PHP
ACCESO A BASES DE DATOS

1. Búsqueda en una base de datos MySQL usando PHP

La razón para usar PHP como una interfaz para MySQL es dar formato a los resultados de las búsquedas SQL mediante una forma visible en una página web. Mientras que puedes acceder a cualquier base de datos MySQL usando el nombre de usuario y contraseña, también puedes hacerlo desde PHP. Sin embargo, en lugar de utilizar la línea de comandos de MySQL para introducir instrucciones y visualizar las salidas, se crean cadenas de consulta que se pasan a MySQL. Cuando MySQL devuelve la respuesta, tienen una estructura de datos que PHP puede reconocer en lugar de la salida con formato cuando se trabaja en la línea de comandos de MySQL. Otros comandos PHP pueden recuperar los datos y darles formato para ser visualizados en una página web.

Para empezar, aquí se utiliza el procedimiento estándar de funciones de llamada en MySQL, por motivos de sencillez. Sin embargo, las nuevas funciones mysqli orientadas a objetos (la i es sinónimo de mejora) se están convirtiendo en la forma recomendada para interactuar con MySQL desde PHP, por lo que posteriormente se mostrará cómo utilizarlas (las viejas funciones han caído en desuso y podrían ser eliminadas de PHP en algún momento).

2. Proceso

El proceso para utilizar MySQL con PHP es:

1. Conectar a MySQL.
2. Seleccione la base de datos a utilizar.
3. Construir una cadena de consulta.
4. Realizar la consulta.
5. Recuperar los resultados y almacenarlas en una página web.
6. Repetir los pasos 3 a 5 hasta que se hayan recuperado todos los datos solicitados.
7. Desconectar de MySQL.

Lo primero y lo más importante es establecer el acceso de forma segura para impedir que la gente que intente acceder en el sistema no tenga acceso a la base de datos.

2.1. Creación de Login

La mayoría de los sitios desarrollados con PHP contienen múltiples archivos de programa que requieren acceso a MySQL y por lo tanto necesitarán el nombre de usuario y la contraseña. Por lo tanto, se ha de crear un único archivo para almacenarlos y luego incluirlo donde se necesite. El siguiente ejemplo muestra un archivo de este tipo, que se llama **login.php**. Escribe el ejemplo, reemplazando los valores (tales como nombre de usuario) con los valores reales que utilices para la base de datos MySQL, y guardarlo en el directorio de desarrollo de la web. En breve se hará uso del archivo. El nombre de host localhost debería funcionar todo el tiempo que se gestione una base de datos MySQL en el sistema local.

Ejemplo 1. El archivo login.php

```
<?php // login.php
      $db_hostname = 'localhost';
      $db_database = 'publicaciones';
      $db_username = 'usuario';
      $db_password = 'password';
?>
```

Las etiquetas envolventes <?php y ?> son muy importantes para el archivo login.php, ya que significan que las líneas entre ellas solo pueden interpretarse como código PHP. Si no se incluyen y alguien llama al archivo directamente, se mostraría como texto y revelaría las claves. Pero, con las etiquetas, todo que se verá es una página en blanco. El archivo se incluirá correctamente en otros ficheros PHP. Otra ventaja es que puedes cambiar la contraseña con frecuencia deseada y habrá que actualizar solo un archivo, no importa cuántos accesos a archivos MySQL existan.

2.2. Conexión a MySQL

Una vez grabado el archivo login.php, se puede incluir en los archivos PHP que lo necesiten para acceder a la base de datos utilizando la instrucción require_once. Esto es preferible a una include, ya que generará un error fatal si no se encuentra el archivo. Y no hallar el archivo que contiene los datos de acceso es un error grave. Además, el uso de require_once en lugar de require, significa que el archivo será leído solo cuando previamente no se ha incluido, lo que impide accesos no necesarios.

Ejemplo 2. Conexión a un servidor MySQL.

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar a MySQL: " . mysql_error());
?>
```

Este ejemplo ejecuta la función PHP mysql_connect, que requiere tres parámetros: nombre de host, nombre de usuario y contraseña, de un servidor MySQL. Al abrir devuelve un identificador al servidor; de lo contrario retorna FALSE. Observa que la segunda línea utiliza una sentencia if con la función die, que muestra el texto que indiquemos y cierra PHP con un mensaje de error si $db_server no es TRUE.

2.3. Selección de una base de datos

Una vez conectado correctamente a MySQL, ahora estás listo para seleccionar la base de datos a utilizar.

Ejemplo 3. Selección de una base de datos.

