CAPÍTULO 10

Acceso a MySQL mediante PHP

Si has asimilado lo expuesto en los capítulos anteriores, eres competente en el uso tanto de MySQL como de PHP. En este capítulo, aprenderás cómo integrar los dos con las funciones integradas de PHP para acceder a MySQL.

Consultas de la bases de datos MySQL con PHP

La razón de usar PHP como interfaz para MySQL es para formatear los resultados de las consultas SQL en un formulario visible en una página web. Siempre y cuando puedas acceder a tu MySQL mediante tu nombre de usuario y contraseña, también puedes hacerlo desde PHP.

Sin embargo, en lugar de usar la línea de comandos de MySQL para introducir instrucciones y ver la salida, vamos a crear cadenas de consultas que se pasarán a MySQL. Cuando MySQL devuelve su respuesta vendrá como una estructura de datos que PHP puede reconocer en lugar del formato que se ve cuando se trabaja en la línea de comandos. Otros comandos PHP pueden recuperar los datos y formatearlos para la página web.

El proceso

El proceso de uso de MySQL con PHP es el siguiente:

1. Conectarse a MySQL y seleccionar la base de datos a utilizar.
2. Preparar una cadena de consulta.
3. Realizar la consulta.
4. Recuperar los resultados y enviarlos a una página web.
5. Repetir los pasos 2 a 4 hasta que se hayan recuperado todos los datos deseados.
6. Desconectarse de MySQL.

Nosotros seguiremos estos pasos, pero primero es importante que configures tus datos de inicio de sesión de forma segura para que la gente que husmee en tu sistema tenga problemas para acceder a tu base de datos.

Creación del archivo de inicio de sesión

La mayoría de los sitios web desarrollados con PHP contienen múltiples archivos de programa que requerirán acceso a MySQL y, por consiguiente, necesitarás los datos de inicio de sesión y contraseña. Por lo tanto, es sensato crear un único archivo para almacenarlas e incluirlas dondequiera que se necesiten. En el Ejemplo 10-1 se muestra un archivo de este tipo, al que he llamado *login.php*.

Ejemplo 10-1. El archivo login.php

```
<?php // login.php
  $hn = 'localhost';
  $db = 'publications';
  $un = 'username';
  $pw = 'password';
?>
```

Escribe el ejemplo, sustituy `username` y `password` por los valores que utilizas para tu MySQL, y guárdalo en la carpeta principal que configuraste en el Capítulo 2. Haremos uso del archivo en breve.

El nombre de host `localhost` debería funcionar siempre y cuando estés utilizando una base de datos MySQL en tu sistema local, y la base de datos `publications` debería funcionar si utilizas los ejemplos que he expuesto hasta ahora.

Las etiquetas inclusivas `<?php` y `?>` son especialmente importantes en el archivo *login.php* en el Ejemplo 10-1, porque significa que las líneas entre ellas *solo* se pueden interpretar como código PHP. Si las omitieras y alguien hiciera una llamada al archivo directamente desde tu sitio web, se mostraría como texto y revelaría tus secretos. Pero, con las etiquetas en su sitio, todo lo que esa persona verá es una página en blanco. El archivo incluirá correctamente tus otros archivos PHP.

La variable `$hn` le dirá a PHP qué ordenador usar para conectarse a una base de datos. Esto es necesario porque puedes acceder a bases de datos MySQL desde cualquier ordenador conectado a tu instalación de PHP, que potencialmente incluye cualquier host en cualquier lugar de la web. Sin embargo, los ejemplos de este capítulo funcionarán en el servidor local. Así que, en lugar de especificar un dominio como `mysql.myserver.com`, puedes usar la palabra `localhost` o la dirección IP `127.0.0.1`).

La base de datos que usaremos, `$db`, es la que llamamos *publications* que creamos en el Capítulo 8. Si usas una base de datos diferente, (una proporcionada por el administrador de tu servidor), tendrás que modificar *login.php* en consecuencia.

Otra ventaja de mantener estos datos de inicio de sesión en un solo lugar es que puedes cambiar tu contraseña con la frecuencia que desees y solo habrá un archivo para actualizar cuando lo hagas, sin importar cuántos archivos PHP tengan acceso a MySQL.

Conexión a la base de datos MySQL

Ahora que has guardado el archivo *login.php*, puedes incluirlo en cualquier archivo PHP que necesitará acceder a la base de datos mediante la declaración `require_once`. Esta es preferible a una declaración `include`, que generará un error fatal si no se encuentra el archivo, y, créeme, no encontrar el archivo que contiene los datos de acceso a tu base de datos es un error fatal.

Además, usar `require_once` en lugar de `require` significa que el archivo solo se leerá cuando no se haya incluido previamente, lo que evita el desperdicio de accesos duplicados al disco. El Ejemplo 10-2 muestra el código a utilizar.

Ejemplo 10-2. Conexión a un servidor MySQL con mysqli

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

Este ejemplo crea un nuevo objeto denominado `$conn` al llamar a una nueva instancia del método `mysqli` y pasa todos los valores recuperados del archivo *login.php*. Logramos la comprobación de errores mediante la referencia a la propiedad `$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.

El objeto `$conn` se utiliza en los siguientes ejemplos para acceder a la base de datos MySQL.

La función `die` es genial cuando desarrollas código PHP, pero por supuesto querrás tener mensajes de error más fáciles de usar en un servidor de producción. En este caso, no abortarás tu programa PHP, sino que formatearás un mensaje que se mostrará cuando el programa finalice normalmente, tal vez algo como esto:

```
function mysql_fatal_error()
{
    echo <<< _END
We are sorry, but it was not possible to complete
the requested task. The error message we got was:

    <p>Fatal Error</p>

Please click the back button on your browser
and try again. If you are still having problems,
please <a href="mailto:admin@server.com">email
our administrator</a>. Thank you.
_END;
}
```

Tampoco debes tener la tentación de dar salida al contenido de ningún mensaje de error recibido de MySQL. En lugar de ayudar a tus usuarios, podrías regalar información confidencial a los hackers, como los datos de inicio de sesión. En lugar de esto, simplemente guía al usuario con información sobre cómo superar su dificultad basándose en el mensaje de error que se reporta a tu código.

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. En el Ejemplo 10-3 se muestra cómo hacerlo.

Ejemplo 10-3. Consulta de una base de datos con mysqli

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

Como puedes ver, la consulta MySQL se parece a la que escribirías directamente en la línea de comandos, excepto que no hay punto y coma final, ya que no se necesita cuando accedes a MySQL desde PHP.

Aquí a la variable `$query` se le asigna una cadena que contiene la consulta a realizar, y luego se pasa al método `query` (consulta) del objeto `$conn`, que devuelve un resultado que colocamos en el objeto `$result`. Si `$result` es `FALSE`, ha habido un problema y

la propiedad `error` del objeto de conexión contendrá los detalles, por lo que se llama a la función `die` para visualizar ese error.

Todos los datos devueltos por MySQL se almacenan ahora en un formato fácilmente interrogable en el objeto `$result`.

Obtención del resultado

Una vez que tengas el resultado devuelto en el objeto en `$result`, puedes usarlo para extraer los datos que quieras, un elemento tras otro, mediante el método del objeto `fetch_assoc`. El Ejemplo 10-4 combina y amplía los ejemplos anteriores en un programa que puedes ejecutar para recuperar los resultados (como se muestra en la Figura 10-1. Escribe este script y guárdalo con el nombre de archivo *query-mysqli.php*, o descárgalo de la página web complementaria (*www.marcombo.info*).

