connect_error tiene un valor, se ha producido un error, por lo que llamamos a la función die para finalizar el programa.

**Creación y ejecución de una consulta.**

Enviar una consulta a MySQL desde PHP es tan sencillo como incluir el SQL correspondiente en el método query() de un objeto de conexión.

```php
<?php
$query = "SELECT * FROM classics";
$result = $conn->query($query);
if(!$result) die ('Error fatal');
?>
```

Ahora tenemos almacenados los datos de nuestra consulta en la variable `$result`. Pero son datos en bruto, necesitamos formatearlos para poder presentarlos correctamente en un navegador. En el siguiente ejemplo vemos como formatear correctamente esta información, haciendo uso de los métodos `num_row`, `data_seek()` y `fetch_assoc()`.

[index.php]

```php
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
```

```php
  if ($conn->connect_error) die("Fatal Error");

  $query  = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die("Fatal Error");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    echo 'Autor: '. htmlspecialchars($result->fetch_assoc()['author']).'<br>';
    $result->data_seek($j);
    echo 'Título: '. htmlspecialchars($result->fetch_assoc()['title']).'<br>';
    $result->data_seek($j);
    echo 'Categoría: '. htmlspecialchars($result->fetch_assoc()['category']).
         '<br>';
    $result->data_seek($j);
    echo 'Año: '. htmlspecialchars($result->fetch_assoc()['year']).'<br>';
    $result->data_seek($j);
    echo 'ISBN: '. htmlspecialchars($result->fetch_assoc()['isbn']).'<br><br>';
  }

  $result->close();
  $conn->close();
?>
```

[1] `num_rows` es una función que se utiliza en conjunto con una consulta a una base de datos para obtener el número de filas o registros devueltos por la consulta. Esta función es comúnmente utilizada en combinación con otras funciones y métodos para manipular y mostrar los resultados de las consultas. Como hemos hecho aquí.

En este ejemplo, la función num_rows se utiliza en el objeto `$result` para obtener el número de filas devueltas por la consulta. Es importante destacar que la función num_rows solo puede ser utilizada después de ejecutar una consulta y obtener un resultado válido.

[2] `data_seek` es un método utilizado en la extensión MySQLi para desplazar el puntero del conjunto de resultados a una fila específica. Este método se llama en un objeto $result, que representa el resultado de una consulta ejecutada mediante MySQLi.

El método data_seek se utiliza para mover el puntero del resultado a una fila determinada, lo que permite acceder a datos específicos dentro del conjunto de resultados.

Es importante destacar que data_seek acepta un parámetro que indica el número de fila al que se desea desplazar el puntero. Los índices de fila comienzan en 0, por lo que data_seek(0) se utiliza para volver al inicio del conjunto de resultados.

[3] `fetch_assoc` es un método utilizado en la extensión MySQLi para obtener la siguiente fila de un conjunto de resultados como un arreglo asociativo. Este método se llama en el objeto $result, que representa el resultado de una consulta ejecutada mediante MySQLi.

El método fetch_assoc se utiliza para obtener los datos de la siguiente fila del resultado y devolverlos como un arreglo asociativo, donde las claves del arreglo son los nombres de las columnas y los valores son los valores de las columnas correspondientes.

Es importante destacar que fetch_assoc se utiliza para obtener una fila a la vez del conjunto de resultados. Cada vez que se llama a este método, el puntero del resultado se mueve a la siguiente fila.

[4] htmlspecialchars, es una función en PHP que se utiliza para convertir ciertos caracteres especiales en entidades HTML. Esto es útil para evitar problemas de seguridad al mostrar datos en una página web, ya que evita que los caracteres especiales sean interpretados como parte del código HTML.

La función htmlspecialchars toma una cadena de texto como entrada y reemplaza los caracteres especiales por sus entidades HTML correspondientes.

Esto ayuda a prevenir ataques de inyección de código, como el Cross-Site Scripting (XSS), al asegurarse de que los caracteres especiales se muestren como texto literal en lugar de ser interpretados como parte del código HTML.

Es importante destacar que htmlspecialchars se utiliza principalmente al mostrar datos en una página web. No se debe utilizar al almacenar datos en una base de datos, ya que los datos deben ser escapados de manera apropiada según el contexto de almacenamiento (por ejemplo, utilizando consultas preparadas o funciones específicas de la extensión de base de datos que estés utilizando).

Formateada para un navegador, la salida del código anterior se vería así:

```
Author: Willian Shakespeare
Title: Romeo y Julieta
Category: Teatro
Year: 1856
ISBN: 9780123456588

Author: Charles Darwin
Title: El origen de las especies
Category: No Ficción
Year: 1856
ISBN: 9780123456589

Author: Jane Austen
Title: Orgullo y prejuicio
Category: Ficción
Year: 1811
ISBN: 9780123456734

Author: Mark Twain
Title: Las aventuras de Tom Sawyer
Category: Ficción
Year: 1876
ISBN: 9780123456789
```

**El método fetch_array.**

Podemos mejorar el código anterior mediante el uso del método fetch_array. Lo que hacemos en él es modificar el contenido del bucle for. El siguiente código ofrece exactamente el mismo resultado en el navegador:

[index.php]

```php
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
```

```php
  if ($conn->connect_error) die("Fatal Error");

  $query  = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die("Fatal Error");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result->fetch_array(MYSQLI_ASSOC);

    echo 'Autor: '    . htmlspecialchars($row['author'])    . '<br>';
    echo 'Título: '     . htmlspecialchars($row['title'])     . '<br>';
    echo 'Categoría: ' . htmlspecialchars($row['category']) . '<br>';
    echo 'Año: '      . htmlspecialchars($row['year'])      . '<br>';
    echo 'ISBN: '      . htmlspecialchars($row['isbn'])      . '<br><br>';
  }

  $result->close();
  $conn->close();
?>
```

[1] El método `fetch_array` es un método utilizado en la extensión MySQLi para obtener la siguiente fila de un conjunto de resultados como un arreglo. Este método se llama en un objeto mysqli_result, que representa el resultado de una consulta ejecutada mediante MySQLi.