```
<?php
mysql_select_db($db_database)
or die("No se puede seleccionar la base de datos:" . mysql_error ());
?>
```

El comando para seleccionar la base de datos es mysql_select_db. Se ha de pasar el nombre de la base de datos que se desea y el servidor al que se conecta. Como en el ejemplo anterior, ha sido incluida una declaración para proporcionar un mensaje de error y la explicación, la única diferencia es que no hay necesidad de mantener el valor de retorno de la función mysql_select_db, ya que simplemente devuelve TRUE o FALSE.

Se utilizó la instrucción PHP or, lo que significa que "si el comando anterior da error, ejecute el siguiente". Ten en cuenta que para que or funcione, no debe haber ningún punto y coma al final de la primera línea de código.

2.4. Establecimiento y ejecución de una consulta

El envío de una consulta a MySQL desde PHP es simple usando la función mysql_query.

Ejemplo 4. Consulta de una base de datos.

```
<?php
$query = "SELECT * FROM clasicos";
$resultado = mysql_query($query);
if (!$resultado) die("El acceso a la base de datos ha fallado: " .
mysql_error());
?>
```

En primer lugar, la variable $query establece la consulta a realizar. En este caso, se está solicitando ver todas las filas de la tabla clasicos. Ten en cuenta que, a diferencia de la línea de comandos de MySQL, no se requiere ningún punto y coma al final de la consulta, ya que la función mysql_query se utiliza para emitir una consulta completa; no se puede usar para consultas enviadas en múltiples partes. Por lo tanto, MySQL sabe que la consulta es completa y no busca un punto y coma.

Esta función devuelve un resultado que asignamos a la variable $resultado. Después de haber utilizado MySQL en la línea de comandos, se puede pensar que el contenido de $resultado podría ser el mismo que el resultado devuelto por una consulta en la línea de comandos, con líneas horizontales y verticales, y así sucesivamente. Sin embargo, este no es el caso con el resultado devuelto a PHP. En cambio, $resultado contiene un recurso que puede ser utilizado para extraer los resultados de la consulta. En caso de fallo, $resultado contiene FALSE. Así termina el ejemplo comprobando $resultado. Si es FALSE, significa que hubo un error, y se ejecuta el comando die.

2.5. Obteniendo un resultado

Una vez que tengas un recurso a partir de una función mysql_query, se puede utilizar para recuperar los datos. La forma más sencilla de hacerlo es buscar las celdas que deseas a la vez, utilizando la función mysql_result. El ejemplo combina y amplía los ejemplos anteriores en un programa que se puede escribir y ejecutar para recuperar los resultados devueltos.

Ejemplo 5. Obteniendo resultados a la vez.

```
<?php // query.php
require_once 'login.php';

$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No se puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database)
or die("No se encuentra la base de datos: " . mysql_error());

$query = "SELECT * FROM clasicos";
$resultado = mysql_query($query);
if (!$resultado) die("Ha fallado el acceso a la base de datos: " .
mysql_error());

$rows = mysql_num_rows($resultado);
for ($j = 0 ; $j < $rows ; ++$j)
{
      echo 'Autor: ' . mysql_result($resultado,$j,'autor') . '<br>';
      echo 'Título: ' . mysql_result($resultado,$j,'titulo') . '<br>';
      echo 'Categoría: ' . mysql_result($resultado,$j,'categoria') .
      '<br>';
      echo 'Año: ' . mysql_result($resultado,$j,'anio') . '<br>';
      echo 'ISBN: ' . mysql_result($resultado,$j,'isbn') . '<br>';
}
?>
```

Las últimas 10 líneas de código son nuevas, así que echemos un vistazo a ellas. Comienzan estableciendo la variable $rows el valor devuelto por una llamada a mysql_num_rows. Esta función indica el número de filas devueltas por una consulta. Sabiendo el número de filas, entramos en un bucle for que extrae cada celda de los datos de cada fila utilizando la función mysql_result. Los parámetros suministrados a esta función son los del recurso $resultado, que fue devuelto por mysql_query, el número de fila $j, y el nombre de la columna de la que extraer los datos.

Los resultados de cada llamada a mysql_query luego son incorporados dentro de la declaración echo para mostrar un campo por línea, con un salto de línea adicional entre filas.

La tabla clasicos consta de cinco filas, y de hecho, cinco filas de datos se devuelven por query.php. Pero, tal y como está, este código es en realidad extremadamente ineficiente y lento, ya que se hacen un total de 25 llamadas a la función mysql_result para recuperar todos los datos, una sola celda a la vez. Por suerte, hay una manera mucho mejor de recuperar los datos, que se están recibiendo en una sola fila a la vez, utilizando la función mysql_fetch_row.

2.6. Obtención de una fila

Es importante mostrar cómo se puede recuperar una sola celda de datos de MySQL, pero ahora vamos a ver un método mucho más eficiente. Reemplaza el ciclo for de query.php (en el ejemplo previo) con el nuevo bucle, y se encontrará que se obtendría exactamente el mismo resultado que se muestra en el ejemplo anterior.

Ejemplo 6. Reemplazo de bucle para buscar los resultados de una fila a la vez.