Ejemplo 10-4. Obtención de resultados celda por celda

```
<?php // query-mysqli.php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  $query = "SELECT * FROM classics";
  $result = $connection->query($query);

  if (!$result) die("Fatal Error");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
   $result->data_seek($j);
   echo 'Author: ' .htmlspecialchars($result-
>fetch_assoc()['author']) .'<br>';
   $result->data_seek($j);
   echo 'Title:  ' .htmlspecialchars($result-
>fetch_assoc()['title'])  .'<br>';
   $result->data_seek($j);
   echo 'Category: ' .htmlspecialchars($result-
>fetch_assoc()['category']) .'<br>';
   $result->data_seek($j);
   echo 'Year:   '. htmlspecialchars($result->fetch_assoc()['year'])
.'<br>';
   $result->data_seek($j);
   echo 'ISBN:   ' .htmlspecialchars($result->fetch_assoc()['isbn'])
.'<br><br>';
  }
```

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

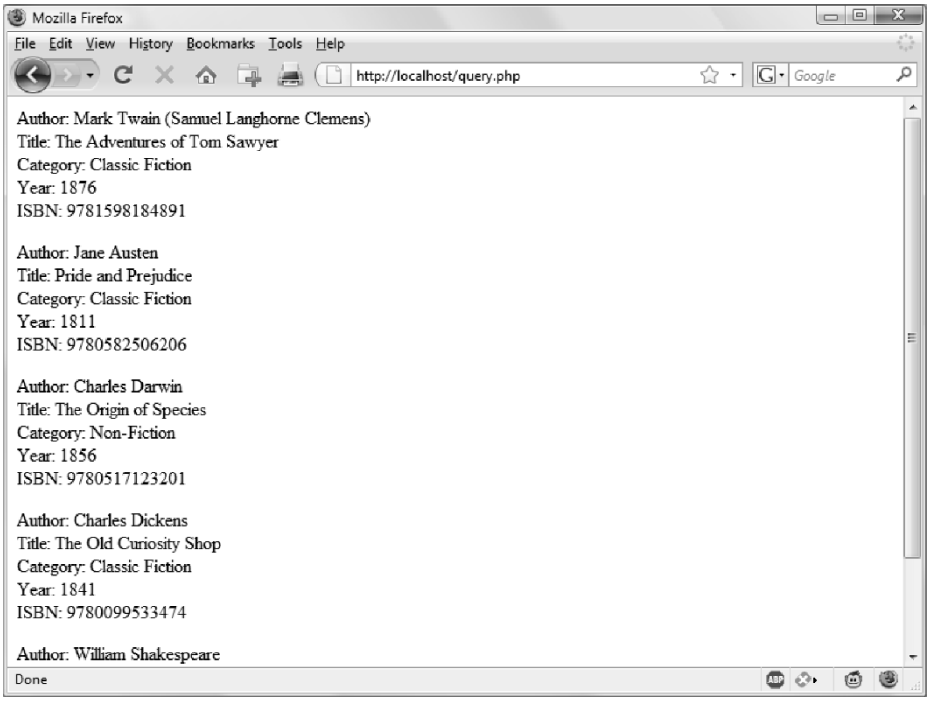

Figura 10-1. Salida del programa query-mysqli.php en el Ejemplo 10-4

Aquí, cada vez que ejecutamos el bucle, llamamos al método `fetch_assoc` para recuperar el valor almacenado en cada celda y dar salida al resultado mediante la declaración `echo`.

Cuando se visualizan datos en un navegador cuya fuente era (o puede haber sido) una entrada de usuario, siempre hay un riesgo de que haya caracteres HTML incrustados en ella, incluso si crees que la han desinfectado previamente. Potencialmente podrían usarlos para un ataque de secuencias de comandos entre sitios (XSS). La forma más sencilla de evitar esta posibilidad es incorporar toda la salida dentro de una llamada a la función `htmlspecialchars`, que reemplaza todos estos caracteres por inofensivas entidades HTML. Esta técnica se implementó en el anterior ejemplo y se utilizará en muchos de los siguientes ejemplos.

Probablemente estarás de acuerdo en que todas estas búsquedas múltiples y demás son bastante engorrosas, y que debería haber un método más eficiente para lograr el mismo resultado. Y, de hecho, hay un método mejor, que es extraer filas una tras otra.

En el Capítulo 9, hablé sobre la primera, segunda y tercera formas normales. Habrás notado que la tabla *classics* no las satisface, porque tanto los detalles de autores como de libros están incluidos en la misma tabla. Esto se debe a que creamos esta tabla antes de encontrar la normalización. Sin embargo, con el propósito de ilustrar el acceso a MySQL desde PHP, la reutilización de esta tabla evita la molestia de escribir un nuevo conjunto de datos de prueba, por lo que nos quedaremos con ella por el momento

Obtención de una fila

Para obtener las filas una por una, tienes que sustituir el bucle `for` del Ejemplo 10-4 con el resaltado en negrita en el Ejemplo 10-5; descubrirás que obtienes exactamente el mismo resultado que se mostró en la Figura 10-1. Puede que desees guardar este archivo revisado con el nombre *fetchrow.php*.

Ejemplo 10-5. Obtención de resultados de filas una por una

```
<?php // fetchrow.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($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 'Author: '. htmlspecialchars($row['author'])      . '<br>';
    echo 'Title:  '. htmlspecialchars($row['title'])       . '<br>';
    echo 'Category: ' . htmlspecialchars($row['category']) . '<br>';
    echo 'Year:   '. htmlspecialchars($row['year'])        . '<br>';
    echo 'ISBN:   '. htmlspecialchars($row['isbn'])     . '<br><br>';
}

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

En este código modificado, solo se hace la quinta parte de las interrogaciones al objeto `$result` (en comparación con el ejemplo anterior), y solo se hace una búsqueda en el objeto en cada iteración del bucle, porque cada fila se obtiene en su totalidad mediante el método `fetch_array`. Este método devuelve una sola fila de datos en forma de matriz, que luego se asigna a la matriz `$row`.

El método `fetch_array` puede devolver tres tipos de matrices en función del valor que se le pase:

`MYSQLI_NUM`
: Matriz numérica. Cada columna aparece en la matriz en el orden que definiste cuando creaste (o alteraste) la tabla. En nuestro caso, el elemento cero de la matriz contiene la columna *author*, el elemento 1 contiene la columna *title,* etc.

`MYSQLI_ASSOC`
: Matriz asociativa. Cada clave es el nombre de una columna. Debido a que las posiciones de datos se referencian por el nombre de la columna (en lugar de por el número de índice), utiliza esta opción en tu código, cuando sea posible, para hacer la depuración más fácil y ayudar a otros programadores a administrar mejor tu código.

`MYSQLI_BOTH`
: Matriz asociativa y numérica.

Las matrices asociativas suelen ser más útiles que las numéricas porque puedes hacer referencia a cada 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

PHP devolverá eventualmente el espacio de memoria que ha asignado para los objetos después de que hayas terminado con el script, así que en scripts pequeños, normalmente no tienes que preocuparte de liberar la memoria. Sin embargo, si asignas una gran cantidad de objetos de resultado u obtienes grandes cantidades de datos, puede ser una buena idea liberar la memoria que has estado utilizando, para prevenir problemas más adelante en tu script.

Esto es particularmente importante en las páginas de mayor tráfico, porque la cantidad de memoria consumida en una sesión puede crecer rápidamente. Por este motivo, puedes observar las llamadas a los métodos `close` de los objetos `$result` y `$conn` en los scripts anteriores, que se emiten tan pronto como cada objeto ya no es necesario, así:

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

Idealmente, deberías cerrar cada objeto de resultado cuando hayas terminado de usarlo y luego cerrar el objeto de conexión cuando tu script ya no acceda a MySQL. Esta mejor práctica asegura que los recursos se devuelvan al sistema tan rápido como sea posible para mantener MySQL funcionando óptimamente y reduce la incertidumbre sobre si PHP devolverá la memoria sin usar a tiempo para la próxima vez que la necesites.

Un ejemplo práctico

Es hora de escribir nuestro primer ejemplo de inserción y borrado de datos de una tabla MySQL usando PHP. Te recomiendo que escribas el Ejemplo 10-6 y lo guardes en tu directorio de desarrollo web con el nombre de archivo *sqltest.php*. Puedes ver un ejemplo de la salida del programa en la Figura 10-2.

El Ejemplo 10-6 crea un formulario HTML estándar. El Capítulo 11 explica los formularios en detalle, pero en este capítulo doy por sentado el manejo de formularios y solo me ocupo de la interacción con la base de datos.

Ejemplo 10-6. Inserción y borrado de datos utilizando sqltest.php

