Capítulo 7

Trabajar con bases de datos y SQL

www.compumusicmania.com

Habilidades y conceptos clave

- Aprender conceptos sobre bases de datos y el lenguaje estructurado de consultas (SQL, Structured Query Language)
- Añadir, editar, borrar y ver registros utilizando las bases de datos MySQL y SQLite
- Recuperar registros de bases de datos con PHP
- Validar y guardar datos de entrada del usuario en una base de datos con PHP
- Escribir programas portátiles sustentados por bases de datos

---

Una de las razones de la popularidad de PHP como lenguaje de creación de scripts para Web es su amplio soporte a diferentes bases de datos. Este soporte facilita que los desarrolladores Web creen sitios sustentados en bases de datos y que se hagan nuevos prototipos de aplicaciones Web de manera rápida y eficiente, sin demasiada complejidad.

PHP soporta más de quince diferentes motores de bases de datos, incluidos Microsoft SQL Server, IBM DB2, PostgreSQL, MySQL y Oracle. Hasta PHP 5, este soporte se proporcionaba mediante extensiones nativas de las bases de datos, cada una con sus propias características y funciones; sin embargo, esto dificultaba a los programadores el cambio de una base a otra. PHP 5 rectificó esta situación introduciendo una API común para el acceso a base de datos: las extensiones de objetos de datos de PHP (PDO, *PHP Data Objects*), que proporcionan una interfaz unificada para trabajar con bases de datos y ayudan a que los desarrolladores manipulen diferentes bases de datos de manera consistente.

En la versión 5.3 de PHP, las extensiones PDO han sido mejoradas, con soporte para más motores de bases de datos y mejoras considerables en la seguridad y el desempeño. Para fines de compatibilidad con versiones anteriores, se sigue dando soporte a las extensiones de bases de datos nativas. Dado que en muchas ocasiones tendrás que escoger entre una extensión nativa (que puede ser más veloz u ofrecer más características) y una PDO (que ofrece portabilidad y consistencia para diferentes motores de bases de datos), en este capítulo se abordan las dos opciones: te presenta una introducción sobre PDO y también explica el funcionamiento de las dos extensiones nativas más populares de PHP: las extensiones mejoradas de MySQL y las extensiones de SQLite.

Introducción a bases de datos y SQL

En la era de Internet, la información no se acumula ya en gabinetes; en cambio, se almacena como 1 y 0 digitales en bases de datos electrónicas, que son los "contenedores" de datos que

www.compumusicmania.com

imponen cierta estructura en la información. Estas bases de datos electrónicas no sólo ocupan menos espacio físico que sus contrapartes de madera y metal; también contienen herramientas para ayudar a los usuarios a filtrar y recuperar información rápidamente utilizando diversos criterios. En particular, la mayoría de las bases de datos en la actualidad son *relacionales*, las cuales le permiten al usuario definir relaciones entre las tablas que conforman la base para realizar búsquedas y análisis más efectivos.

Actualmente hay disponibles muchos sistemas de administración de bases de datos; algunos son comerciales, otros gratuitos. Tal vez hayas oído mencionar algunos: Oracle, Microsoft Access, MySQL y PostgreSQL. Estos sistemas de bases de datos son aplicaciones poderosas, con muchas características, capaces de organizar y realizar búsquedas entre millones de registros a grandes velocidades; por ello son muy utilizados por empresas privadas y oficinas gubernamentales, en muchas ocasiones para llevar a cabo tareas de misión crítica.

Antes de comenzar con los vaivenes de la manipulación de registros con PHP, es esencial tener un claro entendimiento de los conceptos primordiales de las bases de datos. Si eres nuevo en el tema, las siguientes secciones te proporcionarán una base y también te permitirán comenzar a trabajar con ejercicios prácticos de SQL. Esta información será de utilidad para comprender el material más avanzado en secciones posteriores.

Comprender las bases de datos, registros y llaves primarias

Toda base de datos está compuesta por una o más *tablas*. Estas tablas, que estructuran los datos en filas y columnas, imponen una organización de datos. La figura 7-1 muestra una tabla típica.

El ejemplo contiene cifras por ventas de varias localidades, donde cada fila, también llamada *registro*, tiene información sobre diferentes lugares y años. Cada registro, a su vez, está dividido en columnas, también llamadas *campos*, y cada campo contiene un fragmento diferente de información. Esta estructura tabular facilita la búsqueda de registros, dentro de la tabla, que coincidan con ciertos *criterios*, por ejemplo: todos los lugares cuyas ventas sean mayores a $10 000, o las ventas de todos los lugares realizadas en el año 2008. Los registros

ID	*Año*	*Ubicación*	*Ventas* ($)
1	2007	Dallas	9 495
2	2007	Chicago	8 574
3	2007	Washington	12 929
4	2007	Nueva York	13 636
5	2007	Los Ángeles	8 748
6	2007	Boston	3 478
7	2008	Dallas	15 249
8	2008	Chicago	19 433
9	2008	Washington	3 738
10	2008	Nueva York	12 373
11	2008	Los Ángeles	16 162
12	2008	Boston	4 745

Figura 7-1 Tabla de ejemplo

www.compumusicmania.com

en la tabla no tienen ningún orden en particular; pueden organizarse alfabéticamente, por año, por total de ventas, por lugar o cualquier otro criterio que selecciones para especificar su orden. Así las cosas, para facilitar la identificación de un registro en particular, se hace necesario añadir un atributo identificador único para cada registro, como un número en serie o un código de secuencia. En el ejemplo anterior, cada registro es identificado por un campo "ID registro" único; este campo es conocido como la *llave primaria* de la tabla.

TIP

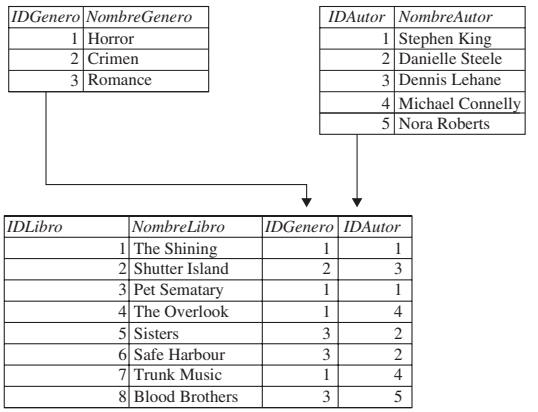
Una manera fácil de comprender estos conceptos es con una analogía. Imagina que la base de datos es una biblioteca, y que cada tabla es un anaquel dentro de la biblioteca. Así, un registro sería el equivalente electrónico de un libro en el anaquel y el título del libro sería su llave primaria. Sin el título sería imposible distinguir fácilmente un libro de otro. (La única manera de hacerlo así sería abrir cada libro y revisar su contenido, proceso que consumiría mucho tiempo, ¡mismo que la llave primaria nos ahorra!)

Una vez que tienes información en una tabla, por lo general querrás utilizarla para responder ciertas preguntas, por ejemplo: ¿cuántos lugares vendieron más de $5 000 en 2008? A estas preguntas se les conoce como *consultas*, y a las preguntas respondidas por la base de datos a esas consultas se les conoce como *colecciones de resultados*. Las consultas se formulan utilizando el lenguaje estructurado de consultas (*SQL*).

Comprender relaciones y llaves externas

Ya sabes que una sola base de datos puede contener varias tablas. En las bases de datos relacionales, las tablas pueden vincularse entre sí por uno o más campos que compartan en común, llamados *llaves externas*. Éstas hacen posible la creación de relaciones uno-a-uno o uno-a-varios entre diferentes tablas, además de combinar datos de diversas tablas para crear colecciones de resultados más complejos.

Para comprenderlo mejor, examina la figura 7-2, que muestra tres tablas vinculadas.



IDGenero	*NombreGenero*
1	Horror
2	Crimen
3	Romance

IDAutor	*NombreAutor*
1	Stephen King
2	Danielle Steele
3	Dennis Lehane
4	Michael Connelly
5	Nora Roberts

IDLibro	*NombreLibro*	*IDGenero*	*IDAutor*
1	The Shining	1	1
2	Shutter Island	2	3
3	Pet Sematary	1	1
4	The Overlook	1	4
5	Sisters	3	2
6	Safe Harbour	3	2
7	Trunk Music	1	4
8	Blood Brothers	3	5

Figura 7-2 Relaciones entre tablas

www.compumusicmania.com

La figura 7-2 muestra tres tablas, que contienen información sobre autores (tabla A), géneros (tabla G) y libros (tabla B), respectivamente. Las tablas G y B son claras: contienen una lista de géneros y nombres de autores, respectivamente, con cada registro identificado con una llave primaria única. La tabla B es un poco más compleja: cada libro de la tabla está vinculado con un género específico mediante la llave primaria perteneciente al género (tabla G) y a un autor mediante la llave primaria de autor (tabla A).

Siguiendo estas llaves hasta sus respectivas tablas fuente, es fácil identificar el autor y género de un libro específico. Por ejemplo, se puede ver que el libro "The Shining" fue escrito por "Stephen King" y pertenece al género "Horror". De manera similar, comenzando por el punto opuesto, se puede ver que el autor "Michael Connelly" ha escrito dos libros: "The Overlook" y "Trunk Music".

Relaciones como las que aparecen en la figura 7-2 son el fundamento de las bases de datos relacionales. Vincular tablas utilizando llaves externas es también más eficiente que su alternativa: crear una sola tabla que incluya hasta el nombre del perico, para almacenar toda la información, puede parecer conveniente a primera vista; pero actualizar una tabla así exige siempre una tarea manual (y propensa a errores) para localizar cada elemento de un valor específico y reemplazarlo con un nuevo valor. Dividir la información en tablas independientes y vincularlas con llaves externas asegura que una pieza de información aparece una, y sólo una vez, en la base de datos. Con esto se eliminan las redundancias, se simplifican las búsquedas (al estar localizables en un solo lugar), y se hace que la base sea más compacta y manejable.

Al proceso de afinar la base de datos definiendo y aplicando relaciones uno-a-uno y uno-a-varios entre las tablas que la integran se le conoce como *normalización de la base de datos*, y es una tarea clave que enfrentan los ingenieros informáticos cuando crean una nueva base de datos. En el proceso de normalización, el ingeniero también identifica relaciones cruzadas y dependencias incorrectas entre tablas; también optimiza la organización de los datos con el fin de que las consultas SQL se ejecuten a su máxima eficiencia. Existen *formas de normalización* disponibles que te ayudan a probar hasta qué punto está normalizada tu base de datos; estas normas proporcionan una guía útil para ayudarte a estar seguro de que el diseño de tu base tiene una estructura consistente y eficiente por igual.

Comprender las declaraciones SQL

El lenguaje estructurado de consultas, SQL, es el lenguaje estándar utilizado para comunicarse con la base de datos, añadir o cambiar registros y privilegios de usuario, y para realizar consultas. Casi todos los RDBMS comerciales utilizan en la actualidad este lenguaje, que se convirtió en estándar ANSI en 1989.

Las declaraciones SQL caen en alguna de las siguientes categorías:

- **Lenguaje de definición de datos (DLL, Data Definition Language)** Consta de declaraciones que definen la estructura y las relaciones de la base de datos y sus tablas. Por lo

www.compumusicmania.com

general, estas declaraciones se utilizan para crear, borrar y modificar bases de datos y tablas; especificar nombres de campos y tipos; así como establecer índices.

- **Lenguaje de manipulación de datos (DML, Data Manipulation Language)** Estas declaraciones alteran o extraen datos de una base; son utilizadas para añadir y borrar registros. También se ocupan para realizar consultas, recuperar registros de una tabla que coincidan con uno o más criterios especificados por el usuario, y unir tablas utilizando los campos que comparten.
- **Lenguaje para el control de datos (DCL, Data Control Language)** Estas declaraciones se utilizan para definir acceso a diferentes niveles y privilegios de seguridad en la base de datos. Utilizarás estas declaraciones para otorgar o negar privilegios a los usuarios, asignar funciones, cambiar claves de acceso, ver permisos y crear colecciones de resultados para proteger el acceso a los datos.

Los comandos SQL imitan la lengua inglesa, lo que facilita su aprendizaje. La sintaxis también es muy intuitiva: cada declaración SQL inicia con una "palabra de acción", como DELETE (borrar), INSERT (insertar), ALTER (alterar) o DESCRIBE (describir) y termina con un punto y coma (;). Los espacios en blanco, tabuladores y retornos de carro son ignorados. A continuación aparecen algunos ejemplos de declaraciones válidas SQL:

```
CREATE DATABASE biblioteca;
SELECT película FROM películas WHERE calificación > 4;
DELETE FROM autos WHERE año_de_manufactura < 1980;
```

La tabla 7-1 muestra la sintaxis de algunas declaraciones SQL comunes, con su respectiva explicación.