```
<?php
for ($j = 0 ; $j < $rows ; ++$j)
{
      $row = mysql_fetch_row($resultado);
      echo 'Autor:     ' . $row[0] . '';
      echo 'Título:    ' . $row[1] . '';
      echo 'Categoría: ' . $row[2] . '';
      echo 'Año:       ' . $row[3] . '';
      echo 'ISBN:      ' . $row[4] . '';
}
?>
```

En este código modificado, sólo una quinta parte de las llamadas se realizan a una función MySQL de llamada (el 80 % menos), debido a que cada fila se recupera en su totalidad a través de la función mysql_fetch_row. Esto devuelve una sola fila de datos en una matriz, que luego se asigna a la variable $row. Todo lo que es necesario, entonces, es hacer referencia a cada elemento de la fila de $row (los elementos comienzan a partir de 0). Por lo tanto $row[0] contiene el dato del autor, $row[1] el título, y así sucesivamente, porque cada columna se coloca en la matriz en el orden en que aparece en la tabla MySQL. Además, mediante el uso de mysql_fetch_row en lugar de mysql_result, se utiliza mucho menos código PHP y el tiempo de ejecución es menor, debido a la simple referencia a cada elemento de los datos por desplazamiento en lugar de por su nombre.

2.7. Cierre de una conexión

Cuando hayas terminado de usar una base de datos, se debe cerrar la conexión.

Ejemplo 7. Cierre de una conexión con el servidor MySQL.

```
<?php
mysql_close($db_server);
?>
```

3. Un ejemplo práctico

Es hora de escribir nuestro primer ejemplo de inserción de datos y eliminación de una tabla de MySQL usando PHP. Se trata de una base de datos de referencias bibliográficas.

3.1. Código

Ejemplo. Inserción y borrado usando sqltest.php

```
<?php // sqltest.php
require_once 'login.php';

$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database, $db_server)
or die("No puede seleccionar la base de datos: " . mysql_error());

if (isset($_POST['delete']) && isset($_POST['isbn']))
{
      $isbn = get_post('isbn');
      $query = "DELETE FROM clasicos WHERE isbn='$isbn'";
      if (!mysql_query($query, $db_server))
            echo "DELETE failed: $query" .
            mysql_error() . "

            ";
}
if (isset($_POST['autor']) &&
      isset($_POST['titulo']) &&
      isset($_POST['categoria']) &&
      isset($_POST['anio']) &&
      isset($_POST['isbn']))
      {
            $author = get_post('autor');
            $title = get_post('titulo');
            $category = get_post('categoria');
            $year = get_post('anio');
            $isbn = get_post('isbn');
            $query = "INSERT INTO clasicos VALUES" .
            "('$autor', '$titulo', '$categoria', '$anio', '$isbn')";

            if (!mysql_query($query, $db_server))
                  echo "INSERT failed: $query
                  " .
                  mysql_error() . "

                  ";
      }
echo <<<_END

<form action="sqltest.php" method="post">
      Autor <input type="text" name="autor">
      Titulo <input type="text" name="titulo">
      Categoría <input type="text" name="categoria">
      Año <input type="text" name="anio">
      ISBN <input type="text" name="isbn">
      <input type="submit" value="Añade registro">
</form>
_END;
```

```
$query = "SELECT * FROM clasicos";

$resultado = mysql_query($query);
if (!$resultado) die ("Fallo en acceso a la base de datos: " .
mysql_error());
$rows = mysql_num_rows($resultado);
for ($j = 0 ; $j < $rows ; ++$j)
{
      $row = mysql_fetch_row($resultado);
      echo <<<_END

      Autor $row[0]
      Título $row[1]
      Categoría $row[2]
      Año $row[3]
      ISBN $row[4]