```
<?php // sqltest.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  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 "DELETE failed<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 "INSERT failed<br><br>";
  }

  echo <<<_END
  <form action="sqltest.php" method="post"><pre>
    Author <input type="text" name="author">
     Title <input type="text" name="title">
  Category <input type="text" name="category">
```

```
    Year <input type="text" name="year">
    ISBN <input type="text" name="isbn">
         <input type="submit" value="ADD RECORD">
  </pre></form>
_END;

  $query = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");

  $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 $r4
  </pre>
  <form action='sqltest.php' method='post'>
  <input type='hidden' name='delete' value='yes'>
  <input type='hidden' name='isbn' value='$r4'>
  <input type='submit' value='DELETE RECORD'></form>
_END;
  }

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

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

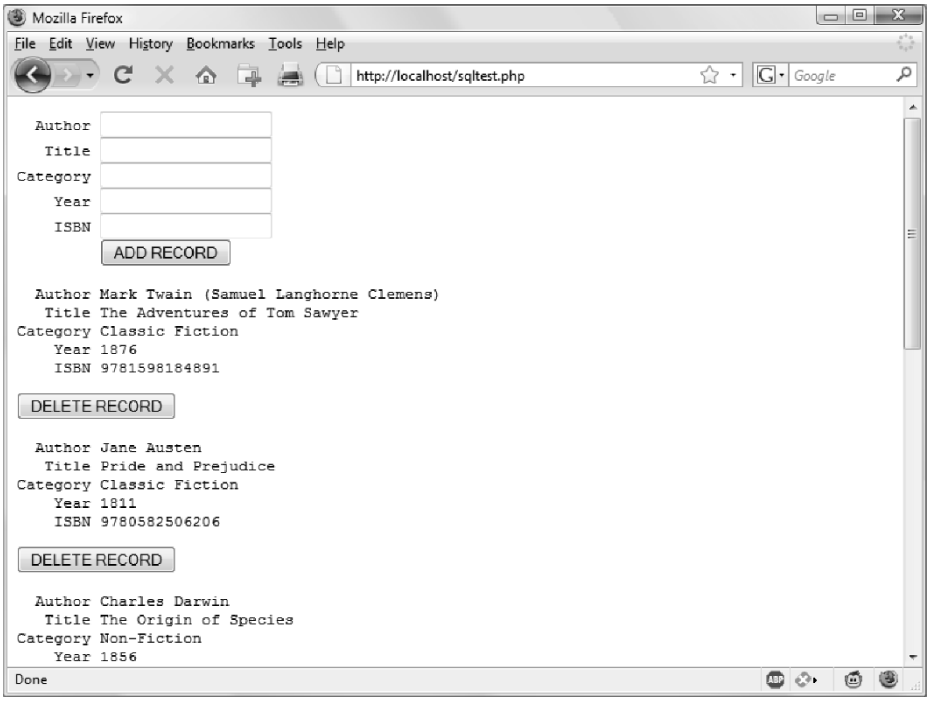

Figura 10-2. La salida del Ejemplo 10-6, sqltest.php

Con más de 80 líneas de código, este programa puede parecer desalentador, pero no te preocupes, ya has escrito muchas de esas líneas del Ejemplo 10-5, y lo que hace el código es en realidad bastante simple.

En primer lugar, comprueba si se han realizado entradas y, a continuación, inserta nuevas entradas en la tabla *classics* de la base de datos *publications* o borra una fila de ella, en función de las entradas suministradas. Independientemente de si hubo entradas, el programa a continuación, da salida a todas las líneas de la tabla en el navegador. Veamos cómo funciona.

La primera sección del nuevo código comienza utilizando la función `isset` para verificar si se han contabilizado en el programa los valores de todos los campos. Tras la confirmación, cada línea dentro de la sentencia `if` llama a la función `get_post`, que aparece al final del programa. Esta función tiene poco trabajo, pero es crítico: obtener la entrada del navegador.

Por razones de claridad y brevedad, y para explicar las cosas de la manera más sencilla posible, muchos de los siguientes ejemplos omiten ciertas precauciones de seguridad, muy sensibles, que los hubieran hecho más largos y posiblemente hubieran impedido explicar la función de los ejemplos de una manera más clara. Por lo tanto, es importante que no omitas la sección que se presenta más adelante en este capítulo sobre cómo evitar que pirateen tu base de datos ("Prevención de intentos de piratería" en la página 258), en la que podrás conocer las acciones complementarias que puedes realizar con tu código para asegurar la base de datos.

La matriz $_POST

He mencionado en un capítulo anterior que 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), por este motivo lo utilizamos aquí. El servidor web agrupa todas las entradas del usuario (incluso si el formulario se ha rellenado con cientos de campos) y las pone en una matriz llamada `$_POST`.

`$_POST` es una matriz asociativa, que apareció en el Capítulo 6. En función de si se ha configurado un formulario para utilizar el método POST o GET, ya sea el método `$_POST` o la matriz asociativa `$_GET`, se rellenará con los datos del formulario. Ambos se pueden leer exactamente de la misma manera.

Cada campo tiene un elemento en la matriz que lleva el nombre de ese campo. Por lo tanto, si un formulario contiene un campo llamado `isbn`, la matriz `$_POST` contiene un elemento marcado con la palabra `isbn`. El programa PHP puede leer ese campo refiriéndose bien a `$_POST['isbn']` o a `$_POST["isbn"]` (las comillas simples y dobles tienen el mismo efecto en este caso).