Declaración SQL	Lo que hace
`CREATE DATABASE` *`nombre de la base`*	Crea una nueva base de datos
`CREATE TABLE` *`nombre de la tabla`* `(`*`campo1`*`,` *`campo2`*`, ...)`	Crea una nueva tabla
`INSERT INTO` *`nombre de la tabla`* `(campo1, campo2, ...) VALUES (`*`valor1`*`,` *`valor2`*`, ...)`	Inserta un nuevo registro en una tabla con valores específicos
`UPDATE` *`nombre de la tabla`* `SET` *`campo1=valor1`*`,` *`campo2=valor2`*`, ... [WHERE condición]`	Actualiza registros en una tabla con nuevos valores
`DELETE FROM` *`nombre de la tabla`* `[WHERE condición]`	Borra registros de una tabla
`SELECT` *`campo1`*`,` *`campo2`*`, ... FROM` *`nombre de la tabla`* `[WHERE condición]`	Recupera los registros de una tabla que coinciden con los criterios de búsqueda
`RENAME` *`nombre de la tabla`* `TO` *`nuevo nombre de la tabla`*	Cambia el nombre de una tabla
`DROP TABLE` *`nombre de la tabla`*	Borra una tabla
`DROP DATABASE` *`nombre de la base`*	Borra una base de datos

Tabla 7-1 Declaraciones SQL comunes

www.compumusicmania.com

Pregunta al experto

P: **¿Qué tanto soporte tiene SQL?**

R: SQL es un estándar tanto ANSI como ISO, y está ampliamente respaldado por todas las bases de datos de compilación estándar SQL. Es decir, muchos proveedores de bases de datos cuentan también con "estándares" SQL extendidos con sus propias extensiones, para ofrecer a sus clientes un conjunto mejorado de características o un mejor rendimiento. Tales extensiones varían de proveedor a proveedor, y es posible que las declaraciones SQL que las utilizan no funcionen de la misma manera en todas las bases de datos. Por lo tanto, siempre es conveniente revisar la documentación de la base de datos con la que trabajes, para saber si ofrece extensiones particulares y cómo afectan las declaraciones SQL que escribas.

Prueba esto 7-1 Crear y alimentar una base de datos

Ahora que conoces los fundamentos, hagamos un ejercicio práctico que te debe familiarizar con el uso de las bases de datos y SQL. En esta sección utilizarás la línea de comando interactiva de cliente de MySQL para crear una base de datos y tablas, añadir y editar registros y generar colecciones de resultados que coincidan con varios criterios.

NOTA

A lo largo del siguiente ejercicio las palabras en negritas indican las instrucciones que debes escribir en la línea de comandos de MySQL. Las instrucciones, también llamadas "comandos", pueden escribirse en mayúsculas o minúsculas. Antes de comenzar con el ejercicio, asegúrate de haber instalado, configurado y probado la base de datos MySQL, de acuerdo con las instrucciones que aparecen en el apéndice A de este libro.

Comienza por iniciar la consola cliente de MySQL y conectarla a la base de datos con tu nombre de usuario y contraseña:

```
shell> mysql –u user -p
Contraword: ******
```

Si todo resulta bien, verás un mensaje de bienvenida y el indicador interactivo SQL, como el siguiente:

```
mysql>
```

Ahora puedes ingresar declaraciones SQL en este indicador. Las declaraciones serán transmitidas al servidor MySQL y ejecutadas en éste, y el resultado será mostrado en las líneas posteriores al indicador. Recuerda terminar cada declaración con un punto y coma.

(*continúa*)

www.compumusicmania.com

Crear la base de datos

Como todas las tablas se almacenan en una base de datos, el primer paso consiste en crear una de éstas, utilizando la declaración `CREATE DATABASE`:

```
mysql> CREATE DATABASE musica;
Query OK, 1 row affected (0.05 sec)
```

A continuación, selecciona esta base de datos recién creada como la que se usará por omisión para todos los futuros comandos; para ello utiliza la declaración `USE`:

```
mysql> USE musica;
Database changed
```

Añadir tablas

Una vez que has inicializado tu base de datos, es hora de añadirle algunas tablas. El comando SQL para realizar esta tarea es la declaración `CREATE TABLE`, que requiere un nombre de tabla y una descripción detallada de sus campos. He aquí un ejemplo:

```
mysql> CREATE TABLE artistas (
    -> artista_id INT(4) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> artista_nombre VARCHAR (50) NOT NULL,
    -> artista_pais CHAR (2) NOT NULL
    -> );
Query OK, 0 rows affected (0.07 sec)
```

Esta declaración crea una tabla llamada *artistas* con tres campos: *artista_id*, *artista_nombre* y *artista_pais*. Advierte que cada nombre de campo va seguido por una *declaración de tipo*; esta declaración identifica el tipo de dato que almacenará el campo, ya sea una cadena de caracteres, numérico, temporal o booleano. MySQL soporta diferentes tipos de datos y los más importantes están resumidos en la tabla 7-2.

En el anterior ejemplo, hay otras pocas indicaciones (*modificadores*) adicionales que se establecen en la tabla:

- El modificador `NOT NULL` asegura que el campo no pueda recibir valores NULL después de cada definición de campo.
- El modificador `PRIMARY KEY` marca el campo correspondiente a la llave primaria de la tabla.
- El modificador `AUTO_INCREMENT`, que sólo es aplicable a campos numéricos, le indica a MySQL que genere automáticamente valores para este campo cada vez que se inserta un nuevo registro en la tabla, incrementando en 1 el valor previo.

www.compumusicmania.com

Tipo de campo	Descripción
INT	Tipo numérico que puede aceptar un rango de valores de –2147483648 a 2147483647
DECIMAL	Tipo numérico que soporta valores de punto flotante y decimales
DATE	Campo de fecha con el formato AAAA-MM-DD
TIME	Campo de tiempo en formato HH:MM:SS
DATETIME	Tipo que combina fecha y hora en formato AAAA-MM-DD HH:MM:SS
YEAR	Campo específico para años que acepta un rango de 1901 a 2155; en formatos AAAA o AA
TIMESTAMP	Tipo sello cronológico en formato AAAAMMDDHHMMSS
CHAR	Tipo cadena de caracteres con un tamaño máximo de 255 caracteres y longitud fija
VARCHAR	Tipo cadena de caracteres con un tamaño máximo de 255 caracteres y longitud variable
TEXT	Tipo cadena de caracteres con un tamaño máximo de 65535 caracteres
BLOB	Tipo binario para datos variables
ENUM	Tipo cadena de caracteres que acepta un valor de una lista de posibles valores definida con anterioridad
SET	Tipo cadena de caracteres que acepta cero o más valores de un conjunto de posibles valores definido con anterioridad

Tabla 7-2 Tipos de datos MySQL

Ahora sigamos adelante para crear otras dos tablas utilizando estas declaraciones SQL:

```
mysql> CREATE TABLE ratings (
    -> rating_id INT(2) NOT NULL PRIMARY KEY,
    -> rating_nombre VARCHAR (50) NOT NULL
    -> );
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE canciones (
    -> cancion_id INT(4) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> cancion_titulo VARCHAR(100) NOT NULL,
    -> ex_cancion_artista INT(4) NOT NULL,
    -> ex_cancion_rating INT(2) NOT NULL
    -> );
Query OK, 0 rows affected (0.05 sec)
```

(*continúa*)

Añadir registros

Añadir registros a la tabla es tan sencillo como invocar la declaración INSERT con los valores apropiados. He aquí un ejemplo, que añade un registro a la tabla *artistas* especificando valores para los campos *artista_id* y *artista_nombre*:

```
mysql> INSERT INTO artistas (artista_id, artista_nombre, artista_pais)
    -> VALUES ('1', 'Aerosmith', 'US');
Query OK, 1 row affected (0.00 sec)
```

Recordarás, de la sección anterior, que el campo *artista_id* fue marcado con el modificador AUTO_INCREMENT. Ésta es una extensión MySQL sobre el SQL estándar, que indica a MySQL que asigne un valor automáticamente a este campo en caso de quedar sin especificación. Para verlo en acción trata de añadir otro registro utilizando la siguiente declaración:

```
mysql> INSERT INTO artistas (artista_nombre, artista_pais)
    -> VALUES ('Abba', 'SE');
Query OK, 1 row affected (0.00 sec)
```

De manera similar, añade algunos registros a la tabla *ratings*:

```
mysql> INSERT INTO ratings (rating_id, rating_nombre) VALUES (4, 'Bueno');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO ratings (rating_id, rating_nombre) VALUES (5, 'Excelente');
Query OK, 1 row affected (0.00 sec)
```

Y algunos a la tabla *canciones*:

```
mysql> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
    ex_cancion_rating) -> VALUES ('Janie\'s Got a Gun', 1, 4);
Query OK, 1 row affected (0.04 sec)
mysql> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
    ex_cancion_rating) -> VALUES ('Crazy', 1, 5);
Query OK, 1 row affected (0.00 sec)
```

Advierte que los registros en la tabla *canciones* están vinculados a los registros de la tabla *artistas* y *ratings* por llaves externas. En la siguiente sección verás en acción este tipo de relaciones por llave externa.

NOTA

El archivo de código de este libro tiene una lista completa de declaraciones SQL INSERT para llenar las tres tablas utilizadas en este ejercicio. Ejecuta estas declaraciones y termina de construir las tablas antes de pasar a la siguiente sección.

www.compumusicmania.com

Ejecutar consultas

Una vez que los datos están en la base, es hora de hacer algo con ellos. SQL te permite buscar registros que coinciden con ciertos criterios específicos utilizando la declaración `SELECT`. He aquí un ejemplo que regresa todos los registros de la tabla *artistas*:

```
mysql> SELECT artista_id, artista_nombre FROM artistas;
+------------+----------------+
| artista_id | artista_nombre |
+------------+----------------+
|          1 | Aerosmith      |
|          2 | Abba           |
|          3 | Timbaland      |
|          4 | Take That      |
|          5 | Girls Aloud    |
|          6 | Cubanismo      |
+------------+----------------+
6 rows in set (0.00 sec)
```

En casi todos los casos querrás añadir filtros a tu consulta, para reducir el tamaño de la colección de resultados y asegurar que contiene sólo los registros que coinciden con ciertos criterios. Esto se hace añadiendo la cláusula `WHERE` a la declaración `SELECT` junto con una o más expresiones condicionales. He aquí un ejemplo que muestra sólo los artistas de Estados Unidos:

```
mysql> SELECT artista_id, artista_nombre FROM artistas
    -> WHERE artista_pais = 'US';
+------------+----------------+
| artista_id | artista_nombre |
+------------+----------------+
|          1 | Aerosmith      |
|          3 | Timbaland      |
+------------+----------------+
2 rows in set (0.00 sec)
```

Todos los operadores de comparación estándar con los que ya estás familiarizado por PHP tienen soporte en SQL. El ejemplo anterior utiliza el operador de igualdad (=); el siguiente ejemplo muestra el funcionamiento del operador "mayor que o igual a" (>=), que regresa una lista con las canciones con rating 4 o superior:

```
mysql> SELECT cancion_titulo, ex_cancion_rating FROM canciones
    -> WHERE ex_cancion_rating >= 4;
+-------------------+-------------------+
| canción_título    | ex_canción_rating |
+-------------------+-------------------+
| Janie's Got A Gun |                 4 |
| Crazy             |                 5 |
| En Las Delicious  |                 5 |
```

(*continúa*)

www.compumusicmania.com

```
| Pray               |                  4 |
| Apologize          |                  4 |
| SOS                |                  4 |
| Dancing Queen      |                  4 |
+--------------------+--------------------+
7 rows in set (0.00 sec)
```

Puedes combinar expresiones condicionales utilizando los operadores lógicos `AND`, `OR` y `NOT` (tal y como se hace en una declaración condicional regular PHP). He aquí un ejemplo, que presenta una lista con los artistas de Estados Unidos y el Reino Unido:

```
mysql> SELECT artista_nombre, artista_pais FROM artistas
    -> WHERE artista_pais = 'US'
    -> OR artista_pais = 'UK';
+-----------------+--------------+
| artista_nombre | artista_país |
+-----------------+--------------+
| Aerosmith      | US           |
| Timbaland      | US           |
| Take That      | UK           |
| Girls Aloud    | UK           |
+-----------------+--------------+
4 rows in set (0.02 sec)
```

Ordenar y limitar colecciones de resultados

Si quieres ver los datos de una tabla ordenados por un campo específico, SQL cuenta con la cláusula `ORDER BY`. Te permite definir el nombre del campo sobre el que desees basar el orden del resultado y su dirección (ascendente o descendente).