      <form action="sqltest.php" method="post">
      <input type="hidden" name="delete" value="yes">
      <input type="hidden" name="isbn" value="$row[4]">
      <input type="submit" value="DELETE RECORD">
      _END;
}

mysql_close($db_server);

function get_post($var)
{
      return mysql_real_escape_string($_POST[$var]);
}
?>
```

Cómo funciona

La primera sección del código nuevo comienza con la función isset para comprobar si los valores de todos los campos se han publicado al programa. Tras la confirmación, cada una de las líneas dentro de la sentencia If llama a la función get_post, que aparece al final del programa. Esta función tiene una misión pequeña, pero crítica: ir a buscar la entrada desde el navegador.

3.2. La matriz $ _POST

Se mencionó anteriormente que un navegador envía la entrada del usuario a través de una petición GET o POST. POST normalmente se prefiere, y lo usamos aquí. El servidor web agrupa a la totalidad de la entrada del usuario (incluso si el formulario cuenta con un centenar de campos) y lo pone en una matriz llamada $_POST, que es una matriz asociativa. En función de si el formulario se ha configurado para utilizar el método POST o GET, ya sea la matriz asociativa $_POST o la $_GET, se rellenará con los datos del formulario. Ambas pueden ser leídas exactamente de la misma manera.

Cada campo se asocia a un elemento de la matriz que lleva el nombre de ese campo. Por lo tanto, si un formulario contiene un campo denominado isbn, la matriz $_POST contiene un elemento introducido por la palabra isbn. El programa PHP puede leer ese campo, haciendo referencia mediante cualquiera de estas expresiones, $_POST['isbn'] o $_POST["ISBN"] (comillas simples y dobles tienen el mismo efecto en este caso).

Si la sintaxis $_POST todavía parece compleja, se puede utilizar la convención de copiar la entrada del usuario a otras variables, y olvidarse de $_POST después de eso. Esto es normal en

los programas de PHP: se recuperan todos los campos de $_POST al inicio del programa y luego se ignoran.

Volviendo a la función get_post: pasa cada elemento que recupera a través de la función mysql_real_escape_string, que debe eliminar todos los caracteres que un hacker puede haber insertado con el fin de entrar o alterar la base de datos.

3.3. Eliminación de un registro

Antes de comprobar si los nuevos datos se han publicado, el programa comprueba si la variable $_POST['delete'] tiene un valor. Si es así, el usuario ha hecho clic en el botón Eliminar registro para borrar un registro. En este caso, también se ha publicado el valor de $isbn. El ISBN identifica de forma exclusiva cada registro. El formulario HTML anexa el ISBN a la cadena DELETE FROM query, creada en la variable $query, que a continuación, se pasa a la función mysql_query, que retorna TRUE or FALSE, lo que provoca un mensaje de error que se mostrará explicando lo que salió mal.

Si $_POST['delete']) no se establece (y por lo tanto no hay ningún registro a borrar), $_POST['autor']) y otros valores publicados se comprueban. Si a todos ellos les han sido dados valores, entonces $query se establece en un comando INSERT INTO, seguido por los cinco valores para ser insertados. Se pasa entonces la variable a mysql_query, que tras completarse genera los retornos de terminación TRUE o FALSE. Si es FALSE, se muestra un mensaje de error.

3.4. Consulta de la base de datos

A continuación, el código devuelve a un ejemplo previo, donde en las siguientes cuatro líneas de código, se envía una consulta a MySQL preguntando para ver todos los registros de la tabla clásicos. Después de eso, $rows se establece en un valor que representa el número de filas en la tabla y se introduce un ciclo for para mostrar el contenido de cada fila.

Se ha alterado el siguiente bit de código para simplificar las cosas. En lugar de utilizar las etiquetas
 para saltos de línea, se ha optado por utilizar una etiqueta <pre> para alinear la pantalla de cada registro de una manera agradable. Después de la visualización de cada registro, hay un segundo formulario que también emplea sqltest.php (el programa en sí) pero esta vez contiene dos campos ocultos: delete e isbn. El campo de borrado está ajustado a "yes" e isbn al valor contenido en $row[4], que contiene el ISBN para el registro. A continuación, se muestra un botón Submit con el texto Borra registro y se cierra el formulario. Un corchete luego completa el bucle for, que continuará hasta que se hayan mostrado todos los registros.

Por último, aparece la definición para la función get_post, que ya hemos visto. Y eso es todo, nuestro primer programa PHP para manipular una base de datos MySQL. Por lo tanto, vamos a ver lo que puede hacer.