El método fetch_array se utiliza para obtener los datos de la siguiente fila del resultado y devolverlos como un arreglo, donde los índices del arreglo pueden ser tanto numéricos como los nombres de las columnas.

En este ejemplo, se ejecuta una consulta SQL y se obtiene el objeto $resultado que representa el resultado de la consulta. Luego, se utiliza el método fetch_array para obtener la siguiente fila del resultado como un arreglo llamado $fila. Los índices del arreglo pueden ser tanto numéricos (0, 1, 2, etc.) como los nombres de las columnas de la tabla, como es el caso.

Es importante tener en cuenta que fetch_array puede devolver un arreglo con duplicados, es decir, puede contener tanto índices numéricos como nombres de columnas. Si deseas obtener solo los nombres de las columnas como índices en el arreglo, puedes utilizar el método fetch_assoc en su lugar, como hemos hecho anteriormente.

El método fetch_array puede devolver tres tipos de matrices en función del valor que se le pase, uno de ellos es el siguiente:

[2] `MYSQLI_ASSOC` es una constante utilizada como argumento en el método fetch_array de la extensión MySQLi. Esta constante indica que se desea obtener un arreglo asociativo como resultado, es decir, un arreglo donde los índices corresponden a los nombres de las columnas de la tabla.

Cuando se utiliza MYSQLI_ASSOC como argumento en el método fetch_array, se obtiene un arreglo que contiene solo los nombres de las columnas de la tabla como claves, y los valores correspondientes a cada columna.

Utilizar MYSQLI_ASSOC es útil cuando se desea acceder a los valores de las columnas por su nombre en lugar de utilizar índices numéricos. Si se utiliza fetch_array sin especificar MYSQLI_ASSOC, se obtendrá un arreglo que contiene tanto índices numéricos como nombres de columnas.

Los otros dos son los siguientes:

```
MYSQLI_NUM
```

Matriz numérica. Cada columna aparece en la matriz en el orden que se indicó cuando se creó o alteró la tabla.

```
MYSQLI_BOTH
```

Matriz asociativa y numérica.

Las matrices asociativa suelen ser más útiles que las numeŕicas porque se puede hacer referencia a caa columna por nombre, como `$row['author']`, en lugar de tratar de recordar dónde se encuentra dentro del orden que siguen las columnas. Este script utiliza una matriz asociativa, que nos lleva a pasar el tipo MYSQLI_ASSOC.

**Cierre de la conexión.**

Se ha podido observar como las llamadas a los métodos close() de los objetos $result y $conn de los scripts anteriores se emiten tan pronto como cada objeto ya no es necesario, del siguiente modo:

```
$result->close();
$conn->close();
```

**Un ejemplo práctico con lo aprendido hasta ahora.**

Veamos un ejemplo de inserción y borrado de datos en una tabla MySQL usando PHP. El código es el siguiente:

```
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  if (isset($_POST['delete']) && isset($_POST['isbn']))
  {
    $isbn   = get_post($conn, 'isbn');
    $query  = "DELETE FROM classics WHERE isbn='$isbn'";
    $result = $conn->query($query);
    if (!$result) echo "Falló el borrando<br><br>";
  }

  if (isset($_POST['author'])   &&
      isset($_POST['title'])    &&
      isset($_POST['category']) &&
      isset($_POST['year'])     &&
      isset($_POST['isbn']))
  {
    $author   = get_post($conn, 'author');
    $title    = get_post($conn, 'title');
    $category = get_post($conn, 'category');
    $year     = get_post($conn, 'year');
    $isbn     = get_post($conn, 'isbn');
    $query    = "INSERT INTO classics VALUES" .
      "('$author', '$title', '$category', '$year', '$isbn')";
    $result   = $conn->query($query);
    if (!$result) echo "Falló la inserción<br><br>";
  }
```

```
  echo <<<_END
  <form action="index.php" method="post"><pre>
    Autor     <input type="text" name="author">
    Título    <input type="text" name="title">
    Categoría <input type="text" name="category">
    Año       <input type="text" name="year">
    ISBN      <input type="text" name="isbn"></br>
            <input type="submit" value="AÑADIR ENTRADA">
  </pre></form>
_END;

  $query  = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die ("Acceso fallido a la base de datos");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result->fetch_array(MYSQLI_NUM);

    $r0 = htmlspecialchars($row[0]);
    $r1 = htmlspecialchars($row[1]);
    $r2 = htmlspecialchars($row[2]);
    $r3 = htmlspecialchars($row[3]);
    $r4 = htmlspecialchars($row[4]);

    echo <<<_END
  <pre>
    Author $r0
     Title $r1
  Category $r2
      Year $r3
      ISBN $r4
  </pre>
  <form action='index.php' method='post'>
  <input type='hidden' name='delete' value='yes'>
  <input type='hidden' name='isbn' value='$r4'>
  <input type='submit' value='BORRAR ENTRADA'></form>
_END;
  }

  $result->close();
  $conn->close();

  function get_post($conn, $var)
  {
    return $conn->real_escape_string($_POST[$var]);
  }
?>
```

El código anterior un script en PHP que realiza operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en una base de datos MySQL. Aquí se proporciona una explicación paso a paso del código:

- El código incluye un archivo de configuración `config.inc.php` utilizando la función `require_once`, que contiene las variables necesarias para la conexión a la base de datos.