Si la sintaxis `$_POST` sigue pareciéndote compleja, puedes sentirte seguro de utilizar la convención que he mostrado en el Ejemplo 10-6, copiar la entrada del usuario a otras variables, y te olvidas de `$_POST` después. Esto es normal en los programas PHP: recuperan todos los campos de `$_POST` al principio del programa y luego se ignora.

No hay razón para escribir en un elemento de la matriz `$_POST`. Su único propósito es comunicar información desde el navegador al programa, y es mejor copiar datos a tus propias variables antes que alterarlas.

Por lo tanto, volvemos a la función `get_post`, que pasa cada elemento que recupera a través del método `real_escape_string` del objeto de conexión para escapar de las comillas que un hacker pueda haber insertado con el fin de entrar o alterar la base de datos, así:

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

Eliminación de un registro

Antes de verificar si se han enviado nuevos datos, el programa comprueba si la variable `$_POST['delete']` tiene un valor. Si es así, el usuario ha hecho clic en el botón DELETE RECORD para borrar un registro. En este caso, el valor de `$isbn` también se habrá enviado.

Como recordarás, el ISBN identifica de manera única cada registro. El formulario HTML adjunta el ISBN a la cadena de consulta `DELETE FROM` creada en la variable `$query`, que pasa entonces al método `query` del objeto `$conn` para emitirlo a MySQL.

Si `$_POST['delete']` no se ha establecido (y por lo tanto no hay ningún registro que borrar), se verifican `$_POST['author']` y otros valores enviados. Si a todos se les han dado valores, `$query` se configura con un comando `INSERT INTO`, seguido de los cinco valores a insertar. A continuación, la cadena se pasa al método `query`, que una vez completado devuelve `TRUE` o `FALSE`. Si se devuelve `FALSE` aparece un mensaje de error, como por ejemplo, el siguiente:

```
if (!$result) echo "INSERT failed<br><br>";
```

Visualización del formulario

Antes de mostrar el pequeño formulario (como se muestra en la Figura 10-2), el programa desinfecta las copias de los elementos que saldrán de la matriz `$row` en las variables `$r0` a `$r4` y las pasa a la función `htmlspecialchars`, para reemplazar cualesquiera caracteres HTML potencialmente peligrosos con entidades HTML inofensivas.

A continuación se presenta la parte del código que muestra la salida, con una estructura `echo <<<_END... _END`, tal y como se ha visto en los capítulos anteriores, que presenta todo lo que hay entre las etiquetas `_END`.

En lugar de usar el comando `echo`, el programa podría salir de PHP usando `?>`, emitir el HTML y luego volver a entrar con `<?php` para procesar PHP. El estilo que se utiliza es una cuestión de preferencia del programador, pero siempre recomiendo permanecer dentro del código PHP, por las siguientes razones:

- Esto deja muy claro cuando se está depurando (y también para otros usuarios) que todo dentro de un archivo *.php* es código PHP. Por lo tanto, no hay necesidad de buscar errores en HTML.
- Cuando se desea incluir una variable PHP directamente dentro de HTML, solo tienes que escribirla. Si hubieras regresado a HTML, habrías tenido que reiniciar temporalmente el procesamiento de PHP, acceder a la variable y luego volver a salir.

La sección del formulario HTML simplemente establece la acción del formulario en *sqltest.php*. Esto significa que cuando se envíe el formulario, el contenido de los campos del mismo se enviará al archivo *sqltest.php*, que es el programa en sí. El formulario también está configurado para enviar los campos como POST en lugar de mediante una petición GET. Esto se debe a que las peticiones GET se añaden al fichero URL que se está enviando y pueden verse desordenadas en el navegador. También permiten a los usuarios modificar fácilmente las presentaciones y tratar de piratear el servidor (aunque eso también se puede conseguir con herramientas de desarrollo dentro del navegador). Además, si eludes las peticiones GET se evita que haya demasiada información en los archivos de registro del servidor. Por lo tanto, siempre que sea posible, debes utilizar las entregas POST, que también tienen la ventaja de revelar menos datos enviados.

Después de dar salida a los campos del formulario, HTML muestra un botón de envío con el nombre ADD RECORD y cierra el formulario. Observa en este caso las etiquetas `<pre>` y `</pre>`, que se han usado para forzar una fuente monoespaciada que alinea todas las entradas ordenadamente. Los retornos de carro al final de cada línea también se emiten cuando están dentro de las etiquetas `<pre>`.

Consulta de la base de datos

A continuación, el código vuelve a nuestro territorio familiar del Ejemplo 10-5, en el que se envía una consulta a MySQL que pide ver todos los registros en la tabla *classics*, así:

```
$query = "SELECT * FROM classics";
$result = $conn->query($query);
```

Después de eso, `$rows` se establece a un valor que representa el número de filas en la tabla:

```
$rows = $result->num_rows;
```

Si usamos el valor de `$rows`, se introduce un bucle `for` para mostrar el contenido de cada fila. Luego el programa rellena la matriz `$row` con una fila de resultados y llama

al método `fetch_array de $result`, al que le pasa el valor constante `MYSQLI_NUM`, que fuerza el retorno de una matriz numérica (en lugar de asociativa), como esta:

```
$row = $result->fetch_array(MYSQLI_NUM);
```

Con los datos en `$row`, ahora es fácil mostrarlos dentro de la declaración `echo` heredoc a continuación, en la que he decidido usar una etiqueta `<pre>` para alinear la presentación de cada registro de una manera atractiva.

Después de la visualización de cada registro, hay un segundo formulario que también se envía a *sqltest.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.

Luego se muestra un botón de envío con el nombre DELETE RECORD y se cierra el formulario. Después, unas llaves completan el bucle `for`, que continuará hasta que todos los registros se hayan visualizado, momento en el que los métodos `close` de los objetos `$result y $conn` se cierran para liberar recursos de nuevo a PHP:

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

Finalmente, viene la definición de la función `get_post`, que ya hemos visto. Y eso es todo, tenemos nuestro primer programa PHP para manipular una base de datos MySQL. Así que, veamos lo que este puede hacer.

Una vez que hayas escrito el programa (y corregido cualquier error de escritura), intenta introducir los siguientes datos en los distintos campos de entrada para añadir un nuevo registro para el libro *Moby Dick* a la base de datos:

```
Herman Melville
Moby Dick
Fiction
1851
9780199535729
```

Ejecución del programa

Cuando hayas enviado estos datos con el botón ADD RECORD, desplázate hacia abajo hasta la parte inferior de la página web para ver la nueva adición. Debería parecerse a la Figura 10-3.