```
Carl Sagan
El mundo y sus demonios
Divulgación científica
2005
9788408058199
```

3.5. Ejecución del programa

Cuando hayas añadido estos datos utilizando el botón *Añade registro*, desplázate hacia abajo para la parte inferior de la página, para ver la nueva adición.

4. MySQL Práctico

Ahora se describirán algunas técnicas prácticas que se pueden utilizar en PHP para acceder a la base de datos MySQL, incluyendo tareas como crear y eliminar tablas; inserción, actualización y supresión de datos; y la protección de la base de datos y el sitio web de los usuarios maliciosos. Ten en cuenta que en los siguientes ejemplos se supone que ha creado el programa login.php discutido anteriormente.

4.1. Creación de una tabla

Supongamos que estás trabajando para un parque natural y necesitas crear una base de datos para mantener los detalles acerca de todos los tipos de felinos que alberga. Te dicen que hay nueve familias de felinos -león, tigre, jaguar, leopardo, puma, guepardo, lince, caracal y doméstico- por lo que tendrás una columna para ellos. A cada felino se le ha dado un nombre, de modo que hay otra columna, y si se desea hacer un seguimiento de sus edades, que es otra columna. Por supuesto, es probable que posteriormente sean necesarias más columnas, tal vez para mantener los requisitos dietéticos, inoculaciones, y otros detalles, pero por ahora eso es suficiente para ponerse en marcha. También se necesita un identificador único para cada animal, por lo que se crea una columna llamada id.

Código que puedes utilizar para crear una tabla MySQL para mantener estos datos, con la asignación principal de consulta en negrita.

Ejemplo 9. Creación de una tabla llamada felinos.

```
<?php

require_once 'login.php';

$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puedo conectar a MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No puedo seleccionar base de datos: " . mysql_error());

$query = "CREATE TABLE felinos (
      id SMALLINT NOT NULL AUTO_INCREMENT,
      familia VARCHAR(32) NOT NULL,
      nombre VARCHAR(32) NOT NULL,
      edad TINYINT NOT NULL,
      PRIMARY KEY (id)
)";

$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
?>
```

Como puedes ver, la consulta MySQL es muy similar a cómo se debe escribir directamente en la línea de comandos.

4.2. Descripción de una tabla

Cuando no se está conectado a la línea de comandos de MySQL, he aquí código que se puede utilizar para verificar que una tabla se ha creado correctamente desde dentro de un navegador.

Simplemente emite la consulta DESCRIBE felinos y luego da salida a una tabla HTML con cuatro títulos - Columna, Tipo, Nulo, y Clave - debajo de los cuales se muestran todas las columnas

dentro de la tabla. Para usarlo con otras tablas, simplemente reemplazar el nombre felinos en la consulta con el de la nueva tabla.

Ejemplo 10. Describiendo la tabla felinos.

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No se puede conectar a MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No puede seleccionar la base de datos: " . mysql_error());
$query = "DESCRIBE felinos";
$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
$rows = mysql_num_rows($result);
echo "";
for ($j = 0 ; $j < $rows ; ++$j)
{
      $row = mysql_fetch_row($resultado);
      echo "";
      for ($k = 0 ; $k < 4 ; ++$k)
      echo "";
      echo "";
}
echo "
Columna     Tipo  Nulo  Clave
$row[$k]
";
?>
```

El programa mostraría una salida de este tipo:

```
Columna Tipo          Nulo  Clave
id      smallint (6)  NO    PRI
familia varchar  (32) NO
nombre  varchar  (32) NO
edad    tinyint  (4)  NO
```

4.3. Borrar una tabla

Borrar una tabla es fácil mediante DROP y por lo tanto es muy peligroso, así que ten cuidado pues perderías toda su información. El ejemplo siguiente muestra el código. Sin embargo, yo no recomiendo que lo pruebes hasta que sea posible con otros ejemplos, ya que eliminarías la tabla felinos y tendrías que volver a crearla.

Ejemplo 11. Borrar la tabla felinos

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No se puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No se puede seleccionar la base de datos: " .
mysql_error());

$query = "DROP TABLE felinos";
$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
?>
```

4.4. Agregar datos

Vamos a añadir algunos datos a la tabla utilizando el código en el ejemplo.

Ejemplo 12. Añadir datos a la tabla felinos.