- Se establece una conexión a la base de datos utilizando la clase `mysqli` y se almacena en la variable `$conn`. Si la conexión falla, se muestra un mensaje de error y se detiene la ejecución del script.

- El código verifica si se ha enviado una solicitud `HTTP POST` con las claves `'delete'` e `'isbn'`. Si es así, recupera el valor de `'isbn'` utilizando la función `get_post` (que se explicará más adelante), construye una consulta SQL para eliminar una fila de la tabla `'classics'` y la ejecuta. Si la consulta no se ejecuta correctamente, se muestra un mensaje de error.

- El código también verifica si se han enviado las claves `'author'`, `'title'`, `'category'`, `'year'` e `'isbn'` en una solicitud `HTTP POST`. Si es así, recupera los valores correspondientes utilizando la función `get_post` (ver más abajo), construye una consulta SQL para insertar una nueva fila en la tabla `'classics'` con los valores proporcionados y la ejecuta. Si la consulta no se ejecuta correctamente, se muestra un mensaje de error.

- Se muestra un formulario HTML que permite al usuario ingresar datos para añadir una nueva entrada en la base de datos. El formulario envía los datos al mismo archivo PHP (`index.php`) utilizando el método POST.

- Se realiza una consulta SQL para seleccionar todos los registros de la tabla `'classics'` y se almacena en la variable `$result`. Si la consulta no se ejecuta correctamente, se muestra un mensaje de error y se detiene la ejecución del script.

- Se obtiene el número de filas en el conjunto de resultados utilizando el método `num_rows` de `$result`.

- Se recorren todas las filas del conjunto de resultados utilizando un bucle for. En cada iteración del bucle, se utiliza el método `fetch_array` con el argumento `MYSQLI_NUM` para obtener un arreglo numérico que representa una fila de la tabla `'classics'`. Luego, se utilizan las funciones `htmlspecialchars` para evitar problemas de seguridad al mostrar los valores de la fila en la salida HTML.

- Se muestra la información de cada fila en un formato predefinido utilizando la sintaxis de heredoc (`<<<_END ... _END;`). Además, se muestra un formulario para borrar la entrada correspondiente utilizando el método `POST`. El valor del `ISBN` se envía como un campo oculto en el formulario.

- Se cierra el conjunto de resultados `$result` y se cierra la conexión a la base de datos `$conn`.

- La función llamada `get_post` que toma la conexión a la base de datos y el nombre de una variable POST como argumentos. Su trabajo es crítico: obtener la entrada del navegador. Esta función utiliza el método `real_escape_string` de la conexión para escapar y evitar posibles inyecciones de SQL en el valor de la variable POST.

En resumen, este código muestra un formulario para añadir nuevas entradas en una base de datos MySQL, muestra las entradas existentes en la base de datos con la opción de borrarlas y realiza las operaciones correspondientes en la base de datos utilizando consultas SQL.

El programa se mostraría en un navegador como aparece en la siguiente imagen (la imagen está recortada, pero se mostrarían todas las entradas que contiene la base de datos del mismo modo que se muestran las dos primeras):

Autor
Título
Categoría
Año
ISBN

AÑADIR ENTRADA

```
  Author Willian Shakespeare
   Title Romeo y Julieta
Category Teatro
    Year 1856
    ISBN 9780123456588
```

BORRAR ENTRADA

```
  Author Charles Darwin
   Title El origen de las especies
Category No Ficción
    Year 1856
    ISBN 9780123456589
```

**La matriz $_POST.**

Un navegador envía entradas de usuario a través de un archivo GET o una petición POST. La solicitud POST suele ser la preferida (porque evita colocar datos antiestéticos en la barra de direcciones del navegador). El servidor web agrupa todas las entradas del usuario (incluso si el formulario se ha rellenado con cientos de campos) y los pone en una matriz llamada `$_POST`.

$_POST es una matriz asociativa. En función de si se ha configurado un formulario para utilizar el método POST o GET, ya sea el método $_POST o la matriz asociativa $_GET, se rellenará con los datos del formulario. Ambos se pueden leer exactamente de la misma manera.

$_POST es una superglobal, una matriz asociativa como hemos dicho, que contiene datos enviados al script PHP a través de una solicitud HTTP POST. La superglobal $_POST se crea automáticamente por PHP y está disponible en todo el script sin necesidad de declararla o inicializarla.

Cuando un formulario HTML utiliza el método POST para enviar datos, los valores de los campos de formulario se envían al script PHP y se almacenan en la superglobal $_POST. Cada campo de formulario se trata como una clave en la matriz $_POST, y el valor correspondiente es el valor ingresado por el usuario en ese campo.