Por ejemplo, para ver la lista de las canciones en orden alfabético, utiliza la siguiente declaración SQL:

```
mysql> SELECT cancion_titulo FROM canciones
    -> ORDER BY cancion_titulo;
+---------------------------+
| canción_título            |
+---------------------------+
| Another Crack In My Heart |
| Apologize                 |
| Babe                      |
| Crazy                     |
| Dancing Queen             |
| En Las Delicious          |
| Gimme Gimme Gimme         |
| Janie's Got A Gun         |
| Pray                      |
| SOS                       |
```

www.computrucionmania.com

```
| Sure                      |
| Voulez Vous               |
+---------------------------+
12 rows in set (0.04 sec)
```

Para invertir el orden de la lista, añade el modificador `DESC`, como se muestra a continuación:

```
mysql> SELECT cancion_titulo FROM canciones
    -> ORDER BY cancion_titulo DESC;
+---------------------------+
| canción_título            |
+---------------------------+
| Voulez Vous               |
| Sure                      |
| SOS                       |
| Pray                      |
| Janie's Got A Gun         |
| Gimme Gimme Gimme         |
| En Las Delicious          |
| Dancing Queen             |
| Crazy                     |
| Babe                      |
| Apologize                 |
| Another Crack In My Heart |
+---------------------------+
12 rows in set (0.00 sec)
```

SQL también te permite limitar la cantidad de registros que aparecen en la colección de resultados con la palabra clave `LIMIT`, que acepta dos parámetros: la posición del registro para comenzar (a partir de 0) y la cantidad de registros que aparecerán. Por ejemplo, para mostrar las filas 4-9 (inclusivas) en una colección de resultados, utiliza la siguiente declaración:

```
mysql> SELECT cancion_titulo FROM canciones
    -> ORDER BY cancion_titulo
    -> LIMIT 3,6;
+-------------------+
| canción_título    |
+-------------------+
| Crazy             |
| Dancing Queen     |
| En Las Delicious  |
| Gimme Gimme Gimme |
| Janie's Got A Gun |
| Pray              |
+-------------------+
5 rows in set (0.00 sec)
```

(*continúa*)

www.compumusicmania.com

Utilizar comodines

La declaración `SELECT` de SQL también da soporte a la cláusula `LIKE`, que puede utilizarse para realizar búsquedas dentro de campos de texto con el uso de comodines. Hay dos tipos de comodines aceptados en la cláusula `LIKE`: el signo de porcentaje (`%`), que es utilizado para representar cero o más apariciones de cierto carácter y el guión bajo (_), cuyo uso significa exactamente una aparición de cierto carácter.

El siguiente ejemplo muestra la cláusula `LIKE` en acción, que busca títulos de canciones que contengan el carácter 'g':

```
mysql> SELECT cancion_id, cancion_titulo FROM canciones
    -> WHERE cancion_titulo LIKE '%g%';

+------------+-------------------+
| canción_id | canción_título    |
+------------+-------------------+
|          1 | Janie's Got A Gun |
|          7 | Apologize         |
|          8 | Gimme Gimme Gimme |
|         10 | Dancing Queen     |
+------------+-------------------+
4 rows in set (0.00 sec)
```

Fusionar tablas

Hasta ahora, todas las consultas que has visto se concentran en una sola tabla. Pero SQL también te permite realizar búsquedas entre dos o más tablas al mismo tiempo y combinar el resultado en una sola colección de resultados. Esta tarea lleva el nombre técnico de *fusión*, pues "fusiona" diferentes tablas que comparten campos entre sí (las llaves externas) para crear nuevas vistas de los datos.

TIP

Cuando fusiones tablas, usa un prefijo para cada nombre de campo con el nombre de la tabla a la que pertenece, con el fin de evitar confusiones en caso de que varios campos en diferentes tablas tengan el mismo nombre.

He aquí un ejemplo de fusión entre las tablas *canciones* y *artistas* utilizando el campo común *artista_id* (la cláusula `WHERE` es utilizada para transformar los campos comunes entre sí):

```
mysql> SELECT cancion_id, cancion_titulo, artista_nombre FROM canciones,
artistas
    -> WHERE canciones.ex_cancion_artista = artistas.artista_id;
```

www.compumusicmania.com

```
+------------+--------------------------+----------------+
| canción_id | canción_título           | artista_nombre |
+------------+--------------------------+----------------+
|          1 | Janie's Got A Gun        | Aerosmith      |
|          2 | Crazy                    | Aerosmith      |
|          8 | Gimme Gimme Gimme        | Abba           |
|          9 | SOS                      | Abba           |
|         10 | Dancing Queen            | Abba           |
|         11 | Voulez Vous              | Abba           |
|          7 | Apologize                | Timbaland      |
|          4 | Sure                     | Take That      |
|          5 | Pray                     | Take That      |
|          6 | Another Crack In My Heart | Take That      |
|         12 | Babe                     | Take That      |
|          3 | En Las Delicious         | Cubanismo      |
+------------+--------------------------+----------------+
12 rows in set (0.00 sec)
```

Y a continuación tenemos un ejemplo que fusiona las tres tablas y luego filtra la colección de resultados aún más para incluir sólo las canciones con rating 4 o superior de artistas que no sean de Estados Unidos:

```
mysql> SELECT cancion_titulo, artista_nombre, rating_nombre
    -> FROM canciones, artistas, ratings
    -> WHERE canciones.ex_cancion_artista = artistas.artista_id
    -> AND canciones.ex_cancion_rating = ratings.rating_id
    -> AND ratings.rating_id >= 4
    -> AND artistas.artista_pais != 'US';
+------------------+----------------+---------------+
| canción_título   | artista_nombre | rating_nombre |
+------------------+----------------+---------------+
| En Las Delicious | Cubanismo      | Excelente     |
| Pray             | Take That      | Buena         |
| SOS              | Abba           | Buena         |
| Dancing Queen    | Abba           | Buena         |
+------------------+----------------+---------------+
4 rows in set (0.02 sec)
```

Modificar y eliminar registros

Así como insertas registros en una tabla (con el comando `INSERT`), también es posible borrarlos con la declaración `DELETE`. Por lo general, seleccionarás un subgrupo específico de filas para ser borradas añadiendo la cláusula `WHERE` a la declaración `DELETE`, como se muestra en el siguiente ejemplo, que elimina todas las canciones con un rating igual a o menor que 3:

```
mysql> DELETE FROM canciones
    -> WHERE ex_cancion_rating <= 3;
Query OK, 5 rows affected (0.02 sec)
```

(*continúa*)

www.compumusicmania.com

También existe una declaración UPDATE, que puede utilizarse para cambiar el contenido de un registro; esta declaración también acepta la cláusula WHERE, de manera que puedes aplicar los cambios sólo en los registros que coinciden con cierto criterio. Examina el siguiente ejemplo, que cambia el rating 'Excelente' por 'Fantástico':

```
mysql> UPDATE ratings SET rating_nombre = 'Fantástico'
    -> WHERE rating_nombre = 'Excelente';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Puedes modificar varios campos separándolos con comas. He aquí un ejemplo que actualiza el registro de una canción en particular en la tabla *canciones*, modificando tanto el título como el rating:

```
mysql> UPDATE ratings SET cancion_titulo = 'Waterloo',
    -> ex_cancion_rating = 5
    -> WHERE cancion_id = 9;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Pregunta al experto

P: Estoy utilizando la base de datos ____ y no soporta el tipo de campo ____. ¿Qué hago ahora?

R: Diferentes bases de datos tienen distinta sintaxis para tipos de datos de los campos. Por ejemplo, MySQL llama a los campos enteros INT, mientras que SQLite llama a los mismos campos INTEGER. De cualquier manera, casi todas las bases de datos soportan (por lo menos) tipos de datos para valores enteros, de punto flotante, cadenas de caracteres, binarios y NULL. Lo único que necesitas es revisar la documentación de tu base de datos y encontrar la sintaxis para utilizar correctamente los tipos de datos en tus declaraciones SQL.

P: ¿Por qué necesito incluir todos los campos requeridos en la declaración SELECT? ¿No puedo utilizar el comodín * para recuperar todos los campos disponibles?

R: SQL sí soporta el asterisco (*) como carácter comodín para representar "todos los campos de la tabla", pero es preferible siempre designar explícitamente los campos que quieras ver en la colección de resultados. Esto permite que la aplicación sobreviva a los cambios estructurales en las tablas (o la tabla), además de que es más eficiente en el uso de memoria porque la colección de resultados no contendrá datos irrelevantes o no deseados.

www.compumusicamania.com

Utilizar la extensión MySQLi de PHP

Como ya se explicó, PHP permite que los desarrolladores interactúen con la base de datos de dos maneras: utilizando extensiones personalizadas específicas de la base, o la extensión neutral (PDO). Mientras que estas últimas son más portátiles, muchos desarrolladores encuentran preferible utilizar las extensiones nativas de la base de datos con la que trabajan, sobre todo cuando ofrecen mejor rendimiento o más características que la versión PDO.

De los diferentes motores de base de datos soportados por PHP, el más popular es MySQL. No resulta difícil entender el porqué: tanto PHP como MySQL son proyectos de código libre, y al utilizarlos juntos, los desarrolladores obtienen beneficios de los grandes ahorros en costos de licencias en comparación con las opciones comerciales. Históricamente, además, PHP ha ofrecido soporte extraoficial para MySQL desde la versión 3, y tiene sentido aprovechar el tremendo esfuerzo que han realizado los desarrolladores de PHP y MySQL para asegurar que los dos paquetes trabajen juntos sin problemas.

Además de dar soporte a MySQL mediante las extensiones de objetos de datos de PHP (que se abordarán en la siguiente sección), PHP también incluye una extensión MySQL personalizada que lleva el nombre de MySQL Mejorado (MySQLi Improved). Esta extensión brinda beneficios de velocidad y de características superiores a la versión PDO y es una buena opción para desarrollar proyectos específicos con MySQL. Las siguientes secciones abordan esta extensión con gran detalle.

Recuperar datos

En la sección anterior creaste una base de datos y utilizaste una consulta `SELECT` para recuperar una colección de resultados de ella. Ahora harás lo mismo utilizando PHP, como en el siguiente script:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user" "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// itera sobre colección de resultados
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysqli ->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_array()) {
      echo $row[0] . ":" . $row[1] . "\n";
    }
```

www.compumusicmania.com

```
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

La figura 7-3 muestra cómo se debe ver el resultado del script.

Como puedes ver, utilizar PHP para obtener datos de una base requiere varios pasos, que se describen a continuación:

1. Con el fin de establecer comunicación con el servidor que contiene la base de datos MySQL, primero necesitas abrir una conexión con el mismo. Toda la comunicación entre PHP y el servidor de base de datos se realiza a través de esta conexión.

Para inicializar esta conexión, inicializa un objeto de la clase MySQLi y pasa cuatro argumentos al constructor del objeto: el nombre del servidor anfitrión de MySQL al que intentas conectarte, un nombre y una contraeña válidos para obtener el acceso necesario, y el nombre de la base de datos que quieres utilizar.

Dando por hecho que la conexión se estableció con éxito, este objeto representa la conexión con la base de datos para todas las futuras operaciones y expone los métodos para realizar consultas, búsquedas y para procesar las colecciones de resultados. Si el intento de conexión fracasó, el objeto será falso; entonces es posible obtener un mensaje de error que explique las razones de la falla; esto se lleva a cabo invocando la función `mysqli_connect_error()`.

Figura 7-3 Registros recuperados de una base de datos MySQL con PHP

www.compumusicamania.com

TIP

Si los dos servidores, el de base de datos y el Web, se ejecutan en la misma computadora, puedes utilizar `localhost` como nombre de servidor.

2. El siguiente paso consiste en crear y ejecutar la consulta SQL. Esto se realiza invocando el método `query()` para el objeto de MySQLi y transmitiéndole la consulta que se va a ejecutar. Si la consulta no tuvo éxito, el método regresa un valor booleano falso, y un mensaje de error que explica la causa de la falla es almacenado en la propiedad `'error'` del objeto MySQLi.

3. Si, por otra parte, la consulta se ejecuta con éxito y regresa uno o más registros, el valor de retorno del método `query()` se convierte en otro objeto, este último una instancia de la clase MySQLi_Result. Este objeto representa la colección de resultados regresada por la consulta, y expone varios métodos para procesar los registros individuales en la colección de resultados.

   Uno de esos métodos es `fetch_array()`. Cada vez que se invoca, regresa el siguiente registro de la colección de resultados como una matriz. Esto hace que el método `fetch_array()` se acople bien con el uso de los bucles `while` y `for`. El contador del bucle determina cuántas veces debe ejecutarse; este resultado se obtiene de la propiedad `'num_rows'` del objeto MySQLi_Result, que almacena el número de filas regresadas por la consulta. Los campos individuales del registro pueden accesarse como elementos de una matriz, utilizando ya sea el índice del campo o su nombre.