```
<php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar a MySQL: " . mysql_error());
mysql_select_db($db_database)
or die("No puede seleccionar base de datos: " . mysql_error());

$query = "INSERT INTO felinos VALUES(NULL, 'León', 'Leo', 4)";
$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
?>
```

Es posible que desees añadir un par de elementos de datos modificando $query como sigue:

```
$query = "INSERT INTO felinos VALUES (NULL, 'Puma', 'Growler", 2) ";
$query = "INSERT INTO felinos VALUES (NULL, 'Cheetah', 'Charly', 3)";
```

El valor NULL es pasado como el primer parámetro, esto se debe a que la columna id es de tipo AUTO_INCREMENT, y MySQL decidirá qué valor asignar de acuerdo con el siguiente número disponible en orden, así que simplemente se ha de pasar un NullValue, que será ignorado.

Por supuesto, la forma más eficiente para rellenar con datos MySQL es crear una matriz e introducir los datos con una sola consulta.

4.5. Recuperación de datos.

Ahora que se han introducido algunos datos en la tabla felinos, el ejemplo muestra cómo se puede comprobar que se han insertado correctamente.

Ejemplo 13. Recuperando filas de la tabla felinos.

```
<php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No puede seleccionar base de datos: " . mysql_error());

$query = "SELECT * FROM felinos";
$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
$rows = mysql_num_rows($result);
echo "";
for ($j = 0 ; $j < $rows ; ++$j){
      $row = mysql_fetch_row($resultado);
      echo "";
      for ($k = 0 ; $k < 4 ; ++$k)
            echo "";
      echo "";
}
echo "
Id    Familia     Nombre      Edad
$row[$k]
";
?>
```

Este código simplemente emite la consulta MySQL SELECT * FROM felinos y muestra todas las filas devueltas. Su salida es la siguiente:

```
Id Felino    Nombre Edad
1  León      León      4
2  Puma      Growler   2
3  Guepardo Charly    3
```

Aquí se puede ver que la columna id se ha incrementado correctamente de forma automática.

4.6. Actualización de datos

El cambio de datos que ya se ha insertado también es bastante simple. ¿Te diste cuenta la mala ortografía de Charly para el nombre del guepardo? Vamos a corregir eso a Charlie, según el siguiente ejemplo.

Ejemplo 14. Cambiar el nombre de Charly al guepardo Charlie.

```
<?php
require_once 'login.php';

$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No puede seleccionar la base de datos: " . mysql_error());

$query = "UPDATE felinos SET nombre='Charlie' WHERE nombre='Charly'";
$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
?>
```

4.7. Eliminación de datos

Growler el puma ha sido transferido a otro zoológico, así que es el momento de sacarlo de la base de datos.

Ejemplo 15. Extracción Growler el puma de la tabla felinos.

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No puede seleccionar la base de datos: " . mysql_error());

$query = "DELETE FROM felinos WHERE nombre='Growler'";
$result = mysql_query($query);
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
?>
```

4.8. Utilización de AUTO_INCREMENT

Cuando se usa AUTO_INCREMENT, no se puede saber qué valor se le ha dado a una columna antes de insertar una fila. En cambio, si necesitas saberlo, debes preguntar a MySQL usando la función mysql_insert_id. Esta necesidad es común: por ejemplo, al procesar una compra, es posible insertar un nuevo cliente en una tabla Clientes y luego hacer referencia a la identidad del cliente recién creada, insertando una compra.

Ejemplo 16. Agregar datos a los felinos de la tabla e informar id de inserción.

```
<?php

require_once 'login.php';

$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("No puede conectar con MySQL: " . mysql_error());
mysql_select_db($db_database)
      or die("No puede seleccionar la base de datos: " . mysql_error());

$query = "INSERT INTO felinos VALUES(NULL, 'Lince', 'Stumpy', 5)";
$result = mysql_query($query);
echo "El id de inserción es: " . mysql_insert_id();
if (!$result) die("Fallo en acceso a la base de datos: " .
mysql_error());
?>
```

4.9. Uso de identificadores de inserción

Es muy común insertar datos en varias tablas: un libro seguido por su autor, o un cliente seguido de su compra, y así sucesivamente. Al hacer esto con una columna de incremento automático, tendrás que retener la Identificación de inserción retornada, para almacenar en la tabla relacionada.

Por ejemplo, vamos a suponer que estos felinos pueden ser "adoptados" por el público como un medio para recaudar fondos, y que cuando un gato nuevo se almacena en la tabla felinos, también se ha de crear una clave para relacionar con el propietario adoptivo. El código para hacer esto es similar al del ejemplo 16, excepto que el ID de inserción devuelto se almacena en la variable $insertID, y se utiliza entonces como parte de la consulta posterior:

```
$consulta = "INSERT INTO felinos VALUES (NULL, 'Lince', 'achaparrado',
5)";
$Resultado = mysql_query ($consulta);
$InsertID = mysql_insert_id ();
$consulta = "INSERT INTO propietarios VALUES ($insertID, 'Pepe',
'Perea')";
$Resultado = mysql_query ($consulta);
```

Ahora el gato está conectado a su "dueño" a través del identificador único del gato, que se creó automáticamente por AUTO_INCREMENT.