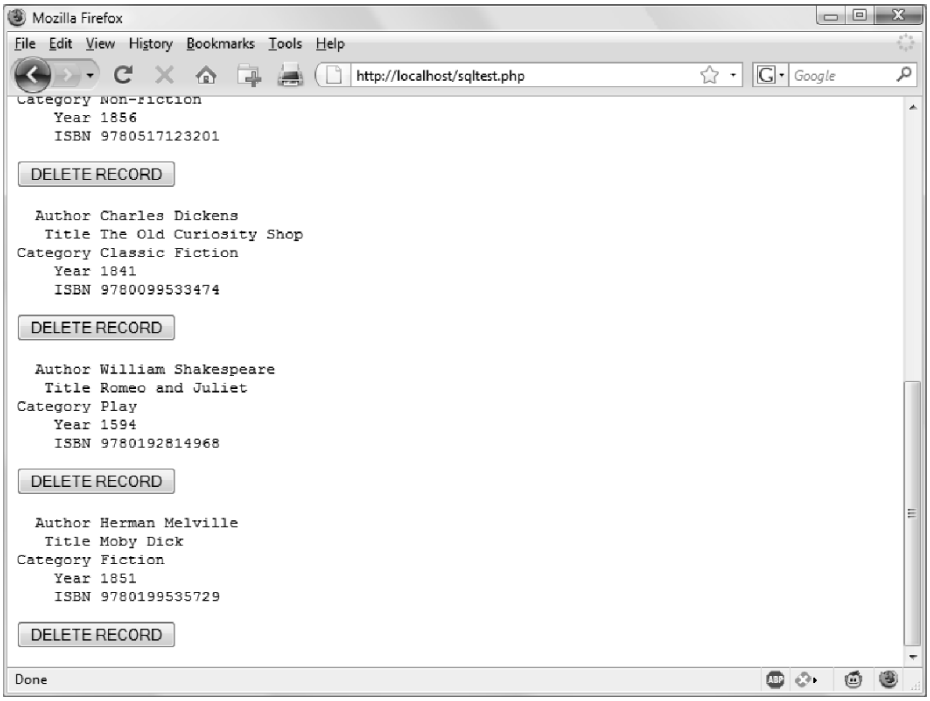

Figura 10-3. El resultado de añadir Moby Dick a la base de datos

Ahora veamos cómo funciona la eliminación de un registro mediante la creación de un registro ficticio. Intenta introducir solo el número 1 en cada uno de los cinco campos y haz clic en el botón ADD RECORD. Si ahora te desplazas hacia abajo, verás un nuevo registro que contiene solo unos (1). Obviamente, este registro no es útil en esta tabla, así que ahora haz clic en el botón DELETE RECORD y desplázate hacia abajo de nuevo para confirmar que el registro ha sido eliminado.

Supongamos que todo ha funcionado, ahora puedes añadir y borrar registros a voluntad. Intenta hacer esto unas cuantas veces, pero deja los registros principales en su sitio (incluido el nuevo para *Moby Dick*), ya que los vamos a utilizar más tarde. También puedes intentar añadir el registro con todos los unos (1) un par de veces y verás el mensaje de error que recibes la segunda vez, que indica que ya existe un ISBN con el número 1.

MySQL práctico

Ahora estás preparado para ver algunas técnicas prácticas que puedes utilizar en PHP para acceder a la base de datos MySQL, incluidas tareas como crear y eliminar tablas; insertar, actualizar y eliminar datos, y proteger tu base de datos y tu sitio web de usuarios

malintencionados. Ten en cuenta que en los siguientes ejemplos se da por sentado que ya has creado el programa *login.php* discutido anteriormente en este capítulo.

Creación de una tabla

Supongamos que estás trabajando para un parque de vida salvaje y necesita crear una base de datos para contiene detalles sobre todos los tipos de gatos que alberga. Te han dicho que hay nueve familias de felinos: leones, tigres, jaguares, leopardos, pumas, guepardos, linces, caracales y gatos domésticos, así que necesitarás una columna para ellos. Después a cada felino se le ha dado un nombre, así que esa es otra columna, y también quieres llevar un registro de sus edades, que es otra. Por supuesto, probablemente necesitarás más columnas más adelante, tal vez para mantener los requisitos de la dieta, inoculaciones y otros detalles, pero por ahora es suficiente para ponerse en marcha. También se necesita un identificador único para cada animal, por lo que también se decide crear una columna para eso llamada *id*.

El Ejemplo 10-7 muestra el código que puedes usar para crear una tabla MySQL que contenga todo esto con la asignación de la consulta principal en negrita.

Ejemplo 10-7. Creación de una tabla llamada cats

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

  $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 ("Database access failed");
?>
```

Como puedes ver, la consulta MySQL se parece a la que escribirías directamente en la línea de comandos, excepto sin el punto y coma final.

Descripción de una tabla

Cuando no estás conectado a la línea de comandos de MySQL, aquí tienes un útil código que puedes usar para verificar que se ha creado una tabla correctamente desde un navegador. Simplemente emite la consulta `DESCRIBE cats` y a continuación se genera una tabla HTML con cuatro encabezados (*Column*, *Type*, *Null* y *Key*) debajo de los cuales se encuentran las columnas dentro de la tabla. Para usarlo con otras tablas,

simplemente reemplaza el nombre `cats` en la consulta por el de la nueva tabla (ver Ejemplo 10-8).

Ejemplo 10-8. Descripción de la tabla cats

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

  $query = "DESCRIBE cats";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");

  $rows = $result->num_rows;

  echo
"<table><tr><th>Column</th><th>Type</th><th>Null</th><th>Key</th></t
r>";

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

    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k)
      echo "<td>" . htmlspecialchars($row[$k]) . "</td>";
    echo "</tr>";
  }

  echo "</table>";
?>
```

La salida del programa debería tener este aspecto:

```
Column    Type          Null      Key
Id        smallint(6)   NO        PRI
Family    varchar(32)   NO
name      varchar(32)   NO
age       tinyint(4)    NO
```

Eliminación de una tabla

Eliminar una tabla es muy fácil de hacer y, por lo tanto, es muy peligroso, así que ten cuidado. El Ejemplo 10-9 muestra el código que necesitas. Sin embargo, no te recomiendo que lo pruebes hasta que hayas repasado los otros ejemplos (hasta "Realizar consultas adicionales" en la página 253), ya que eliminará la tabla *cats* y tendrás que recrearla con el Ejemplo 10-7.

Ejemplo 10-9. Eliminación de la tabla cats

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

  $query = "DROP TABLE cats";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");
?>
```

Adición de datos

Ahora vamos a añadir algunos datos a la tabla onc el código del Ejemplo 10-10.

Ejemplo 10-10. Adición de datos a la tabla cats

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

  $query = "INSERT INTO cats VALUES(NULL, 'Lion', 'Leo', 4)";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");
?>
```

Puede que quieras añadir un par de datos más; modifica `$query` como se indica a continuación y llama de nuevo al programa en tu navegador:

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

Por cierto, ¿has visto el valor `NULL` pasado como primer parámetro? Esto se debe a que la columna *id* es del tipo `AUTO_INCREMENT`, y MySQL decidirá qué valor asignar de acuerdo al siguiente número disponible en la secuencia. Entonces, simplemente pasamos un valor `NULL`, que será ignorado.

Por supuesto, la manera más eficiente de rellenar MySQL con datos es crear una matriz e insertar los datos con una sola consulta.

En este punto del libro trato de centrarme en mostrarte cómo insertar directamente datos en MySQL (y proporcionarte algunas precauciones de seguridad para mantener el proceso seguro). Sin embargo, más adelante en este libro pasaremos a un método mejor que puedes emplear y que implica el uso de marcadores (ver "Uso de marcadores de posición" en la página 256), que hace que sea virtualmente imposible para los usuarios inyectar hacks maliciosos en la base de datos. Por lo tanto, al leer esta sección, comprende que esta es la sección de los fundamentos de cómo funciona la inserción en MySQL, y recuerda que lo mejoraremos más tarde.

Recuperación de datos

Ahora que se han introducido algunos datos en la tabla *cats*, el Ejemplo 10-11 muestra cómo puedes comprobar que se han insertado correctamente.