PRECAUCIÓN

La propiedad `'num_rows'` sólo tiene significado cuando se utiliza con consultas que regresan datos, como `SELECT`; no debe utilizarse con los consultas `INSERT`, `UPDATE` o `DELETE`.

4. Cada colección de resultados regresada después de una consulta ocupa cierta cantidad de memoria. Así, una vez que el resultado ha sido procesado, es buena idea destruir el objeto MySQLi_Result, para liberar la memoria que utiliza; esto se realiza con el método `close()`. Y una vez que has terminado de trabajar con la base de datos, también es buena idea destruir el objeto principal MySQLi de manera similar, invocando el método `close()`.

Regresar registros como matrices y objetos

El ejemplo anterior mostró un método para procesar la colección de resultados: el método `fetch_array()` del objeto MySQLi_Result. Este método regresa cada registro de la colección de resultados como una matriz que contiene llaves indexadas tanto numéricamente como por cadenas de caracteres; esto ofrece a los desarrolladores la conveniencia de hacer referencia a campos individuales de cada registro ya sea por índice o por nombre de campo.

www.compumusicmania.com

El ejemplo anterior mostró cómo recuperar campos individuales utilizando el número de índice. El siguiente ejemplo, equivalente al anterior, realiza la misma tarea utilizando nombres de campo:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// reitera sobre colección de resultados
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysqli ->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_array()) {
      echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";
    }
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Sin embargo, existe un tercer método para recuperar registros: como objetos, utilizando el método `fetch_object()`. Aquí, cada registro se representa como un objeto y los campos de un registro se representan como propiedades del objeto. Después, los campos individuales pueden ser accesados utilizando la notación estándar `$objeto->propiedad`. El siguiente ejemplo ilustra este procedimiento:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// reitera sobre colección de resultados
```

www.compumusicamania.com

```
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysqli->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_object()) {
      echo $row->artista_id . ":" . $row->artista_nombre . "\n";
    }
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Añadir y modificar datos

Es igual de sencillo ejecutar un consulta que cambie los datos en la base, ya sea insertar (`INSERT`), actualizar (`UPDATE`) o borrar (`DELETE`). El siguiente ejemplo lo muestra, añadiendo un nuevo registro a la tabla *artistas*:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// añade un nuevo registro
// datos de salida: " Nuevo artista con id: 7 ha sido añadido. "
$sql = "INSERT INTO artistas (artista_nombre, artista_pais) VALUES
('Kylie Minogue', 'AU')";
if ($mysqli->query($sql)=== true) {
  echo 'Nuevo artista con id: ' . $mysqli->insert_id . 'ha sido
  añadido.';
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Como verás, esto no es muy diferente de la necesidad de aplicar una consulta SELECT al programar. De hecho, es un poco más sencillo porque no hay una colección de datos para procesar; todo lo que se necesita es probar el valor regresado del método query() y verificar si la consulta se ejecutó correctamente o no. Advierte también el uso de la nueva propiedad 'insert_id', que regresa el ID generado por la última consulta INSERT (sólo es útil si la tabla en la cual se aplicó INSERT contiene un campo que se incrementa automáticamente).

¿Qué hay de incrementar un registro existente? Lo único que necesitas es cambiar la línea de comando SQL:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// añade un nuevo registro
// datos de salida: "1 fila(s) actualizadas."
$sql = "UPDATE artistas SET artista_nombre = 'Eminem', artista_pais =
'US' WHERE artista_id = 7;
if ($mysqli->query($sql)=== true) {
  echo $mysqli->affected_rows . ' fila(s) actualizadas.';
} else {
  echo "ERROR: No fue posible ejecutar: $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Cuando ejecutas UPDATE o DELETE, el número de filas afectadas por la declaración se almacenarán en la propiedad 'affected_rows' del objeto MySQLi. El ejemplo siguiente muestra el uso de esta propiedad.

Utilizar declaraciones preparadas

En caso de que necesites ejecutar un consulta particular varias veces con diferentes valores (por ejemplo, una serie de declaraciones INSERT), el servidor MySQL da soporte a *declaraciones preparadas*, que ofrecen medios más eficientes para completar esta tarea en lugar de invocar repetidamente el método $mysqli->query().

En esencia, una declaración preparada es una consulta SQL modelo que contiene variables prealmacenadas para los valores que serán insertados o modificados. Esta declaración es almacenada en el servidor de base de datos y la invoca todas las veces que sea necesario; las variables almacenadas previamente se reemplazan con los nuevos valores cada vez que se ejecutan. Dado que la declaración está almacenada en el servidor de base de datos, una decla-

www.compumusicamania.com

ración preparada suele ser más rápida para realizar operaciones por lote que implican ejecutar la misma consulta SQL una y otra vez con diferentes valores.

Pregunta al experto

P: ¿Por qué las declaraciones preparadas ofrecen mejor rendimiento?

R: En condiciones normales, cada vez que se ejecuta una declaración SQL sobre el servidor de base de datos, éste debe segmentar el código SQL y verificar su sintaxis y estructura antes de permitir su ejecución. Con una declaración preparada, la declaración SQL está almacenada temporalmente en el servidor de base de datos y, por lo mismo, sólo necesita segmentarse y validarse una sola vez. Más aún, cada vez que se ejecuta la declaración, sólo los valores que cambian necesitan transmitirse al servidor, en lugar de enviar la declaración completa.

Para ver una declaración preparada en acción, examina el siguiente script, que inserta varias canciones en la base de datos utilizando precisamente una declaración preparada con la extensión MySQLi de PHP:

```
<?php
// define valores a ser insertados
$canciones = array(
  array('Patience', 4, 3),
  array('Beautiful World', 4, 4),
  array('Shine', 4, 4),
  array('Hold On', 4, 3),
);

// intenta establecer la conexión con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// prepara el consulta modelo
// lo ejecuta varias veces
$sql = "INSERT INTO canciones (cancion_titulo, ex_cancion_artista, ex_
cancion_rating) VALUES (?, ?, ?)";
if ($stmt = $mysqli->prepare($sql)) {
  foreach ($canciones as $s) {
    $stmt->bind_param('sii', $s[0], $s[1], $s[2]);
    if ($stmt->execute()) {
     echo "Nueva canción con id: " . mysqli->insert_id . "ha sido añadida.\n";
    } else {
     echo "ERROR: No fue posible ejecutar consulta: $sql. " . $mysqli->error;
     }
  }
```

```
} else {
  echo "ERROR: No fue posible preparar consulta: $sql. " . $mysqli
->error;
}

// cierra conexión
$mysqli->close();
?>
```

Como se aprecia en este ejemplo, el uso de una declaración preparada implica un proceso diferente al que has visto en ejemplos previos.

Para utilizar una declaración preparada, primero es necesario definir la cadena de la consulta que será ejecutada varias veces. Esta cadena de consulta por lo regular contendrá uno o varios contenedores, representados por los signos de interrogación (`?`). Estos contenedores serán sustituidos por los nuevos valores cada vez que se ejecute la declaración. Entonces se transmite la cadena de consulta con sus contenedores al método `prepare()` del objeto MySQLi, que verifica el comando SQL en busca de errores y regresa un objeto MySQLi_Stmt que representa la declaración preparada.

Ahora, ejecutar la declaración preparada es cuestión de realizar dos acciones, por lo general con un bucle:

1. *Unir los valores a la declaración preparada*. Los valores que van a ser interpolados en la declaración necesitan *unirse* a sus contenedores con el método `bind_param()` del objeto MySQLi_Stmt. El primer argumento para este método debe ser una cadena de secuencia ordenada que indique los tipos de datos de los valores que serán interpolados (*s* para una cadena, *i* para un entero, *d* para un número de doble precisión); este argumento es seguido por los nuevos valores. En el ejemplo anterior, la cadena `'sii'` indica que los valores que se interpolarán en la declaración preparada serán, en secuencia, un tipo cadena de caracteres (el título de la canción), un tipo entero (la llave externa del artista) y otro tipo entero (la llave externa para el rating).
2. *Ejecutar la declaración preparada*. Una vez que se han unido los valores a sus contenedores, se ejecuta la declaración preparada invocando el método `execute()` del objeto MySQLi_Stmt. Este método reemplaza los contenedores en la declaración preparada con los nuevos valores y se ejecuta en el servidor.

De esta manera, el uso de una declaración preparada ofrece beneficios de rendimiento cuando necesitas ejecutar la misma declaración varias veces, y sólo los valores cambian en cada ejecución del comando. Las bases de datos más populares, incluyendo MySQL, PostgreSQL, Oracle e InterBase, soportan declaraciones preparadas, y esta característica debe ser utilizada cuando esté disponible para realizar tus operaciones SQL por lote de manera más eficiente.

www.compumusicmania.com

Manejo de errores

Si tu código de base de datos no funciona como esperabas, no te preocupes; la extensión MySQLi contiene una gran cantidad de funciones que te pueden decir la causa. Ya las has visto en acción en diferentes lugares de los ejemplos anteriores, pero a continuación aparece una lista completa:

- La propiedad '`error`' del objeto MySQLi guarda el último mensaje de error generado por el servidor de base de datos.
- La propiedad '`errno`' del objeto MySQLi guarda el último error de código regresado por el servidor de base de datos.
- La función `mysqli_connect_error()` regresa el mensaje de error generado por el último intento (fallido) de conexión.
- La función `mysqli_connect_errno()` regresa el error de código generado por el último intento (fallido) de conexión.

TIP

Para hacer tu aplicación más robusta contra errores, y para localizarlos y resolverlos con mayor facilidad, es una buena idea utilizar estas funciones para manejo de errores con toda libertad dentro de tu código.

Prueba esto 7-2 Añadir empleados a una base de datos

Ahora que ya conoces el uso básico de las extensiones MySQLi de PHP, practicarás lo que aprendiste en la sección anterior en una aplicación "real". La aplicación es un formulario Web que permite al usuario insertar nombres de empleados y sus puestos en una base de datos para empleados basada en MySQL. PHP valida y limpia los valores ingresados en el formulario; después se transforman en registros de la base con el uso de la extensión MySQLi de PHP.

Para empezar a construir esta aplicación, primero ejecuta el programa cliente de línea de comando MySQL y crea una base de datos vacía.

```
mysql> CREATE DATABASE empleados;
Query OK, 1 row affected (0.20 sec)
```

Después crea una tabla que contenga los registros de los empleados:

```
mysql> USE empleados;
Database changed
mysql> CREATE TABLE empleados (
```

(*continúa*)

www.compumusicamania.com

```
    -> id INT (4) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(255) NOT NULL,
    -> puesto VARCHAR(255) NOT NULL
    -> );
Query OK, 0 row affected (0.20 sec)
```

¿Todo listo? Lo siguiente es construir un formulario Web que acepte los datos del empleado (en este caso, su nombre y puesto). Al momento de enviarlo, el procesador del formulario verificará los datos de entrada y, de ser válidos, generará una consulta `INSERT` para insertarlos en la tabla creada en el paso anterior.

He aquí el script (*empleados.php*):

```
<!DOCTYPE html PUBLIC "-// W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-2: Añadir empleados a una base de datos</title>
    <style type="text/css">
    div#message{
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-2: Añadir empleados a una base de datos</h2>
    <h3>Añade un Nuevo Empleado</h3>
<?php
    // si el formulario ha sido enviado
    // procesa los datos del formulario
    if (isset($_POST['submit'])) {
      // intenta conectarse con la base de datos MySQL
      $mysqli = new mysqli("localhost", "usuario" "contra", "empleados");
      if ($mysqli === false) {
        die("ERROR: No fue posible conectarse con la base de datos. " .
mysqli_connect_error());
      }

      // abre bloque de mensaje
      echo '<div id="message">';

      // recupera y verifica los valores de entrada
      $inputError = false;
      if (empty($_POST['emp_nombre'])) {
        echo 'ERROR: Por favor ingrese un nombre de empleado válido';
```

www.compumusicamania.com

```
        $inputError = true;
      } else {
        $nombre = $mysqli->escape_string($_POST['emp_nombre']);
      }

      if ($inputError != true && empty($_POST['emp_puesto'])) {
        echo 'ERROR: Por favor ingrese un puesto válido';
        $inputError = true;
      } else {
        $puesto = $mysqli->escape_string($_POST['emp_puesto']);
      }

      // añade valores a la base de datos utilizando el consulta INSERT
      if ($inputError != true) {
      $sql = "INSERT INTO empleados (nombre, puesto)
                VALUES ('$nombre', '$puesto')";
        if ($mysqli->query($sql) === true) {
          echo 'Nuevo registro de empleado añadido con ID: ' . $mysqli
          ->insert_id;
        } else {
          echo "ERROR: No fue posible ejecutar el consulta: $sql. " .
          $mysqli->error;
        }
      }

      // cierra bloque de mensaje
      echo '</div>';

      // cierra conexión
      $mysqli->close();
    }
?>
    </div>

    <form action="empleados.php" method="POST">
      Nombre del empleado: <br />
      <input type="text" name="emp_nombre" size="40" />
      <p />
      Puesto del empleado: <br />
      <input type="text" name="emp_puesto" size="40" />
      <p />
      <input type="sumbit" name="submit" value="Enviar" />
    </form>

  </body>
</html>
```

Cuando se invoca este script, genera un formulario Web con campos para el nombre y puesto de los empleados. La figura 7-4 muestra cómo luce el formulario inicial.