Por ejemplo, si tenemos un campo de formulario llamado "nombre" en un formulario HTML, puedes acceder al valor ingresado por el usuario de la siguiente manera:

```
$nombre = $_POST['nombre'];
```

Aquí, `$nombre` contendrá el valor que el usuario ingresó en el campo de formulario con el nombre "nombre".

Es importante tener en cuenta que los datos en la superglobal $_POST son tratados como cadenas de texto. Si se esperan valores numéricos u otros tipos de datos, puede ser necesario realizar la conversión de tipo apropiada.

La superglobal $_POST solo está disponible si se envía una solicitud HTTP POST al script PHP. Si se utiliza el método GET para enviar los datos del formulario,

los valores se encontrarán en la superglobal `$_GET`. Además, tanto $_POST como $_GET pueden contener valores si los datos se envían mediante el método POST y se agregan a la URL (por ejemplo, a través de un formulario con `'method="get"'`).

En resumen, $_POST es una matriz asociativa que almacena los datos enviados mediante una solicitud HTTP POST, y permite acceder a esos datos en el script PHP utilizando los nombres de los campos de formulario como claves en la matriz.

En el script que hemos presentado, antes de verificar si se han enviado nuevos datos, el programa comprueba si la variable `$_POST['delete']` tiene un valor. Si es así, el usuario ha hecho clic en el botón de borrado para borrar un registro. Si $_POST['delete'] no se ha establecido (no hay ningún registro que borrar), se verifican $_POST['author'] y otros valores enviados. Si ha todos se les ha dado un valor, $query se configura con un comando `INSERT INTO`, seguido de los cinco valores a insertar.

Después de la visualización de cada registro, hay un segundo formulario que también se envía a index.php (el programa mismo), pero esta vez contiene dos campos ocultos: delete e isbn. El campo delete está configurado a yes e isbn al valor de la fila $row[4], que contiene el campo ISBN del registro.

**MySQL práctico.**

**Creación de una tabla.**

```php
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  $query = "CREATE TABLE cats (
    id SMALLINT NOT NULL AUTO_INCREMENT,
    family VARCHAR(32) NOT NULL,
    name VARCHAR(32) NOT NULL,
    age TINYINT NOT NULL,
    PRIMARY KEY (id)
  )";

  $result = $conn->query($query);
  if (!$result) die ("Acceso a la base de datos fallido");
?>
```

**Adicción de datos.**

```php
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  $query  = "INSERT INTO cats VALUES(NULL, 'Lion', 'Leo', 4)";
  $result = $conn->query($query);
  if (!$result) die ("Acceso a la base de datos fallido");
?>
```

El valor `NULL` pasado como primer parámetro se debe a que la columna `id` es del tipo `AUTO_INCREMENT` y MySQL decide qué valor asignar. El valor NULL será ignorado y MySQL hará su trabajo.

Por supuesto, la manera más eficiente de rellenar una tabla con datos es crear una matriz e insertar los datos con una sola consulta.

**Actualización de datos.**

```php
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  $query  = "UPDATE cats SET name='Lu' WHERE name='Leo'";
  $result = $conn->query($query);
  if (!$result) die ("Acceso a la base de datos fallido");
?>
```

**Borrado de datos.**

```php
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  $query  = "DELETE FROM cats WHERE name='Lu'";
  $result = $conn->query($query);
  if (!$result) die ("Acceso a la base de datos fallido");
?>
```

**PREVENCIÓN DE INTENTOS DE HACKING.**

**Prevención de intentos de inyección SQL.**

Es peligroso pasar las entradas del usuario sin comprobar las Consultas SQL generadas por nuestro código con los datos introducidos por el usuario. Por ejemplo supongamos este illo código para verificar a un usuario:

```
$user = $_POST['user'];
$pass = $_POST['password'];
$query = "SELECT * FROM users WHERE user='$user' AND password='$pass'";
```

El usuario podría introducir valores esperados y todo sería normal, por ejemplo generando la consulta:

```
SELECT * FROM users WHERE user='Salvarado' AND password='qwerty'
```

Pero imaginemos que alguien introduce lo siguiente para $user y no introduce nada para $pass:

```
admin' #
```

Ahora la cadena de la consulta quedaría del siguiente modo:

```
SELECT * FROM users WHERE user='admin' #' AND password=''
```

Aquí se origina un problema. Se ha producido un ataque de inyección de SQL. En MySQL, el símbolo # representa el inicio de un comentario. Por lo tanto el usurio registrará el usuario `admin` (asumiendo que existe un usuario admin) sin tener que introducir una contraseña. A continuación se muestra en negrita la consulta que se ejecutará, el restos e ignorará:

**`SELECT * FROM users WHERE user='admin'`** `#' AND password=''`

Además, podría inyectarse otro tipo de código en las consultas. Supongamos este código:

```
$user = $_POST['user'];
$pass = $_POST['password'];
$query = "DELETE FROM users WHERE user='$user' AND password='$pass'";
```

Supongamos que en un ataque de inyección SQL se introduce lo siguiente para $user:

```
algo' OR 1=1 #
```

Esto se interpretaría de la siguiente manera:

**`DELETE FROM users WHERE user='algo' OR 1=1`** `# ' AND password='$pass'`

Esta consulta siempre será `TRUE`, y por lo tanto provocará la pérdida de toda la base de datos de usuario.

Estos inconvenientes pueden ser solucionados. Veamos cómo.