Ejemplo 10-11. Recuperación de filas de la tabla cats

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

  $query = "SELECT * FROM cats";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");

  $rows = $result->num_rows;
  echo "<table><tr> <th>Id</th>
<th>Family</th><th>Name</th><th>Age</th></tr>";

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

    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k)
      echo "<td>" . htmlspecialchars($row[$k]) . "</td>";
    echo "</tr>";
  }

  echo "</table>";
?>
```

Este código simplemente emite la consulta MySQL `SELECT * FROM cats` y luego muestra todas las filas devueltas. Su resultado es el siguiente:

```
Id    Family       Name         Age
1     Lion         Leo          4
2     Cougar       Growler      2
3     Cheetah      Charly       3
```

Aquí puedes ver que la columna *id* se ha autoincrementado correctamente.

Actualización de datos

El cambio de datos que ya has insertado es también bastante sencillo. ¿Has notado la ortografía de *Charly* para el nombre del guepardo? Corrijamos eso con *Charlie*, como en el Ejemplo 10-12.

Ejemplo 10-12. Cambio de nombre de Charly el guepardo por Charlie

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

  $query = "UPDATE cats SET name='Charlie' WHERE name='Charly'";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");
?>
```

Si ejecutas de nuevo el Ejemplo 10-11 verás que ahora el resultado es el siguiente:

```
Id      Family      Name        Age
1       Lion        Leo         4
2       Cougar      Growler     2
3       Cheetah     Charlie     3
```

Borrado de datos

Growler, el puma, ha sido transferido a otro zoológico, así que es hora de eliminarlo de la base de datos, como puedes ver en el Ejemplo 10-13.

Ejemplo 10-13. Borrado de Growler el puma de la tabla cats

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

  $query = "DELETE FROM cats WHERE name='Growler'";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");
?>
```

Este ejemplo utiliza la consulta estándar `DELETE FROM`, y cuando se ejecuta el Ejemplo 10-11, se puede ver que la fila se ha eliminado en la siguiente salida:

```
Id      Family      Name        Age
1       Lion        Leo         4
3       Cheetah     Charlie     3
```

Uso de AUTO_INCREMENT

Cuando se utiliza `AUTO_INCREMENT`, no se puede saber qué valor se ha dado a una columna antes de insertar una línea. Pero, si necesitas saberlo, debes preguntarselo a continuación a MySQL mediante la función `mysql_insert_id`. Esta necesidad es habitual: por ejemplo, cuando se procesa una compra, se puede insertar un nuevo cliente en la tabla *Customers* y, a continuación, referirse a la recién creada *CustId* al insertar una compra en la tabla *Purchases*.

Se recomienda usar `AUTO_INCREMENT` en lugar de seleccionar el ID más alto en la columna *id* e incrementarlo en uno, ya que las consultas simultáneas podrían cambiar los valores en esa columna después de que se haya obtenido el valor más alto y antes de que se almacene el valor calculado.

El Ejemplo 10-10 puede reescribirse como el Ejemplo 10-14 para mostrar este valor después de cada inserción.

Ejemplo 10-14. Adición de datos a la tabla cats e informe del ID de inserción

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

  $query = "INSERT INTO cats VALUES(NULL, 'Lynx', 'Stumpy', 5)";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");

  echo "The Insert ID was: " . $conn->insert_id;
?>
```

El contenido de la tabla debería tener el siguiente aspecto (ten en cuenta que el valor de *id* anterior de 2 no se reutiliza, ya que esto podría causar complicaciones en algunos casos):

```
Id      Family      Name        Age
1       Lion        Leo         4
3       Cheetah     Charlie     3
4       Lynx        Stumpy      5
```

Uso de los ID de inserción

Es muy común insertar datos en varias tablas: un libro seguido por su autor, una cliente seguido de su compra, etc. Al hacer esto con un autoincremento deberás conservar el ID de inserción devuelto para almacenarlo en la tabla relacionada.

Por ejemplo, supongamos que estos felinos pueden ser "adoptados" por el público como un medio de recaudar fondos, y que cuando los datos de un felino nuevo se almacenan en la tabla *cats*, también queremos crear una clave para relacionarlos con el dueño adoptivo del animal. El código para hacer esto es similar al del Ejemplo 10-14, excepto que el ID de inserción devuelto se almacena en la variable `$insertID` y luego se utiliza como parte de la consulta posterior:

```
$query    = "INSERT INTO cats VALUES(NULL, 'Lynx', 'Stumpy', 5)";
$result   = $conn->query($query);
$insertID = $conn->insert_id;

$query    = "INSERT INTO owners VALUES($insertID, 'Ann', 'Smith')";
$result   = $conn->query($query);
```

Ahora el felino está conectado con su "dueño" a través del ID único del felino, que lo ha creado automáticamente `AUTO_INCREMENT`.

Realización de consultas adicionales

Vale, ya hemos tenido suficiente diversión felina. Para explorar algunas preguntas un poco más complejas, necesitamos volver a utilizar las tablas *customers* y *classics* que creaste en el Capítulo 8. Habrá dos clientes en la tabla *customers*; la tabla *classics* contiene los detalles de algunos libros. También comparten una columna común de ISBN, llamada *isbn*, que puedes usar para realizar consultas adicionales.

Por ejemplo, para visualizar todos los clientes junto con los títulos y autores de los libros que han comprado, puedes usar el código del Ejemplo 10-15.

Ejemplo 10-15. Realización de una consulta adicional

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

  $query = "SELECT * FROM customers";
  $result = $conn->query($query);
  if (!$result) die ("Database access failed");

  $rows = $result->num_rows;

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

```
    echo htmlspecialchars($row[0]) . " purchased ISBN " .
         htmlspecialchars($row[1]) . ":<br>";

    $subquery = "SELECT * FROM classics WHERE isbn='$row[1]'";
    $subresult = $conn->query($subquery);
    if (!$subresult) die ("Database access failed");

    $subrow = $subresult->fetch_array(MYSQLI_NUM);
    echo "  " . htmlspecialchars("'$subrow[1]'") . " by " .
                          htmlspecialchars(  $subrow[0]) . "<br><br>";
  }
?>
```

Este programa utiliza una consulta inicial a la tabla *customers* de clientes para buscar a todos los clientes y luego, dados los ISBN de los libros que cada cliente compró, realiza una nueva consulta a la tabla *classics* para averiguar el título y el autor de cada uno. La salida de este código debería ser similar a la siguiente:

```
Joe Bloggs purchased ISBN 9780099533474:
  'The Old Curiosity Shop' by Charles Dickens

Jack Wilson purchased ISBN 9780517123201:
  'The Origin of Species' by Charles Darwin

Mary Smith purchased ISBN 9780582506206:
  'Pride and Prejudice' by Jane Austen
```

Por supuesto, aunque no ilustraría la realización de consultas adicionales, en este caso en particular también podrías devolver la misma información mediante una consulta `NATURAL JOIN` (ver Capítulo 8), como esta:

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

Prevención de intentos de piratería

Si no lo has investigado, es posible que te resulte difícil darte cuenta de lo peligroso que es pasar las entradas de usuario sin comprobar a MySQL. Por ejemplo, supongamos que tienes un sencillo código para verificar a un usuario, y se ve así:

```
$user  = $_POST['user'];
$pass  = $_POST['pass'];
$query = "SELECT * FROM users WHERE user='$user' AND
pass='$pass'";
```