(*continúa*)

www.compumusicmania.com

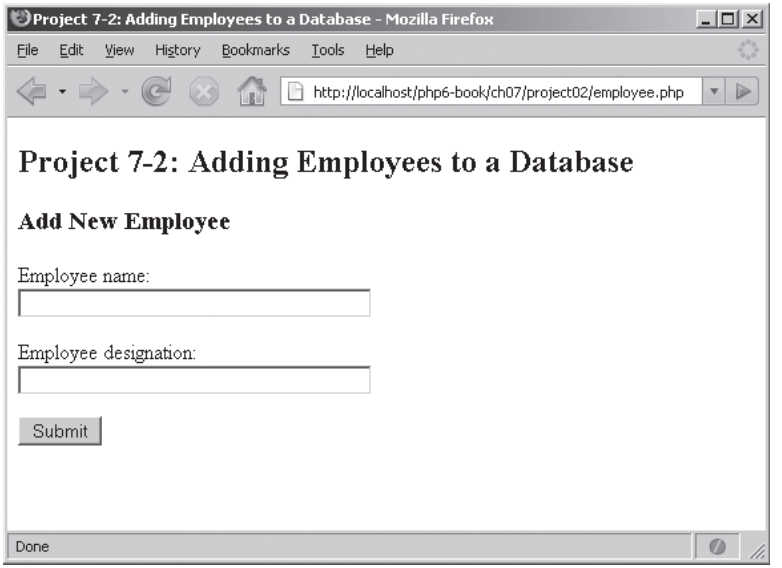


Figura 7-4 Formulario Web para insertar nuevos empleados

Cuando se envía el formulario, el script procede a inicializar un nuevo objeto MySQLi y abre la conexión para el servidor de bases de datos MySQL. Luego verifica los campos del formulario para asegurarse de que contienen valores de entrada válidos. De ser así, cada uno de estos valores es "limpiado" utilizando el método `escape_string()` del objeto MySQLi, que automáticamente limpia los caracteres especiales en los datos de entrada como preludio para insertarlos en la base de datos y luego interpola en la consulta INSERT, que se ejecuta con el método `query()` del objeto para guardar los valores en la base de datos. Los errores, si existen, se muestran utilizando la propiedad error del objeto MySQLi.

La figura 7-5 muestra los datos de salida cuando se agrega un registro correctamente y la figura 7-6 muestra los datos de salida cuando falla la validación en un campo de entrada.

Eso fue muy fácil. Ahora, ¿qué tal si rehacemos el script para que, además de permitir la inserción de nuevos empleados a la base de datos, también muestre los registros existentes en la misma? No es muy difícil; se añade otra invocación al método `query()`, esta vez con una consulta `SELECT`, y se procesa la colección de resultados utilizando un bucle.

He aquí el código modificado:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-2: Añadir empleados a una base de datos</title>
    <style type="text/css">
    div#message {
```

www.compumusicmania.com

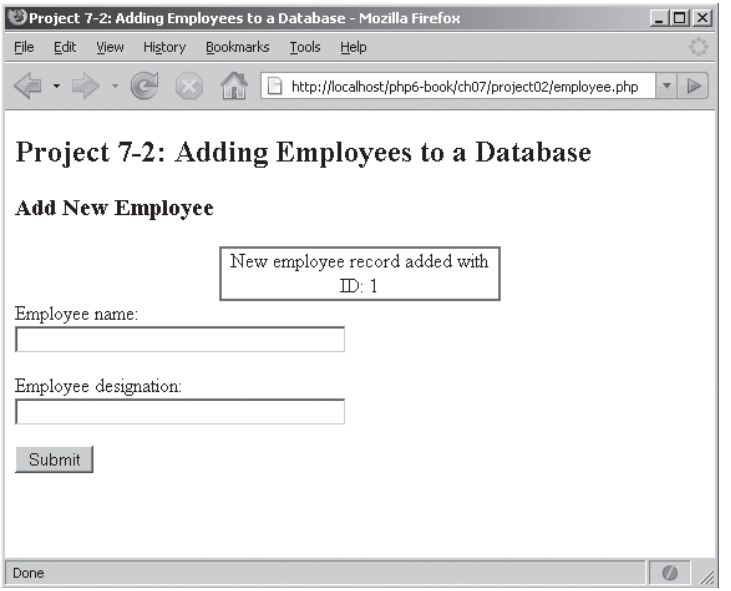


Figura 7-5 El resultado de una inserción correcta de un nuevo empleado en la base de datos

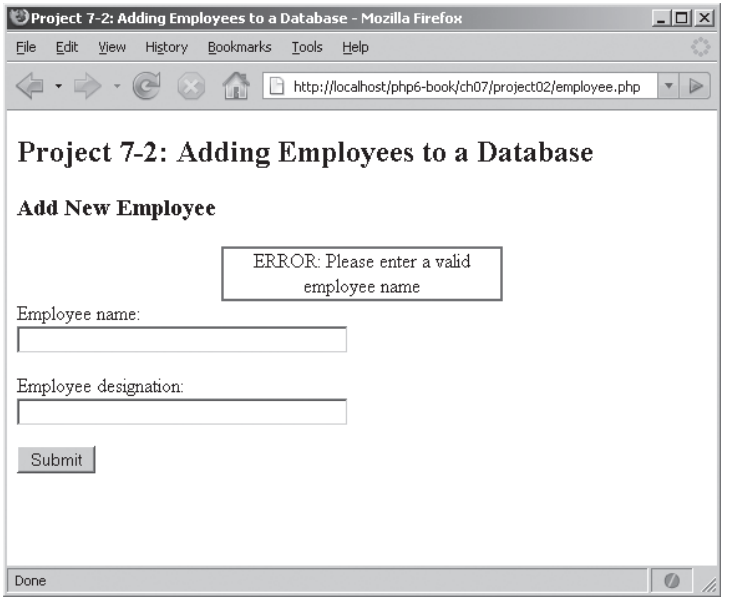


Figura 7-6 Los datos de salida después de enviar datos no válidos en el formulario Web

(*continúa*)

www.compumusicmania.com

```
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
    }
    table {
      border-collapse: collapse;
      width: 320px;
    }
    tr.heading {
      font-weight: bolder;
    }
    td {
      border: 1px solid black;
      padding: 0 0.5em;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-2: Añadir empleados a una base de datos</h2>
    <h3>Añade un Nuevo Empleado</h3>
<?php
    // intenta conectarse con la base de datos MySQL
    $mysqli = new mysqli("localhost", "usuario", "contra", "empleados");
    if ($mysqli === false) {
      die("ERROR: No fue posible conectarse con la base de datos. " .
mysqli_connect_error());
    }

    // si el formulario ha sido enviado
    // procesa los datos del formulario
    if (isset($_POST['submit'])) {
      // abre bloque de mensaje
      echo '<div id="message">';

      // recupera y verifica los valores de entrada
      $inputError = false;
      if (empty($_POST['emp_nombre'])) {
        echo 'ERROR: Por favor ingrese un nombre de empleado válido';
        $inputError = true;
      } else {
        $nombre = $mysqli->escape_string($_POST['emp_nombre']);
      }

      if ($inputError != true && empty($_POST['emp_puesto'])) {
        echo 'ERROR: Por favor ingrese un puesto válido';
        $inputError = true;
      } else {
```

```
        $puesto = $mysqli->escape_string($_POST['emp_puesto']);
      }

      // añade valores a la base de datos utilizando el consulta INSERT
      if ($inputError != true) {
      $sql = "INSERT INTO empleados (nombre, puesto)
              VALUES ('$nombre', '$puesto')";
        if ($mysqli->query($sql) === true) {
          echo 'Nuevo registro de empleado añadido con ID: ' . $mysqli
          ->insert_id;
        } else {
          echo "ERROR: No fue posible ejecutar el consulta: $sql. " .
$mysqli->error;
        }
      }

      // cierra bloque de mensaje
      echo '</div>';
    }
?>
    </div>

    <form action="empleados.php" method="POST">
      Nombre del empleado: <br />
      <input type="text" name="emp_nombre" size="40" />
      <p />
      Puesto del empleado: <br />
      <input type="text" name="emp_puesto" size="40" />
      <p />
      <input type="submit" name="submit" value="Enviar" />
    </form>

    <h3>Lista de empleados</h3>
<?php
    // obtiene registros
    // da formato como una tabla HTML
    $sql = "SELECT id, nombre, puesto FROM empleados";
    if($result = $mysqli->query($sql)) {
      if ($result->num_rows > 0) {
        echo "<table>\n";
        echo "  <tr class=\"heading\">\n";
        echo "    <td>ID</td>\n";
        echo "    <td>Nombre</td>\n";
        echo "    <td>Puesto</td>\n";
        echo "    </tr>\n";
        while ($row = $result->fetch_object()) {
          echo "  <tr>\n";
          echo "    <td>" . $row->id . "</td>\n";
```

(*continúa*)

```
            echo "    <td>" . $row->nombre . "</td>\n";
            echo "    <td>" . $row->puesto . "</td>\n";
            echo "  </tr>\n";
          }
          echo "</table>";
          $result->close();
      } else {
          echo "No hay empleados en la base de datos.";
        }
      } else {
        echo "ERROR: No fue posible ejecutar el consulta: $sql. " .
$mysqli->error;
      }

      // cierra conexión
      $mysqli->close();
?>
  </body>
</html>
```

Esta versión del script ahora incluye la consulta `SELECT`, que recupera los registros de la tabla de empleados como un objeto y los procesa utilizando un bucle. Estos registros son formados como una tabla HTML y presentados en la parte inferior de la página Web. Un efecto secundario útil de estas modificaciones es que los nuevos registros de empleados guardados mediante el formulario se reflejarán de inmediato en la tabla HTML una vez que se envíe el formulario.

La figura 7-7 muestra los datos de salida de esta versión modificada del script.

Utilizar la extensión SQLite de PHP

En este punto ya sabes conectar un script PHP a una base de datos MySQL para recuperar, añadir, editar y borrar registros. Sin embargo, MySQL no es el único juego en la ciudad; PHP 5.x también incluye soporte integrado para SQLite, opción eficiente y más ligera que MySQL. Esta sección aborda la base de datos SQLite y la extensión SQLite de PHP con gran detalle.

Introducción a SQLite

SQLite es una base de datos rápida y eficiente que ofrece una opción viable a MySQL, sobre todo para aplicaciones pequeñas y de tamaño mediano. Sin embargo, a diferencia de MySQL, que contiene una gran cantidad de componentes interrelacionados, SQLite está integrada en una sola biblioteca. También es significativamente más pequeña que MySQL, su línea de comandos pesa menos de 200 KB, y soporta todos los comandos SQL estándar con los que estás familiarizado. MySQL y SQLite también difieren en sus políticas de licencia: a diferencia de

www.compumusicmania.com

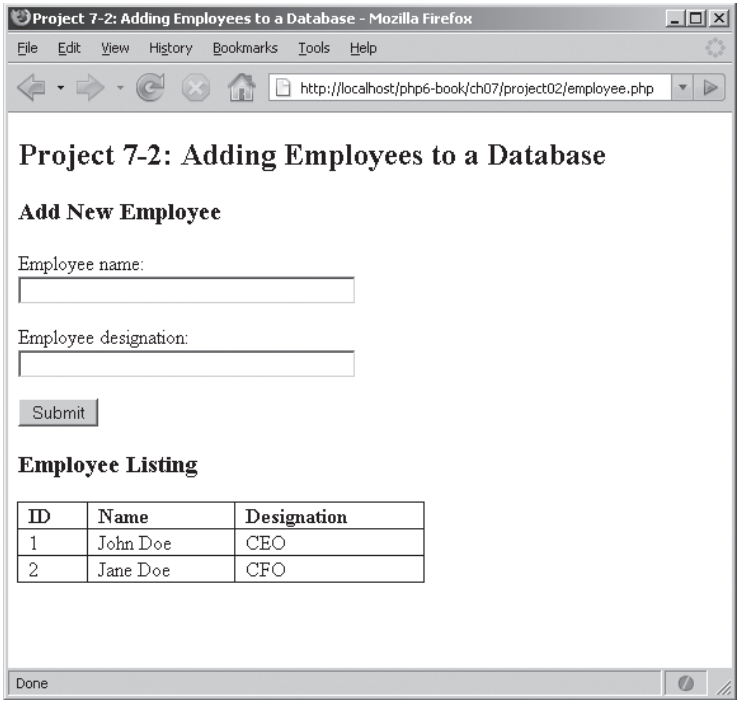


Figura 7-7 Página Web que muestra la lista de los empleados

MySQL, el código fuente de SQLite es completamente de dominio público, lo cual significa que los desarrolladores pueden utilizarlo y distribuirlo como les plazca, para productos tanto comerciales como no comerciales.

Sin embargo, el tamaño de SQLite disfraza su poder. La base de datos tiene una capacidad de soportar dos terabytes y puede tener un mejor rendimiento que MySQL en ciertas situaciones. En parte, esto se debe a razones estructurales: SQLite lee y escribe registros directamente del disco y por lo mismo ocupa menos recursos que MySQL, que opera sobre una arquitectura cliente-servidor que puede ser afectada por variables relacionadas con el nivel de la red.

NOTA

En el siguiente ejercicio, las palabras en negritas indican comandos que debes escribir en la línea de comando de SQLite. Los comandos pueden escribirse en mayúsculas o minúsculas. Antes de comenzar con el ejercicio asegúrate de haber instalado, configurado y probado la base de datos SQLite de acuerdo con las instrucciones que aparecen en el apéndice A de este libro.

SQLite da soporte a todas las declaraciones estándar de SQL que ya conoces y has llegado a amar en las dos secciones anteriores: `SELECT`, `INSERT`, `DELETE`, `UPDATE` y

www.compumusicmania.com

`CREATE TABLE`. El siguiente ejemplo muestra el uso de SQLite replicando las tablas de la base de datos MySQL utilizadas en la sección anterior. Comienza iniciando el programa cliente de línea de comandos de SQLite y crea una nueva base de datos en el directorio en uso que lleve el nombre *musica.db*:

```
shell> sqlite musica.db
```

Si todo salió bien, verás un mensaje de bienvenida y un indicador SQL interactivo como el siguiente:

```
sqlite>
```

Ahora puedes comenzar a dictar declaraciones SQL desde este indicador. Comienza por crear una tabla que contenga la información de los artistas:

```
sqlite> CREATE TABLE artistas (
   ...> artista_id INTEGER NOT NULL PRIMARY KEY,
   ...> artista_nombre TEXT NOT NULL,
   ...> artista_pais TEXT NOT NULL
   ...>);
```

Contrariamente a MySQL, que ofrece un amplio rango de diferentes tipos de datos para sus campos, SQLite soporta sólo cuatro tipos, los cuales se muestran en la tabla 7-3.

Tipo de campo	Descripción
`INTEGER`	Campo numérico para firmar valores enteros
`REAL`	Campo numérico para valores de punto flotante
`TEXT`	Tipo cadena de caracteres
`BLOB`	Tipo binario

Tabla 7-3 Tipos de datos de SQLite

Lo que es más importante, SQLite es una base de datos "sin tipos". Esto significa que los campos de esta base de datos NO necesitan estar asociados a un tipo específico, e incluso cuando lo están, pueden almacenar valores de un tipo diferente al especificado. La única excepción a esta regla son los campos tipo `INTEGER PRIMARY KEY`: que son campos "autonuméricos" que generan identificadores numéricos únicos para cada registro de la tabla, de manera similar a los campos `AUTO_INCREMENT` de MySQL.

Con estos hechos en mente, continúa con el ejercicio y crea las restantes dos tablas utilizando estas declaraciones SQL:

```
sqlite> CREATE TABLE ratings (
   ...> rating_id INTEGER NOT NULL PRIMARY KEY,
```

```
   ...> rating_nombre TEXT NOT NULL
   ...> );
sqlite> CREATE TABLE canciones (
   ...> cancion_id INTEGER NOT NULL PRIMARY KEY,
   ...> cancion_titulo TEXT NOT NULL,
   ...> ex_cancion_artista INTEGER NOT NULL,
   ...> ex_cancion_rating INTEGER NOT NULL
   ...> );
```

Ahora alimenta las tablas con registros:

```
sqlite> INSERT INTO artistas (artista_id, artista_nombre, artista_pais)
   ...> VALUES ('1', 'Aerosmith', 'US');
sqlite> INSERT INTO artistas (artista_nombre, artista_pais)
   ...> VALUES ('Abba', 'SE');
sqlite> INSERT INTO ratings (rating_id, rating_nombre)
   ...> VALUES (4, 'Bueno');
sqlite> INSERT INTO ratings (rating_id, rating_nombre)
   ...> VALUES (5, 'Excelente');
sqlite> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
        ex_cancion_rating)
   ...> VALUES ('Janie''s Got a Gun', 1, 4);
sqlite> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
        ex_cancion_rating)
   ...> VALUES ('Crazy', 1, 5);
```

NOTA

El archivo de código para este libro tiene una lista completa con las declaraciones SQL `INSERT` necesarias para alimentar las tres tablas utilizadas en este ejercicio. Ejecuta esas declaraciones y termina de construir las tablas antes de proseguir.

Una vez que hayas terminado, prueba la declaración `SELECT`:

```
sqlite> SELECT artista_nombre, artista_pais FROM artistas
   ...> WHERE artista_pais = 'US'
   ...> OR artista_pais = 'UK';
Aerosmith|US
Timbaland|US
Take That|UK
Girls Aloud|UK
```

SQLite soporta por completo las fusiones SQL; he aquí un ejemplo:

```
sqlite> SELECT cancion_titulo, artista_nombre, rating_nombre
   ...> FROM canciones, artistas, ratings
   ...> WHERE canciones.ex_cancion_artista = artista.artista_id
   ...> AND canciones.ex_canciones_rating = ratings.rating_id
```

www.compumusicmania.com

```
   ...> AND ratings.rating_id >= 4
   ...> AND artistas.artista_pais != 'US';
En Las Delicious|Cubanismo|Fantastic
Pray|Take That|Good
SOS|Abba|Fantastic
Dancing Queen|Abba|Good
```

Recuperar datos

Recuperar datos de una base SQLite con PHP no es muy diferente de recuperarlos de una base MySQL. El siguiente script PHP muestra el proceso utilizando la base de datos *musica.db* creada en la sección anterior:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// reitera sobre una colección de resultados
// presenta cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $sqlite->query($sql)) {
  if($result->numRows() > 0) {
    while($row = $result->fetch()) {
      echo $row[0] . ":" . $row[1] . "\n";
    }
  } else {
    echo "No se encontraron registros que coincidieran con los criterios
del consulta.";
  }
} else{
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

El script realiza las siguientes acciones:

1. Abre el archivo de base de datos, para realizar las operaciones, al inicializar una instancia de la clase SQLiteDatabase y transmitiendo al constructor del objeto la ruta de acceso completa a la base de datos. Si este archivo de base de datos no es localizado, se creará uno vacío con el nombre proporcionado (siempre y cuando el script tenga privilegios de escritura sobre el directorio correspondiente).

www.compumusicamania.com

2. El siguiente paso consiste en crear y ejecutar la consulta SQL. Esto se realiza invocando el método `query()` del objeto SQLiteDatabase y transmitiéndole el consulta que se va a ejecutar. Dependiendo de si el consulta fue exitoso o fracasó, la función regresa un valor verdadero o falso; en caso de fallo, el código de error correspondiente a la razón de la falla puede obtenerse invocando el método `lastError()` del objeto SQLiteDatabase. La función `sqlite_error_string()` convierte este código de error en un mensaje comprensible para los humanos.

TIP

Hay una semejanza cercana entre estos pasos y los que seguiste para utilizar la extensión MySQL en la sección anterior.

3. Por otra parte, si la consulta se ejecutó correctamente y regresa uno o más registros, el valor de retorno del método `query()` es otro objeto, este último será una instancia de la clase SQLiteResult. Este objeto representa la colección de resultados regresada por el consulta, y expone la consulta para procesar registros individuales en la colección de resultados.

   Este script utiliza el método `fetch()`, que regresa el siguiente resultado de la colección de resultados como una matriz cada vez que se invoca. Utilizado en un bucle, este método proporciona una manera conveniente de hacer reiteraciones sobre la colección de resultados, un registro a la vez. Los campos individuales del registro pueden ser accesados como elementos de la matriz, utilizando ya sea el índice o el nombre del campo. La cantidad de resultados en la colección de éstos puede recuperarse a través del método `numRows()` del objeto SQLiteResult.

4. Una vez que la colección de resultados entera ha sido procesada y no restan operaciones por ejecutarse en el archivo de base de datos, es buena idea cerrar el manejador de la base de datos para liberar la memoria que ocupa, lo cual se realiza destruyendo el objeto SQLiteDatabase.

Recuperar registros como matrices y objetos

El método `fetch()` del objeto SQLiteResult acepta un modificador adicional, que controla la manera en que se recuperan los elementos de la colección de resultados. Este modificador puede ser cualquiera de los valores `SQLITE_NUM` (para recuperar cada registro como una matriz numérica indexada), `SQLITE_ASSOC` (para regresar cada registro como una matriz asociativa) o `SQLITE_BOTH` (para regresar cada registro de ambas maneras, como una matriz numérica indexada y como una matriz asociativa, además de la opción por defecto). He aquí un ejemplo, que muestra estos modificadores en acción y produce una serie de datos de salida equivalente al ejemplo anterior:

www.compumusicmania.com

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");
// intenta ejecutar el consulta
// presenta registros utilizando diferentes estilos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if($result = $sqlite->query($sql)) {

  // recupera registros como una matriz numérica
  $row = $result->fetch(SQLITE_NUM);
  echo $row[0] . ":" . $row[1] . "\n";

  // recupera registros como una matriz asociativa
  $row = $result->fetch(SQLITE_ASSOC);
  echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";

  // recupera registros como un objeto
  $row = $result->fetchObject();
  echo $row->artista_id . ":" . $row->artista_nombre . "\n";

} else {
  echo "ERROR: no fue posible ejecutar $sql. " . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

También es posible regresar cada registro como un objeto, reemplazando `fetch()` con el método `fetchObject()`. He aquí un ejemplo equivalente al anterior, sólo que en lugar de recuperar valores de campo como elementos de una matriz, lo hace como propiedades de un objeto:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");
// intenta ejecutar el consulta
// reitera sobre una colección de resultados
// presenta cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if($result = $sqlite->query($sql)) {
  if ($result->numRows() > 0) {
    while ($row = $result->fetchObject()) {
      echo $row->artista_id . ":" . $row->artista_nombre . "\n";
```