En primer lugar no se debe confiar en las comillas mágicas integradas de PHP, que escapan automáticamente cualquier carácter como comillas simples o dobles, precediéndolas con una barra invertida (\). No se debe confiar dado que esta característica se puede desactivar.

En segundo lugar, siempre se debe utilizar el método `real_escape_srting` para todas las llamadas a MySQL. La siguiente función elimina cualesquiera comillas mágicas añadidas a una cadena introducida por el usuario, y a continuación la desinfecta correctamente:

```
<?php
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Cuando hablamos de "desinfectar" una cadena en el contexto de PHP, nos referimos a aplicar un conjunto de técnicas o funciones para garantizar que la cadena de texto esté libre de caracteres o secuencias que puedan causar problemas o vulnerabilidades en la seguridad de una aplicación.

La desinfección de cadenas es especialmente importante cuando se trabaja con datos de entrada proporcionados por usuarios o fuentes externas, como formularios web, archivos CSV, parámetros de URL, entre otros. Los datos de entrada no confiables pueden contener caracteres especiales, secuencias de escape, etiquetas HTML o JavaScript maliciosas, que podrían explotarse para inyecciones de código, ataques de XSS (Cross-Site Scripting) u otras vulnerabilidades.

En el código anterior, la función `get_magic_quotes_gpc` devuelve `TRUE` si las comillas mágicas están activas. En este caso, hay que eliminar cualquier barra oblicua que se haya añadido a una cadena, o el método `real_escape_string` podría termianr con un doble escape de algunos caracteres y crear cadenas dañadas.

El siguiente ejemplo muestra como incorporar la fucnión anterior `mysql_fix_string` a nuestro código:

```
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  $user  = mysql_fix_string($conn, $_POST['user']);
  $pass  = mysql_fix_string($conn, $_POST['pass']);
  $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

  // Etc...

  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Sin embargo, estas precauciones son cada vez menos importantes. Existe una manera mucho más fácil y segura de acceder a MySQL desde PHP. El uso de marcadores de posición.

**Marcadores de posición.**

La mejor y más recomendada manera de interactuar con MySQL es el uso de marcadores de posición. No es recomendable y no debe hacer la inserción directa

de datos en MySQL (aunque era importante conocer el concepto), siempre, en su lugar de deben usar los marcadores de posición.

Los marcadores de posición son posiciones dentro de declaraciones preparadas, en las que los datos se transfieren directamente a la base de datos, sin posibilidad de que el usuario (u otros) envía datos que se interpreten como declaraciones MySQL (y la posibilidad de hacking a la que podría dar lugar). Los marcadores de posición se utilizan en consultas preparadas o sentencias parametrizadas. En lugar de incluir los valores directamente en la consulta SQL, se utilizan marcadores de posición como marcadores de sustitución. Estos marcadores se representan por un signo de interrogación (?) o un nombre simbólico (por ejemplo, :nombre) dentro de la consulta.

Marcadores de posición en MySQL.

```
PREPARE statement FROM "INSERT INTO classics VALUES(?,?,?,?,?)";

SET    @author = "Emily Brontë",
       @title = "Wuthering Heights",
       @category = "Classic Fiction",
       @year = "1847",
       @isbn = "9780553212587";

EXECUTE statement USING @author,@title,@category,@year,@isbn;
DEALLOCATE PREPARE statement;
```

Puede resultar engorroso enviar las declaraciones anteriores a MySQL desde PHP, por lo que la extensión `mysqli` hace la gestión de marcadores de posición sea más fácil haciendo uso de un método ya preparado llamado `prepare()`. Al método `prepare()` se le llama del siguiente modo:

```
$stmt = $conn->prepare('INSERT INTO classics VALUES(?, ?, ?, ?, ?)');
```

El objeto `$stmt` devuelto por este método se utiliza para enviar los datos al servidor en lugar de los signos de interrogación:

```
$stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);
```

El primer argumento para `bind_param` es una cadena que representa a su vez el tipo de cada uno de los argumentos. En este caso, se compone de cinco caracteres `s`, que representan cadenas de caracteres, pero aquí se puede especificar cualquier combinación de tipos, de entre los siguientes:

```
i     int
d     double
s     string
b     BLOB
```

Veamos un ejemplo del uso de marcadores de posición en un código real:

```
<?php
  require_once 'config.inc.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Error fatal");

  $stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
  $stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);

  $author   = 'Emily Brontë';
  $title    = 'Cumbres borrascosas';
```

```
  $category = 'Ficción';
  $year     = '1847';
  $isbn     = '9780553212587';

  $stmt->execute();
  printf("%d Fila insertada.\n", $stmt->affected_rows);
  $stmt->close();
  $conn->close();
?>
```

Hemos visto un uso basado en el modelo orientado a objetos de la extensión mysqli. Pero también puede usarse en un modelo por procedimientos.

**GESTIÓN DE FORMULARIOS.**

**Creación de formularios.**

Para crear un formulario, se debe contar al menos con los siguientes elementos:

- Una etiqueta de apertura <form> y otra de cierre </form>.
- Un tipo de presentación que especifique uno de los dos métodos GET o POST.
- Uno o más campos input (de entrada).
- El URL de destino a la que deben enviarse los datos del formulario.

```
<?php
echo <<<_END
  <html>
    <head>
      <title>Test de formulario</title>
    </head>
    <body>
    <form method="post" action="formtest.php">
      Introduzca su nombre:
      <input type="text" name="name">
      <input type="submit">
    </form>
    </body>
  </html>
_END;
?>
```

El constructor echo <<<_END..._END se utiliza siempre que de deba generar código HTML de varias líneas. El formulario está configurado para enviar los datos que contiene, mediante el método POST, al programa PHP formtest.php.