A primera vista, podrías pensar que este código está perfectamente bien. Si el usuario introduce valores de `fredsmith` y `mypass` para `$user` y `$pass`, respectivamente, entonces la cadena de consulta, tal como se pasó a MySQL, será como sigue:

```
SELECT * FROM users WHERE user='fredsmith' AND pass='mypass'
```

Todo esto está muy bien, pero ¿qué pasa si alguien introduce lo siguiente para `$user` (y ni siquiera introduce nada para `$pass`)?

```
admin' #
```

Veamos la cadena que se enviaría a MySQL:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

¿Ves el problema ahí? Se ha producido un ataque de *inyección de SQL*. En MySQL, el símbolo # representa el inicio de un comentario. Por lo tanto, el usuario se registrará como *admin* (asumiendo que hay un *admin* de usuario), sin tener que introducir una contraseña. A continuación se muestra en negrita la parte de la consulta que se ejecutará; el resto se ignorará:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Pero deberías considerarte muy afortunado si eso es todo lo que te hace un usuario malicioso. Por lo menos todavía puedes ser capaz de entrar en tu aplicación y deshacer cualquier cambio que el usuario haga como *administrador*. Pero ¿qué pasa con el caso en el que tu código de aplicación elimina a un usuario de la base de datos? El código podría parecerse a esto:

```
$user  = $_POST['user'];
$pass  = $_POST['pass'];
$query = "DELETE FROM users WHERE user='$user' AND pass='$pass'";
```

De nuevo, esto parece bastante normal a primera vista, pero ¿qué pasa si alguien introduce lo siguiente para `$user`?

```
anything' OR 1=1 #
```

Esto lo interpretaría MySQL de la siguiente manera:

```
DELETE FROM users WHERE user='anything' OR 1=1 #' AND pass=''
```

¡Vaya, esa consulta SQL siempre será `TRUE`, y por lo tanto has perdido toda la base de datos de usuarios! Entonces, ¿qué puedes hacer con este tipo de ataque?

Pasos que puedes seguir

Lo primero es no confiar en las *comillas mágicas* integradas de PHP, que escapan automáticamente a cualquier carácter como comillas simples o dobles, precediéndolas con un barra invertida (\). ¿Por qué? Porque esta característica se puede desactivar. Muchos programadores lo hacen con el fin de poner su propio código de seguridad en su lugar, y no hay garantía de que esto no haya ocurrido en el servidor en el que estás trabajando. De hecho, esta característica ya estaba obsoleta a partir de PHP 5.3.0 y fue eliminada en PHP 5.4.0.

En su lugar, siempre debes utilizar el método `real_escape_string` para todas las llamadas a MySQL. El Ejemplo 10-16 es una función que puedes utilizar para eliminar

cualesquiera comillas mágicas añadidas a una cadena introducida por el usuario y, a continuación, desinfectarla correctamente.

Ejemplo 10-16. Cómo desinfectar correctamente una entrada de usuario para MySQL

```
<?php
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

La función `get_magic_quotes_gpc` devuelve `TRUE` si las comillas mágicas están activas. En ese caso, hay que eliminar cualquier barra oblicua que se haya añadido a una cadena, o el método `real_escape_string` podría terminar con un doble escape de algunos caracteres y crear cadenas dañadas. El Ejemplo 10-17 ilustra cómo incorporarías `mysql_fix_string` en tu código.

Ejemplo 10-17. Cómo acceder de forma segura a MySQL con la entrada de usuario

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

  $user  = mysql_fix_string($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);
  }
?>
```

Estas precauciones son, sin embargo, cada vez menos importantes porque hay una manera mucho más fácil y segura de acceder a MySQL, lo que evita la necesidad de este tipo de funciones: el uso de marcadores de posición, que se explica a continuación.

Uso de marcadores de posición

Todos los métodos que has visto hasta ahora funcionan con MySQL pero tienen implicaciones de seguridad, con cadenas que constantemente requieren escapar para prevenir riesgos de seguridad. Así que, ahora que conoces lo básico, déjame presentarte la mejor y más recomendada manera de interactuar con MySQL, que es más o menos a prueba de balas en términos de seguridad. Una vez que hayas leído esta sección, ya no

debes utilizar la inserción directa de datos en MySQL (aunque era importante mostrarte cómo hacer esto), pero siempre debes usar marcadores de posición en su lugar.

Entonces, ¿qué son 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) envíe datos que se interpretan como declaraciones MySQL (y la posibilidad de hacking a la que podrían dar lugar).

La tecnología funciona requiriéndote que primero prepares la declaración que deseas que se ejecute en MySQL, pero deja todas las partes de la declaración que se refieren a datos como simples signos de interrogación.

En MySQL sencillo, las sentencias preparadas se parecen al Ejemplo 10-18.

Ejemplo 10-18. 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;
```

Esto puede ser engorroso de enviar a MySQL, por lo que la extensión `mysqli` hace que la gestión de marcadores de posición sea más fácil con un método ya preparado llamado `prepare`, al que llamamos así:

```
$stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
```

El objeto `$stmt` (que es la abreviatura de *statement*) devuelto por este método se utiliza para enviar los datos al servidor en lugar de los signos de interrogación. Su primer uso es vincular algunas variables PHP con cada uno de los signos de interrogación (los parámetros del marcador de posición) sucesivamente, así:

```
$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`: Los datos son enteros.
- `d`: Los datos son dobles.
- `s`: Los datos son una cadena.
- `b`: Los datos son `BLOB` (y se enviarán en paquetes).

Con las variables vinculadas a la sentencia preparada, ahora es necesario rellenar estas variables con los datos que se pasarán a MySQL, así:

```
$author   = 'Emily Brontë';
$title    = 'Wuthering Heights';
$category = 'Classic Fiction';
$year     = '1847';
$isbn     = '9780553212587';
```

En este punto, PHP tiene todo lo que necesitas para ejecutar la sentencia preparada, por lo que puedes emitir el siguiente comando, que llama al método `execute` del objeto `$stmt` creado anteriormente:

```
$stmt->execute();
```

Antes de continuar, tiene sentido verificar si el comando se ha ejecutado correctamente. He aquí cómo puedes hacerlo comprobando la propiedad `affected_rows` de `$stmt`:

```
printf("%d Row inserted.\n", $stmt->affected_rows);
```

En este caso, la salida debería indicar que se ha insertado una fila.

Una vez que estés satisfecho de que la sentencia se haya ejecutado correctamente (o hayas resuelto cualquier error), puedes cerrar el objeto `$stmt`, así:

```
$stmt->close();
```

y finalmente cerrar el objeto `$conn` (suponiendo que también has terminado con él), así:

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

Cuando pones todo esto junto, el resultado es el Ejemplo 10-19.