www.compumusicamania.com

```
    }
  } else {
    echo "No se encontraron registros que coincidieran con los criterios
del consulta.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

Una característica importante de la extensión SQLite es su capacidad de recuperar *todos* los registros de una colección de resultados al mismo tiempo como un conjunto de matrices anidadas, a través del método `fectchAll()` del objeto SQLiteResult. Un bucle `foreach` puede reiterar sobre esta colección anidada, recuperando un registro tras otro. El siguiente ejemplo muestra este procedimiento en acción:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// reitera sobre una colección de resultados
// presenta cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $sqlite->query($sql)) {
  $data = $result->fetchAll();
  if (count($data) > 0) {
    foreach ($data as $row) {
      echo $row[0] . ":" . $row[1] . "\n";
    }
  } else {
    echo "No se encontraron registros que coincidieran con los criterios
del consulta.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

www.compumusicmania.com

PRECAUCIÓN

El método `fetchAll()` regresa la colección de resultados completa como un conjunto de matrices anidadas, y todos ellos se almacenan en la memoria de la computadora hasta que termina el proceso. Para no agotar la memoria, no utilices este método si es posible que tu consulta regrese un gran número de registros.

Añadir y modificar datos

Para consultas que no regresan una colección de resultados, como `INSERT`, `UPDATE` y `DELETE`, la extensión SQLite ofrece el método `queyExec()`. El siguiente ejemplo lo muestra en acción, añadiendo un nuevo registro a la tabla *artistas*.

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// añade un nuevo registro
// datos de salida: "Nuevo artista con el id:8 ha sido añadido."
$sql = "INSERT INTO artistas (artista_nombre, artista_pais) VALUES
('James Blunt', 'UK')";
if (sqlite->consultaExec($sql) == true) {
  echo 'Nuevo artista con el id:' . $sqlite->lastInsertRowid() . 'ha sido
añadido.';
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

Dependiendo de si el consulta se ejecutó con éxito o no, la función `consultaExec()` regresa un valor verdadero o falso; es fácil revisar este valor de respuesta y mostrar un mensaje de éxito o de falla. Si el registro fue insertado en una tabla con un campo `INTEGER PRIMARY KEY`, SQLite automáticamente asignará al registro un número de identificación único. Este número puede ser recuperado con el método `lastInsertRowid()` del objeto SQLiteDatabase, como se mostró en el ejemplo anterior.

PRECAUCIÓN

Para que funcionen los comandos `INSERT`, `UPDATE` y `DELETE`, recuerda que el script PHP *debe* tener privilegios de escritura para el archivo de base de datos SQLite.

Cuando realizas un consulta `UPDATE` o `DELETE`, la cantidad de filas afectadas por éste también puede ser recuperada a través del método `changes()` del objeto SQLiteDatabase. El siguiente ejemplo lo muestra, actualizando los ratings de las canciones en la base de datos y regresando la cuenta de los registros afectados:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// actualiza registro
// datos de salida: "3 fila(s) actualizadas."
$sql = "UPDATE canciones SET ex_cancion_rating = 4 WHERE ex_cancion_
rating = 3";
if (sqlite->consultaExec($sql) == true) {
  echo $sqlite->changes() . 'fila(s) actualizadas.';
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

Manejo de errores

Ambos métodos, `query()` y `consultaExec()`, regresan un valor falso si ocurre un error durante la preparación o ejecución del consulta. Es fácil verificar el valor de retorno de estos métodos y recuperar el código correspondiente al error invocando el método `lastError()` del objeto SQLiteDatabase. Desafortunadamente, este código de error no es de gran utilidad por sí mismo; así que, para obtener una descripción literal de lo que sucedió, se debe envolver la invocación `lastError()` en la función `sqlite_error_string()`, como se hizo en la mayoría de los ejemplos anteriores.

Prueba esto 7-3 Crear una lista personal de pendientes

Ahora que ya tienes una idea del manejo de SQLite, ¿qué tal si lo utilizas en una aplicación práctica?: una lista personal de pendientes que puedas consultar y actualizar a través del explorador Web. Esta lista de pendientes te permitirá ingresar tareas y fechas de vencimiento, asignar prioridades a las tareas, editar sus descripciones y marcarlas cuando hayan sido com-

www.compumusicmania.com

pletadas. Es un poco más complicada que las aplicaciones que has hecho hasta ahora, porque incluye varias partes dinámicas; sin embargo, si has puesto atención hasta este punto, no será muy difícil que comprendas lo que está sucediendo.

Para comenzar, crea una nueva base de datos SQLite llamada *pendientes.db* y añade una tabla vacía para contener la descripción de las tareas y las fechas:

```
shell> sqlite pendientes.db
sqlite> CREATE TABLE tareas (
   ...> id INTEGER PRIMARY KEY,
   ...> nombre TEXT NOT NULL,
   ...> vencimiento TEXT NOT NULL,
   ...> prioridad TEXT NOT NULL
   ...> );
```

Esta tabla tiene cuatro campos: para el ID del registro, el nombre de la tarea, la fecha de vencimiento de ésta y la prioridad que tiene.

El siguiente paso consiste en crear un formulario Web para añadir nuevas tareas a la base de datos (*guarda.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-3: Crear una lista personal de pendientes</title>
    <style type="text/css">
    div#message{
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-3: Crear una lista personal de pendientes</h2>
    <h3>Añade una Nueva Tarea</h3>

    <?php
    // si el formulario no ha sido enviado
    // genera un nuevo formulario
    if (!isset($_POST['submit'])) {
    ?>

    <form method="post" action="guarda.php">
      Descripción: <br />
      <input type="text" name="nombre" />
```

www.compumusicmania.com

```
      <p>
      Fecha de vencimiento (dd/mm/aaaa): <br />
      <input type="text" name="dd" size="2" />
      <input type="text" name="mm" size="2" />
      <input type="text" name="aa" size="4" />
      <p>
      Prioridad: <br />
      <select name="prioridad">
        <option name="Alta">Alta</option>
        <option name="Media">Media</option>
        <option name="Baja">Baja</option>
      </select>
      <p>
      <input type="submit" name="sumbit" value="Guardar" />
    </form>

    <?php
    } else {
      // si el formulario ya fue enviado
      // intenta establecer conexión con la base de datos
      $sqlite = new SQLiteDataBase('pendientes.db') or die ("No fue
posible abrir la base de datos");

      // verifica y limpia los datos de entrada
      $nombre = !empty($_POST['nombre']) ? sqlite_escape_string
($_POST['nombre']) : die('ERROR: Se requiere el nombre de la tarea');
      $dd = !empty($_POST['dd']) ? sqlite_escape_string (int)
($_POST['dd']) : die('ERROR: Se requiere fecha de vencimiento');
      $mm = !empty($_POST['mm']) ? sqlite_escape_string (int)
($_POST['mm']) : die('ERROR: Se requiere fecha de vencimiento');
      $aa = !empty($_POST['aa']) ? sqlite_escape_string (int)
($_POST['aa']) : die('ERROR: Se requiere fecha de vencimiento');
      $fecha = checkdate($mm, $dd, $aa) ? mktime(0, 0, 0, $mm, $dd, $aa)
: die('ERROR: La fecha de vencimiento es inválida');
      $prioridad = !empty($_POST['prioridad']) ? sqlite_escape_string
($_POST['prioridad']) : die('ERROR: Se requiere la prioridad de la
tarea');

      // intenta ejecutar el consulta
      // añade nuevo registro
      $sql = "INSERT INTO tareas (nombre, vencimiento, prioridad) VALUES
('$nombre', '$vencimiento', '$prioridad')";
      if ($sqlite->consultaExec($sql) == true){
        echo '<div id="message"> El registro ha sido añadido con éxito a
la base de datos.</div>';
      } else {
        echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
```

(*continúa*)

```
    }

    // cierra conexión
    unset($sqlite);
  }
  ?>

  </body>
</html>
```

La figura 7-8 muestra el formulario Web.

Cuando el usuario envía este formulario, primero se validan los datos para asegurar que estén presentes todos los campos requeridos. Los datos ingresados en los tres campos correspondientes a la fecha también son verificados con la función PHP `checkdate()` para asegurar que los tres juntos formen una fecha válida. Después, se limpian los datos de entrada al pasarlos por la función `sqlite_escape_string()`, que elimina los caracteres especiales de los datos de entrada automáticamente y los guarda en la base de datos utilizando el consulta `INSERT`.

La figura 7-9 muestra el resultado de añadir con éxito una nueva tarea a la base de datos.

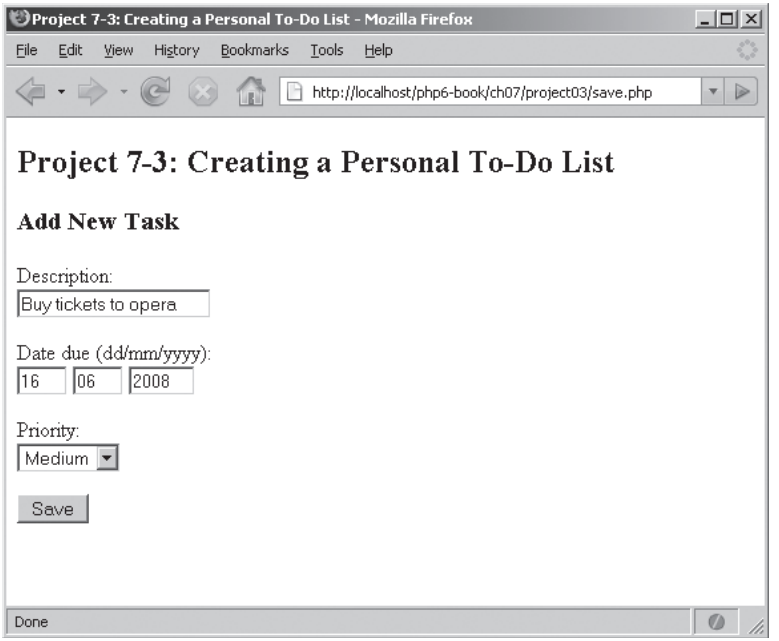


Figura 7-8 Formulario Web para añadir tareas a la base de datos

www.comprumusicmania.com

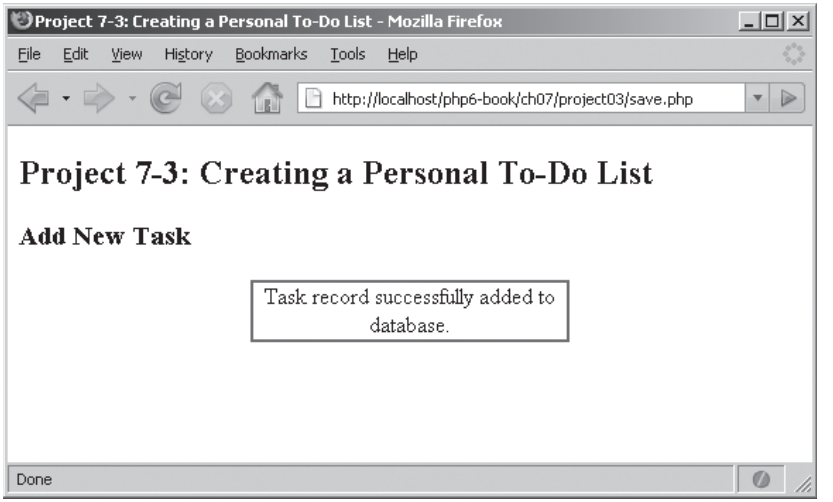


Figura 7-9 El resultado de añadir un nuevo pendiente a la base de datos

Eso es suficiente para ingresar tareas a la lista de pendientes. ¿Qué tal si ahora los vemos? Como probablemente lo habrás adivinado, es cuestión de utilizar una consulta `SELECT` para recuperar todos los registros de la base de datos y luego dar formato a la información resultante de manera que sea apropiada para desplegarse en una página Web. He aquí el código (*lista.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-3: Crear una lista de pendientes personales</title>
    <style type="text/css">
    div#message{
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:60%;
      border: solid 2px green
    }
    table{
    border-collapse: collapse;
    width: 500px;
    }
    tr.heading{
    font-weight: bolder;
     }
```

(*continúa*)

www.compumusicmania.com

```
     td{
    border: 1px solid black;
    padding: 1em;
     }
     tr.high {
       background: #cc1111;
     }
     tr.medium {
       background: #00aaaa;
     }
     tr.low {
       background: #66dd33;
     }
     a {
        color: black;
        border: outset 2px black;
        text-decoration: none;
        padding: 3px;
     }
     </style>
  </head>
  <body>
    <h2>Proyecto 7-3: Crear una lista de pendientes personales</h2>
    <h3>Lista de Tareas</h3>

<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('pendientes.db') or die ("No fue posible
abrir la base de datos");

// obtiene registros
// como una tabla HTML
$sql = "SELECT id, nombre, vencimiento, prioridad FROM tareas ORDER BY
vencimiento";
if ($result = $sqlite->query($sql)) {
  if ($result->numRows() > 0) {
    echo "<table>\n";
    echo "  <tr class=\"heading\">\n";
    echo "    <td>Descripción</td>\n";
    echo "    <td>Fecha de vencimiento</td>\n";
    echo "    <td></td>\n";
    echo "  </tr>\n";
    while($row = $result->fetchObject()) {
      echo "  <tr class=\"" . strtolower($row->priority) . "\">\n";
      echo "    <td>" . $row->nombre . "</td>\n";
      echo "    <td>" . date('d M Y', $row->vencimiento) . "</td>\n";
      echo "    <td><a href=\"marca.php?id=" . $row->id . "\"> Marcar
como finalizada</a></td>\n";
      echo "  </tr>\n";
```

www.compumusicamania.com

```
    }
    echo "</table>";
  } else {
    echo '<div id="message"> No hay tareas en la base de datos.</div>';
  }
} else {
  echo 'ERROR: No fue posible ejecutar consulta: $sql.' . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>

  <p/>
  <a href="guarda.php">Añadir una nueva tarea</a>

  </body>
</html>
```

No hay nada extraño aquí: el script ejecuta la consulta `SELECT` con el método `exec()`, y luego itera sobre la colección de resultados, presentando cada registro encontrado como una fila de una tabla HTML. Advierte que dependiendo del campo *prioridad* de cada registro, la fila de la tabla HTML correspondiente tiene un color diferente: rojo, verde o azul.

La figura 7-10 muestra un ejemplo de los datos de salida.

Verás algo más en la figura: cada elemento de la lista de pendientes viene con una opción `'Marcar como Finalizada'`. Esta opción apunta hacia otro script PHP, *marca.php*, que es responsable de eliminar el registro correspondiente de la base de datos. Mira con atención el URL que apunta a *marca.php*, y verás que el ID del registro también es transmitido, como la variable `$id`. Dentro de *marca.php*, este ID será accesado a través de la matriz `$_GET`, como `$_GET['id']`.

¿Qué hace *marca.php*? Nada muy complejo, simplemente utiliza el ID del registro transmitido por `$_GET` para formular y ejecutar un consulta `DELETE`, que elimina el registro de la base de datos. He aquí el código (*marca.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-3: Crear una lista de pendientes personales</title>
    <style type="text/css">
    div#message {
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
```