Nota: Es importante tener en cuenta que los datos enviados a través del método POST no se muestran en la URL del navegador, lo que los hace más seguros en comparación con el método GET. Además, PHP tiene un límite de tamaño de datos que se pueden enviar mediante POST, que generalmente está configurado en la configuración del servidor y puede variar.

El código anterior es básicamente HTML, ahora es el momento de añadir código PHP para procesar los datos enviados por el formulario.

```
<?php
  if (isset($_POST['name'])) $name = $_POST['name'];
  else $name = "(Not entered)";

  echo <<<_END
  <html>
    <head>
      <title>Test de formulario</title>
    </head>
    <body>
    Su nombre es: $name<br>
    <form method="post" action="formtest.php">
      Introduzca su nombre:
      <input type="text" name="name">
      <input type="submit">
    </form>
    </body>
  </html>
_END;
?>
```

**Valores por defecto.**

```
<form method="post" action="calc.php"><pre>
      Total del préstamo:     <input type="text" name="principle">
      Cuota mensual:          <input type="text" name="monthly">
      Número de años:         <input type="text" name="years" value="25">
      Interés:                <input type="text" name="rate"  value="6">
                              <input type="submit">
</pre></form>
```

El formulario anterior contiene dos valores por defecto. Pero los valores por defecto también se utilizan para campos ocultos, por ejmplo si se desea pasar información adicional desde la página web al programa, además de los datos que introducen los usuarios.

**Tipos de entradas.**

Cuadros de texto.

Es un cuadro de una solo línea de texto. El formato general de una estrada de cuadro de texto es el siguiente:

```
<input type="text" name="nombre" size="tamaño" maxlength="longitud"
   value="valor">
```

`size` es el ancho del cuadro y `maxlength` especifica el núemro máximo de caracteres que el usuario puedo introducir en el campo.

Áreas de texto.

Entradas de más de una línea de texto. Su formato general es el siguiente:

```
<textarea name="nombre" cols="ancho" rows="alto" wrap="tipo"
Esto es un texto por defecto.
</textarea>
```

`textarea` tiene su propia etiqueta, no es un subtipo de la etiqueta input. `wrap` es el tipo de empaquetado, y existe tres tipos:

`Off`

El texto no tiene empaquetado y las líneas aparecen exactamente como las ha tecleado el usuario.

`Soft`

El texto va empaqueta, pero se envía al servidor como una cadena larga sin retorno de carro ni avance de línea.

`Hard`

El texto va empaquetado y se envía al servidor con retorno de carro y avance de línea.

Es importante tener en cuenta que el atributo wrap es un atributo obsoleto en HTML5, y se recomienda utilizar estilos CSS para controlar el ajuste y el comportamiento del texto en lugar de depender del atributo wrap.

Casilla de verificación.

Ejemplo:

```
Vainilla    <input type="checkbox" name="ice[]" value="Vainilla">
Chocolate   <input type="checkbox" name="ice[]" value="Chocolate">
Cereza      <input type="checkbox" name="ice[]" value="Cereza">
```

Esta resultado se enviará como una matriz, por ejmplo, `$ice[0] = Vainilla` y se podrá acceder a él mediante la matriz $_POST del siguiente modo:

```
$ice = $_POST['ice']
```

El código PHP para mostrar su contenido es bastante sencillo y podría verse así:

```
foreach($ice as $item) echo "$item<br>";
```

Botones de selección.

Ejemplo:

```
8am-12am    <input type="radio" name="hora" value="1">
12am-4pm    <input type="radio" name="hora" value="2" checked="checked">
4pm-8pm     <input type="radio" name="hora" value="3">
```

**Campos ocultos.**

Los campos ocultos son elementos <input> que se utilizan para almacenar datos que no se muestran directamente al usuario en la interfaz del formulario, pero que se envían junto con el resto de los datos cuando se envía el formulario.

Los campos ocultos se crean utilizando el atributo "type" con el valor "hidden" en un elemento <input>. A diferencia de otros tipos de campos de entrada, los campos ocultos no se muestran en la interfaz del formulario. En su lugar, se utilizan para almacenar datos relevantes o información adicional que se necesita enviar al servidor junto con otros datos del formulario.

Aquí tienes un ejemplo de cómo se utiliza un campo oculto en un formulario HTML:

```
<form action="procesar.php" method="post">
  <input type="hidden" name="campo_oculto" value="valor_oculto">
  <input type="text" name="nombre" placeholder="Nombre">
  <input type="email" name="email" placeholder="Email">
  <input type="submit" value="Enviar">
</form>
```

En este ejemplo, se ha agregado un campo oculto con el nombre "campo_oculto" y el valor "valor_oculto". Este campo oculto se envía junto con los campos de texto "nombre" y "email" cuando el formulario se envía al servidor.