Ejemplo 10-19. Emisión de declaraciones preparadas

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

  $stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
  $stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);

  $author   = 'Emily Brontë';
  $title    = 'Wuthering Heights';
  $category = 'Classic Fiction';
  $year     = '1847';
  $isbn     = '9780553212587';

  $stmt->execute();
  printf("%d Row inserted.\n", $stmt->affected_rows);
  $stmt->close();
  $conn->close();
?>
```

Siempre que puedas, utiliza sentencias preparadas en lugar de las no preparadas, cerrarás un potencial agujero de seguridad, por lo que vale la pena dedicar algún tiempo a saber cómo usarlas.

Prevención de la inyección de HTML

Hay otro tipo de inyección de la que debes preocuparte, no por la seguridad de tus propios sitios web, sino para la privacidad y protección de tus usuarios. Se trata de *cross-ste scripting* también conocido como *ataque* XSS.

Esto ocurre cuando se permite la entrada por parte de un usuario de código HTML o, más a menudo, de código JavaScript mediante un comando y luego se muestra en tu sitio web. Un lugar donde esto es común es en un formulario de comentarios. Lo que sucede con más frecuencia es que un usuario malicioso intenta escribir código que roba cookies de los usuarios de tu sitio, lo que a veces incluso les permite descubrir nombre de usuario y contraseña si no están bien gestionados, o cualquier otra información que podría habilitar el secuestro de sesión (en el que el inicio de sesión de un usuario lo asume un hacker, ¡que podría entonces hacerse cargo de la cuenta de esa persona!). O el usuario malicioso podría lanzar un archivo para descargar un troyano en el ordenador de un usuario.

Pero prevenir esto es tan sencillo como llamar a la función `htmlentities`, que elimina todo el marcado HTML y lo reemplaza con un formulario que muestra los caracteres, pero no permite que un navegador actúe sobre ellos. Por ejemplo, considera este HTML:

```
<script src='http://x.com/hack.js'>
</script><script>hack();</script>
```

Este código se carga en un programa JavaScript y luego ejecuta funciones maliciosas. Pero si primero se pasa a través de `htmlentities`, se convertirá en la siguiente cadena totalmente inofensiva:

```
&lt;script src='http://x.com/hack.js'&gt; &lt;/script&gt;
&lt;script&gt;hack();&lt;/script&gt;
```

Por lo tanto, si alguna vez vas a mostrar algo que tus usuarios introduzcan, ya sea inmediatamente o después de almacenarlo en una base de datos, primero necesitas desinfectarlo mediante la función `htmlentities`. Para ello, te recomiendo que crees una nueva función, como la primera en el Ejemplo 10-20, que puedes desinfectar tanto para inyecciones SQL como XSS.

Ejemplo 10-20. Funciones para prevenir ataques de inyección tanto en SQL como en XSS

```
<?php
  function mysql_entities_fix_string($conn, $string)
  {
    return htmlentities(mysql_fix_string($conn, $string));
  }
```

```
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

La función `mysql_entities_fix_string` llama primero a `mysql_fix_string` y luego pasa el resultado a través de `htmlentities` antes de devolver la cadena completamente desinfectada. Para utilizar cualquiera de estas funciones, debes tener ya un objeto de conexión activo abierto a una base de datos MySQL.

El Ejemplo 10-21 muestra la nueva versión de "protección superior" del Ejemplo 10-17.

Ejemplo 10-21. Cómo acceder de forma segura a MySQL y prevenir ataques XSS

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

  $user  = mysql_entities_fix_string($conn, $_POST['user']);
  $pass  = mysql_entities_fix_string($conn, $_POST['pass']);
  $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

  // Etc.

  function mysql_entities_fix_string($conn, $string)
  {
    return htmlentities(mysql_fix_string($conn, $string));
  }

  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Uso procidemental de mysqli

Si lo prefieres, hay un conjunto alternativo de funciones que puedes utilizar para acceder a `mysqli` de una manera procedimental (en lugar de orientada a objetos).

Así que, en lugar de crear un objeto `$conn` como este:

```
$conn = new mysqli($hn, $un, $pw, $db);
```

puedes utilizar lo siguiente:

```
$link = mysqli_connect($hn, $un, $pw, $db);
```

Para comprobar que se ha realizado la conexión y gestionarla, puedes utilizar un código como este:

```
if (mysqli_connect_errno()) die("Fatal Error");
```

Y para hacer una consulta MySQL, usarías un código como el siguiente:

```
$result = mysqli_query($link, "SELECT * FROM classics");
```

Una vez devuelto, `$result` contendrá los datos. Puedes averiguar el número de líneas devueltas como se indica a continuación:

```
$rows = mysqli_num_rows($result);
```

Un entero se devuelve en `$rows`. Puedes obtener los datos reales de las filas, una por una, que devuelve una matriz numérica, de la siguiente manera:

```
$row = mysqli_fetch_array($result, MYSQLI_NUM);
```

En este caso, `$row[0]` contendrá la primera columna de datos, `$row[1]` la segunda, y así sucesivamente. Como se describe en "Obtención de una fila" en la página 237, las filas también se pueden devolver como matrices asociativas o de ambos tipos, en función del valor que se pase en el segundo argumento.

Cuando necesites saber el ID de una operación de inserción, siempre puedes llamar a la función `mysqli_insert_id`, así:

```
$insertID = mysqli_insert_id($result);
```

Escapar las cadenas de forma procedimental con `mysqli` es tan fácil como usar lo siguiente:

```
$escaped = mysqli_real_escape_string($link, $val);
```

Y preparar una declaración con `mysqli` es tan simple como esto:

```
$stmt = mysqli_prepare($link, 'INSERT INTO classics VALUES(?,?,?,?,?)');
```

Para enlazar variables a la declaración preparada, se utilizaría lo siguiente:

```
mysqli_stmt_bind_param($stmt, 'sssss', $author, $title,
  $category, $year, $isbn);
```

Y para ejecutar la declaración preparada después de asignar las variables con los valores requeridos, se emitiría esta llamada:

```
mysqli_stmt_execute($stmt);
```

Para cerrar una declaración, hay que ejecutar el siguiente comando:

```
mysqli_stmt_close($stmt);
```

Y para cerrar la conexión a MySQL, introduce este comando:

```
mysqli_close($link);
```

Echa un vistazo a la documentación del Manual PHP para encontrar detalles completos sobre el uso de declaraciones preparadas, de forma procedimental o no (*http://us3.php.net/manual/es/mysqli.prepare.php*), y para más consejos sobre todos los aspectos de mysqli (*http://uk1.php.net/manual/es/book.mysqli.php*).

Ahora que ya has aprendido a integrar PHP con MySQL en varios lenguajes de programación diferentes, en el próximo capítulo veremos la creación de formularios fáciles de usar y la gestión de los datos presentados por ellos.

Preguntas

1. ¿Cómo te conectas a una base de datos MySQL con `mysqli`?
2. ¿Cómo envías una consulta a MySQL mediante `mysqli`?
3. ¿Cómo se puede recuperar una cadena que contiene un mensaje de error cuando ocurre un error de `mysqli`?
4. ¿Cómo se puede determinar el número de filas devueltas por una consulta `mysqli`?
5. ¿Cómo se puede recuperar una fila particular de datos de un conjunto de resultados de `mysqli`?
6. ¿Qué método de `mysqli` se puede usar para escapar apropiadamente de la entrada de usuario para prevenir el código inyección?
7. ¿Qué efectos negativos pueden producirse si no se cierran los objetos creados por métodos `mysqli`?

Consulta "Respuestas del Capítulo 10" en la página 712 en el Apéndice A para comprobar las respuestas a estas preguntas.