4.10. Uso de bloqueos

Un procedimiento completamente seguro para vincular tablas a través de la de inserción ID es el uso de bloqueos. Se puede ralentizar el tiempo de respuesta un poco cuando hay muchas personas que añadan datos a la misma tabla, pero también puede valer la pena. La secuencia es:

1. Bloquear la primera tabla (por ejemplo, felinos).
2. Insertar datos en la primera tabla.
3. Recuperar el identificador único de la primera tabla a través de mysql_insert_id.
4. Desbloquear la primera tabla.
5. Insertar datos en la segunda tabla.

Se puede liberar de forma segura el bloqueo antes de insertar datos en la segunda tabla, debido a que el ID de inserción se ha recuperado y se almacena en una variable de programa. También puede utilizar una transacción en lugar de bloquear, aunque ralentiza el servidor MySQL aún más.

5. Elegir una API

PHP ofrece tres API diferentes para conectarse a MySQL. Abajo se muestran las API proporcionadas por las extensiones mysql, mysqli, y PDO. Cada trozo de código crea una conexión al servidor de MySQL que se está ejecutando en "ejemplo.com" con el nombre de usuario "usuario" y la contraseña "contraseña". También se ejecuta una consulta para saludar al usuario.

Ejemplo #1 Comparación de las tres API de MySQL

```php
<?php

// mysqli

$mysqli = new mysqli("ejemplo.com", "usuario", "contraseña", "basedatos");

$resultado = $mysqli->query("SELECT '¡Hola, querido usuario de MySQL!' AS _message FROM DUAL");

$fila = $resultado->fetch_assoc();

echo htmlentities($fila['_message']);

// PDO

$pdo = new PDO('mysql:host=ejemplo.com;dbname=basedatos', 'usuario', 'contraseña');

$sentencia = $pdo->query("SELECT '¡Hola, querido usuario de MySQL!' AS _message FROM DUAL");

$fila = $sentencia->fetch(PDO::FETCH_ASSOC);

echo htmlentities($fila['_message']);

// mysql

$c = mysql_connect("ejemplo.com", "usuario", "contraseña");

mysql_select_db("basedatos");

$resultado = mysql_query("SELECT '¡Hola, querido usuario de MySQL!' AS _message FROM DUAL");

$fila = mysql_fetch_assoc($resultado);

echo htmlentities($fila['_message']);

?>
```

5.1. API recomendada

Se recomienda usar las extensiones mysqli o PDO_MySQL. No se recomienda usar la extensión mysql antigua para nuevos desarrollos, ya que ha sido declarada obsoleta en PHP 5.5.0 y eliminada en PHP 7. Se proporciona una matriz detallada de comparación de características más abajo. El rendimiento global de las tres extensiones se considera que sea aproximadamente el mismo. Aunque el rendimiento de la extensión aporta solamente una fracción del total del tiempo de ejecución de una consulta web de PHP. A menudo, el impacto es tan bajo como 0.1%.

Comparación de características

	ext/mysqli	PDO_MySQL	ext/mysql
Versión de PHP donde se introdujo	5.0	5.1	2.0
Incluida con PHP 5.x	Sí	Sí	Sí
Incluida con PHP 7.x	Sí	Sí	No
Estado de desarrollo	Activo	Activo	Mantenimiento solamente en 5.x; eliminada en 7.x
Ciclo de vida	Activo	Activo	Obsoleto en 5.x; eliminado en 7.x
Recomendada para nuevos proyectos	Sí	Sí	No
Interfaz de POO	Sí	Sí	No
Interfaz procedimental	Sí	No	Sí
La API admite la no espera, consultas asíncronas con mysqlnd	Sí	No	No
Conexiones persistentes	Sí	Sí	Sí
La API admite conjunto de caracteres	Sí	Sí	Sí
La API admite sentencias preparadas del lado del servidor	Sí	Sí	No
La API admite sentencias preparadas del lado del cliente	No	Sí	No
La API admite procedimientos almacenados	Sí	Sí	No
API admite sentencias múltiples	Sí	La mayoría	No
La API admite transacciones	Sí	Sí	No
Las transacciones se pueden controlar con SQL	Sí	Sí	Sí
Admite toda la funcionalidad de MySQL 5.1+	Sí	La mayoría	No

5.2. Desarrollo de PDO con MYSQL

Las conexiones se establecen creando instancias de la clase base PDO. No importa el controlador que se utilice; siempre se usará el nombre de la clase PDO. El constructor acepta parámetros para especificar el origen de la base de datos (conocido como DSN) y, opcionalmente, el nombre de usuario y la contraseña (si la hubiera).