En el archivo "procesar.php", puedes acceder al valor del campo oculto utilizando la superglobal $_POST de PHP:

```
<?php
$valor_oculto = $_POST['campo_oculto'];
// Realiza cualquier acción necesaria con el valor oculto
?>
```

Los campos ocultos pueden ser útiles en diversas situaciones, como pasar información de seguimiento, identificadores o datos de estado relevantes para el

procesamiento del formulario en el servidor. Es importante tener en cuenta que aunque los campos ocultos no se muestran directamente al usuario, su valor se puede ver utilizando herramientas de inspección del navegador. Por lo tanto, no deben utilizarse para almacenar información sensible o confidencial.

Listas desplegables.

La etiqueta <select> permite crear un alista desplegable de opciones, que ofrece selecciones únicas o múltiples. Se ajusta a la siguiente sintaxis:

```
<select name="nombre" size="número_líneas" multiple="multiple">
```

Ejemplo:

```
<select name="vegetales" size="5" multiple="multiple">
<option value="Espinacas">Espinacas</option>
<option value="Lechuga">Lechuga</option>
<option value="Espárragos">Espárragos</option>
<option value="Endivias">Endivias</option>
<option value="Brócoli">Brócoli</option>
</select>
```

El atributo `size` es le número de líneas a mostrar.

Etiquetas.

Se puede mejorar aún la experiencia de usuario con la etiqueta `<label>`. Con ella, se puede rodear un elemento del formulario y hacerlo seleccionable al hacer clic en cualquier parte visible que se encuentre entre las etiquetas <label> de apertura y cierre.

Así, si volvemos al ejemplo anterior de la selección en un plazo de entrega, se podría permitir al usuario hacer clic en el botón de selección en sí y en el texto asociado de esta manera:

```
<label>8am-12am<input type="radio" name="hora" value="1"></label>
```

El botón de envío.

Ejemplos:

```
<input type="submit" value="Enviar">
<input type="image" name="submit" scr="image.gif">
```

**Desinfección de entradas.**

La desinfección de entradas en un formulario HTML se refiere al proceso de validar y limpiar los datos ingresados por el usuario antes de utilizarlos en una aplicación o almacenarlos en una base de datos. La desinfección de entradas es una práctica importante para garantizar la seguridad y la integridad de los datos recibidos de los usuarios.

Cuando los usuarios envían datos a través de un formulario, es posible que ingresen información maliciosa o no válida que pueda causar problemas de seguridad o afectar el funcionamiento de la aplicación. Al desinfectar las entradas, se realizan diferentes acciones para garantizar que los datos sean seguros y cumplan con ciertos criterios establecidos.

Aquí hay algunas técnicas comunes utilizadas en la desinfección de entradas en un formulario HTML:

1. Validación: Se verifica si los datos ingresados cumplen con los requisitos específicos, como la longitud adecuada, el formato correcto o la presencia de caracteres especiales no permitidos. Esto se puede lograr utilizando expresiones regulares u otras funciones de validación en el lenguaje de programación utilizado.

2. Escape de caracteres especiales: Los caracteres especiales, como las comillas, las etiquetas HTML o los caracteres de escape, se reemplazan con secuencias de escape adecuadas para evitar problemas de seguridad, como la inyección de código o la manipulación de la estructura HTML.

3. Eliminación de etiquetas HTML: En algunos casos, es posible que se desee eliminar completamente cualquier etiqueta HTML ingresada por los usuarios para evitar posibles ataques de tipo Cross-Site Scripting (XSS) o para evitar problemas de visualización en la presentación de los datos.

4. Normalización: Se asegura de que los datos estén en un formato coherente y consistente. Esto puede implicar eliminar espacios en blanco adicionales, convertir mayúsculas y minúsculas, o aplicar formatos específicos a los datos (por ejemplo, formatear fechas o números).

Es importante mencionar que la desinfección de entradas es una práctica complementaria a otras medidas de seguridad, como el uso de consultas parametrizadas o preparadas al interactuar con una base de datos para prevenir la inyección de SQL. Además, es esencial considerar las necesidades y los requisitos específicos de seguridad de tu aplicación, ya que la desinfección de entradas puede variar dependiendo del contexto y del tipo de datos recibidos.

En resumen, la desinfección de entradas en un formulario HTML implica validar, limpiar y procesar los datos ingresados por los usuarios para garantizar su seguridad y cumplir con los requisitos específicos de la aplicación. Esto ayuda a prevenir ataques y a mantener la integridad de los datos.

No se debe confiar en ninguna variable de las matrices $_GET o $_POST hasta que hayan sido desinfectadas. Si no se hace los usuarios pueden intentar inyectar Javascript o añadir SQL, como ya hemos visto.

Por lo tanto en lugar de usar solo este código para leer la entrada del usuario:

```
$var = $_POST['user_input']
```

Podríamos usar este otro:

```
$var = $_POST['user_input']
$var = htmlentities($var);
```

Existen métodos complementarios y variados para desinfectar las entradas del usuario. Hemos visto algunos para desinfectar inyecciones de SQL.

La función htmlentities().

La función `htmlentities()` en PHP es una función que se utiliza para convertir caracteres especiales y entidades HTML en sus equivalentes de entidad HTML. Esta función es especialmente útil cuando se trabaja con datos que se mostrarán en una página web y se desea evitar que ciertos caracteres especiales o etiquetas HTML sean interpretados como código y se muestren incorrectamente.

La función htmlentities() toma una cadena de texto ($string) como argumento y devuelve una nueva cadena donde los caracteres especiales y las entidades HTML se reemplazan por sus equivalentes de entidad HTML.

Un ejemplo de cómo se utiliza la función htmlentities():

```php
<?php
$texto = "Hola <b>amigo</b> & compañía";
$texto_con_entidades = htmlentities($texto);
echo $texto_con_entidades;
?>
```

La salida de este código sería:

```
Hola &lt;b&gt;amigo&lt;/b&gt; &amp; compañía
```

En este ejemplo, la función htmlentities() reemplaza los caracteres especiales y las etiquetas HTML por sus equivalentes de entidad HTML. < se convierte en <, > se convierte en >, & se convierte en &, y así sucesivamente.

La función htmlentities() también tiene algunos parámetros opcionales, como `$flags`, que permite especificar el comportamiento de la función, y `$encoding` y `$double_encode`, que permiten controlar la codificación y la codificación duplicada de las entidades.

Es importante tener en cuenta que la función htmlentities() solo convierte caracteres especiales en entidades HTML, pero no filtra ni escapa automáticamente los datos para su uso en consultas SQL o en otros contextos de seguridad. Si estás trabajando con datos que se utilizarán en consultas SQL u otros contextos sensibles, debes utilizar funciones y técnicas de seguridad apropiadas para evitar ataques de inyección de código o problemas de seguridad.

Saber las desinfecciones que necesitamos par aun programa no es sencillo. El siguiente código muestra un ejemplo de un par de funciones que reúnen algunas comprobaciones para proporcionar un buen nivel de seguridad:

```php
<?php
  function sanitizeString($var)
  {
    if (get_magic_quotes_gpc())
      $var = stripslashes($var);
    $var = strip_tags($var);
    $var = htmlentities($var);
    return $var;
  }

  function sanitizeMySQL($connection, $var)
  {
    $var = $connection->real_escape_string($var);
    $var = sanitizeString($var);
    return $var;
  }
?>
```

Si añadimos este código al final de un programa luego podremos llamar a estas funciones cuando las necesitemos:

```
$var = sanitizeString($_POST['user_input']);
o
$var = sanitizeMySQL($connection, $_POST['user_input']);
```

**Programa de ejemplo de integración de PHP y formulario HTML.**

[convert.php]

```
<?php
  $f = $c = '';

  if (isset($_POST['f'])) $f = sanitizeString($_POST['f']);
  if (isset($_POST['c'])) $c = sanitizeString($_POST['c']);

  if (is_numeric($f))
  {
    $c = intval((5 / 9) * ($f - 32));
    $out = "$f &deg;f equals $c &deg;c";
  }
  elseif(is_numeric($c))
  {
    $f = intval((9 / 5) * $c + 32);
    $out = "$c &deg;c equals $f &deg;f";
  }
  else $out = "";

  echo <<<_END
<html>
  <head>
    <title>Conversor de temperatura</title>
  </head>
  <body>
    <pre>
      Introduzca grados Fahrenheit o Celsius y pulse Convertir:

      <b>$out</b>
      <form method="post" action="convert.php">
        Fahrenheit <input type="text" name="f" size="7">
           Celsius <input type="text" name="c" size="7">
                   <input type="submit" value="Convertir">
      </form>
    </pre>
  </body>
</html>
_END;

function sanitizeString($var)
{
  if (get_magic_quotes_gpc())
    $var = stripslashes($var);
      $var = htmlentities($var);
      $var = strip_tags($var);
      return $var;
}
?>
```

Se trata de un programa simple que convierte temperaturas entre grados Fahrenheit y grados Celsius. A continuación, se proporciona una explicación paso a paso del código:

1. Primero, se declaran las variables $f y $c y se inicializan con una cadena vacía.

2. Luego, se verifica si se recibieron valores para las variables `$_POST['f']` y `$_POST['c']` a través de la solicitud POST. Si se recibieron, se asignan

a las variables `$f` y `$c` después de pasar por la función `sanitizeString()`. Esta función se utiliza para limpiar y validar los datos ingresados por el usuario.

3. A continuación, se verifica si el valor de $f es numérico utilizando la función `is_numeric()`. Si es así, se realiza la conversión de Fahrenheit a Celsius utilizando la fórmula correspondiente y se almacena en la variable $c. Luego, se genera una cadena $out que muestra el resultado de la conversión.

4. Si el valor de `$c` es numérico, se realiza la conversión de Celsius a Fahrenheit utilizando la fórmula correspondiente y se almacena en la variable $f. También se genera la cadena $out que muestra el resultado de la conversión.

5. Si ninguna de las variables `$f` o `$c` contiene un valor numérico, se asigna una cadena vacía a $out.

6. Luego, se muestra el formulario HTML que permite al usuario ingresar temperaturas en grados Fahrenheit o Celsius. La variable $out se muestra en negrita para mostrar el resultado de la conversión, si corresponde.

7. El formulario HTML tiene dos campos de entrada, uno para Fahrenheit (<input type="text" name="f" size="7">) y otro para Celsius (<input type="text" name="c" size="7">). Al hacer clic en el botón "Convertir", se envía el formulario a través del método POST y los datos se envían al mismo archivo PHP (convert.php) para su procesamiento.

8. Al final del archivo, se define la función `sanitizeString()` que se utiliza para limpiar los datos ingresados por el usuario. Esta función utiliza las funciones `stripslashes()`, `htmlentities()` y `strip_tags()` para eliminar barras invertidas, convertir caracteres especiales en entidades HTML y eliminar etiquetas HTML.