5.3. Conectarse a MySQL

```
<?php

$mbd = new PDO('mysql:host=localhost;dbname=prueba',
```

```
                $usuario,
                $contraseña);
?>
```

5.4. Manejo de errores de conexión

Si hubiera errores de conexión, se lanzará un objeto *PDOException*. Se puede capturar la excepción si fuera necesario manejar la condición del error, o se podría optar por dejarla en manos de un manejador de excepciones global de aplicación configurado mediante set_exception_handler().

```
<?php
try{
      smbd = new PDO('mysql:host=localhost;dbname=prueba',
                     $usuario,
                     $contraseña);
      foreach($mbd->query('SELECT * from FOO') as $fila) {
            print_r($fila);
      }
      smbd = null;
} catch (PDOException $e){
      print "¡Error!: " . $e->getMessage() . "<br/>";
      die();
}
?>
```

5.5. Advertencia

Si la aplicación no captura la excepción lanzada por el constructor de PDO, la acción predeterminada que toma el motor zend es la de finalizar el script y mostrar información de seguimiento. Esta información probablemente revelará todos los detalles de la conexión a la base de datos, incluyendo el nombre de usuario y la contraseña. Es su responsabilidad capturar esta excepción, ya sea explícitamente (con una sentencia *catch*) o implícitamente por medio de set_exception_handler().

5.6. Cerrar una conexión

Una vez realizada con éxito una conexión a la base de datos, será devuelta una instancia de la clase PDO al script. La conexión permanecerá activa durante el tiempo de vida del objeto PDO. Para cerrar la conexión, es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas; esto se puede hacer asignando **NULL** a la variable que contiene el objeto. Si no se realiza explícitamente, PHP cerrará automáticamente la conexión cuando el script finalice.

Nota: Si aún existen otras referencias a esta instancia de PDO (tales como desde una instancia de PDOStatement, o desde otras variables que hacen referencia a la misma instancia de PDO), estas también han de eliminarse (por ejemplo, asignando NULL a la variable que hace referencia al PDOStatement).

```
<?php
$mbd = new PDO('mysql:host=localhost;dbname=prueba', $usuario,
$contraseña);
// Utilizar la conexión aquí
$sth = $mbd->query('SELECT * FROM foo');
// Ya se ha terminado; se cierra
$sth = null;
$mbd = null;
?>
```

5.7. Conexiones persistentes

Muchas aplicaciones web se beneficiarán del uso de **conexiones persistentes** a servidores de bases de datos. Las conexiones persistentes no son cerradas al final del script, sino que son almacenadas en caché y reutilizadas cuando otro script solicite una conexión que use las mismas credenciales. La caché de conexiones persistentes permite evitar la carga adicional de establecer una nueva conexión cada vez que un script necesite comunicarse con la base de datos, dando como resultado una aplicación web más rápida.

```
<?php

$mbd = new PDO('mysql:host=localhost;dbname=prueba', $usuario, $contraseña, array(

PDO::ATTR_PERSISTENT => true

));

?>
```

Nota:

Para utilizar conexiones persistentes, se deberá establecer `PDO::ATTR_PERSISTENT` en las opciones del array del controlador pasado al constructor de PDO. Si este atributo se establece con PDO::setAttribute() después de la instanciación del objeto, el controlador no utilizará conexiones persistentes.

Nota:

Si se usa el controlador PDO y las bibliotecas ODBC admiten el aprovisionamiento de conexiones ODBC (unixODBC y Windows lo hacen; podrían haber más), se recomienda no utilizar las conexiones persistentes de PDO, y, en su lugar, dejar el almacenamiento en caché de conexiones a la capa del aprovisionamiento de conexiones de ODBC. La provisión de conexiones de ODBC se comparte con otros módulos en el proceso; si se le indica a PDO que almacene en caché la conexión, entonces dicha conexión nunca será devuelta a la provisión de conexiones de ODBC, dando como resultado la creación de conexiones adicionales para servir a los demás módulos.