(*continúa*)

www.compumusicmania.com

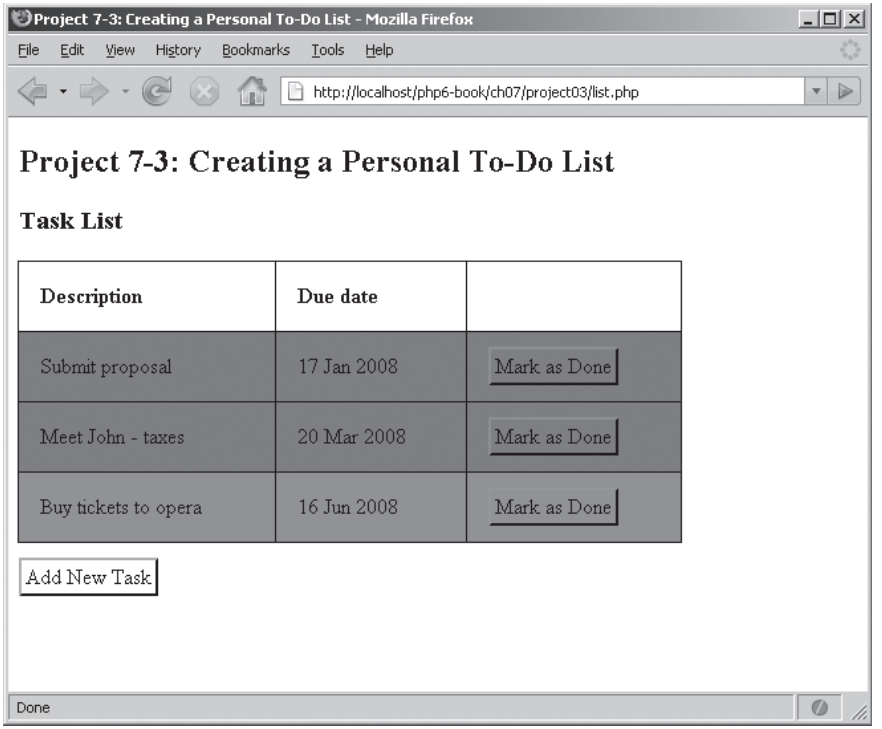


Figura 7-10 Página Web que muestra los elementos de la lista de pendientes, ordenados por fecha de vencimiento

```
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-3: Crear una lista de pendientes personales</h2>
    <h3>Elimina la Tarea Terminada</h3>

    <?php
    if (isset($_GET['id'])) {
      // intenta establecer conexión con la base de datos
      $sqlite = new SQLiteDataBase('pendientes.db') or die ("No fue
posible abrir la base de datos");

      // verifica y limpia los datos de entrada
      $id = !empty($_GET['id']) ? sqlite_escape_string((int)$_GET['id'])
: die('ERROR: Se requiere el ID de la tarea');

      // elimina registro
      $sql = "DELETE FROM tareas WHERE id = '$id'";
```

www.computrucionmania.com

```
      if ($sqlite->consultaExec($sql) == true) {
        echo '<div id="message">Registro de la tarea eliminado
exitosamente de la base de datos.</div>';
      } else {
        echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
      }

      // cierra conexión
      unset($sqlite);
    } else {
      die ('ERROR: Se requiere el ID de la tarea');
    }
    ?>

  </body>
</html>
```

La figura 7-11 muestra los datos de salida al eliminar de manera correcta una tarea de la base de datos.

Y ahora, cuando revisas *lista.php*, tu lista de pendientes ya no mostrará el elemento que has marcado como terminado. Sencillo, ¿no es cierto?

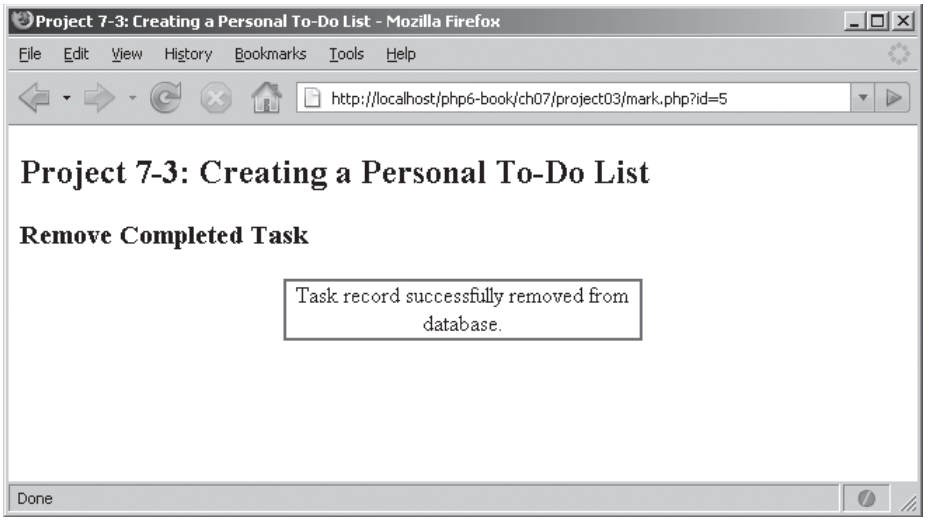


Figura 7-11 El resultado de marcar una tarea como realizada

www.compumusicmania.com

Utilizar las extensiones PDO de PHP

En las secciones anteriores aprendiste a integrar tus aplicaciones PHP con las bases de datos MySQL y SQLite. Como ya habrás notado, las extensiones MySQLi y SQLite utilizan diferentes nombres de métodos para realizar su trabajo; como resultado, conmutar de una base de datos a otra implica, en esencia, volver a escribir todo el código de base de datos para utilizar los nuevos métodos. Por ello PHP ofrece una extensión neutral para bases de datos: objetos de datos de PHP (PDO), que brinda gran portabilidad y puede reducir de manera significativa el esfuerzo que implica conmutar de un sistema de base de datos a otro.

La siguiente sección aborda la extensión PDO con gran detalle, proporcionando información sobre la manera en que se utiliza para conectarse a diferentes sistemas de bases de datos, realizar consultas, procesar colecciones de resultados, además de manejar errores de consultas y conexión. La mayor parte de los ejemplos dan por hecho que se trabaja con una base de datos MySQL; sin embargo, como verás, los programas basados en PDO requieren mínimas modificaciones para trabajar con otros sistemas de bases de datos, incluyendo SQLite.

Recuperar datos

PDO trabaja proporcionando un conjunto estándar de funciones para realizar operaciones comunes de base de datos, como conexión, consultas, procesamiento de colecciones de resultados y manejo de errores; internamente traduce estas funciones a las invocaciones API nativas comprensibles para la base de datos en uso. Para mostrar la manera en que funciona, examina el siguiente ejemplo, que ejecuta una consulta `SELECT` y despliega los registros encontrados:

```
<?php
// intenta establecer conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("Error: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta SELECT
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $pdo->query($sql)) {
  while($row = $result->fetch()) {
    echo $row[0] . ":" . $row[1] . "\n";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
```

www.compumusicamania.com

```
}

// cierra conexión
unset($pdo);
?>
```

Como lo muestra este ejemplo, utilizar PDO para obtener datos de una base implica pasos semejantes a los que has visto en secciones previas:

1. El primer paso consiste en inicializar una instancia de la clase PDO y pasar al constructor del objeto tres argumentos: una cadena con el nombre del origen de datos (DSN, Data Source Name), indicando el tipo de base de datos al que se va a conectar, además de otras opciones específicas de la base, un nombre de usuario válido reconocido por la base en cuestión y su correspondiente contraseña. La cadena DSN varía de una a otra base de datos; por lo general puedes obtener el formato exacto para esta cadena de la documentación de la base que estés utilizando.

   La tabla 7-4 muestra algunos formatos de cadenas DSN comunes.

   Si el intento de conexión fracasa, se generará una excepción; esta excepción puede ser recogida y manejada utilizando el mecanismo de manejo de excepciones de PHP (puede obtenerse más información sobre el manejo de excepciones en el capítulo 10 de este libro).

2. Suponiendo que la conexión fue correcta, el siguiente paso consiste en formular una consulta SQL y ejecutarla utilizando el método `query()` de PDO. El valor de regreso de este método es una colección de resultados, representada por el objeto PDOStatement. El contenido de la colección de resultados puede procesarse utilizando el método `fetch()`

Base de datos	Cadena DSN
MySQL	`'mysql:host=`*host*`; port=`*puerto*`;dbname=`*db*`'`
SQLite	`'sqlite:`*ruta/a/archivo/basededatos*`'`
PostgreSQL	`'pgsql:host=`*host* `port=`*puerto* `dbname=`*db* `usuario=`*usuario* `password=`*contra*`'`
Oracle	`'oci:dbname=//`*host*`:`*puerto*`/`*db*`'`
Firebird	`'firebird:Usuario=`*usuario*`;Password=`*contra*`; Database=`*db*`; DataSource=`*host*`;Port=`*puerto*`'`

Tabla 7-4 Cadenas DSN comunes

www.compumusicamania.com

del objeto, que regresa el siguiente registro en la colección de resultados como una matriz (tanto asociativa como indexada). Es posible acceder a los campos individuales del registro como elementos de la matriz en un bucle, utilizando el índice del campo o su nombre.

Pregunta al experto

P: ¿Cómo calculo el número de registros en mi colección de resultados con PDO?

R: A diferencia de las extensiones MySQL y SQLite, PDO no ofrece métodos ni propiedades integrados para recuperar directamente el número de registros en una colección de resultados. Esto se debe a que no todos los sistemas de bases de datos regresan esta información, por lo que PDO no puede proporcionar esta información de manera portátil. Sin embargo, si necesitas esta información, el manual PHP recomienda ejecutar la declaración `SELECT COUNT(*) ...` para obtenerla, con la consulta deseada y recuperar el primer campo de la colección de resultados, que contendrá el número de registros que coinciden con la consulta. Para mayor información, revisa en el análisis de www.php.net/manual/en/function.PDOStatement-rowCount.php.

El método `fetch()` del objeto PDOStatement acepta un modificador adicional, que controla la manera en que se realiza la búsqueda en la colección de resultados. Algunos valores aceptados por este modificador se muestran en la tabla 7-5.

Modificador	Lo que hace
`PDO::FETCH_NUM`	Regresa cada registro como una matriz numérica indexada
`PDO::FETCH_ASSOC`	Regresa cada registro como una matriz asociativa cuya clave es el nombre de campo
`PDO::FETCH_BOTH`	Regresa cada registro de ambas maneras, como una matriz numérica indexada y como una matriz asociativa (el valor por defecto)
`PDO::FETCH_OBJ`	Regresa cada registro como un objeto con propiedades correspondientes a los nombres de campo
`PDO::FETCH_LAZY`	Regresa cada registro como una matriz numérica indexada, como una matriz asociativa y como un objeto

Tabla 7-5 Modificadores del método `fetch()` de PDO

www.compumusicmania.com

El siguiente ejemplo muestra estos modificadores en acción:

```
<?php
// intenta establecer una conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("Error: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta SELECT
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $pdo->query($sql)){

  // regresa registro como matriz numérica
  $row = $result->fetch(PDO::FETCH_NUM);
  echo $row[0] . ":" . $row[1] . "\n";

  // regresa registro como matriz asociativa
  $row = $result->fetch(PDO::FETCH_ASSOC);
  echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";

  // regresa registro como un objeto
  $row = $result->fetch(PDO::FETCH_OBJ);
  echo $row->artista_id . ":" . $row->artista_nombre . "\n";

  // regresa registro utilizando una combinación de estilos
  $row = $result->fetch(PDO::FETCH_LAZY);
  echo $row['artista_id'] . ":" . $row->artista_nombre . "\n";

} else {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo-
>errorInfo());
}

// cierra conexión
unset($pdo);
?>
```

Añadir y modificar datos

PDO también facilita la ejecución de consultas `INSERT`, `UPDATE` y `DELETE` con su método `exec()`. Este método, que está diseñado para instrucciones que de alguna manera modifican la base de datos, regresa la cantidad de registros afectados por el consulta. He aquí un ejemplo de su uso para insertar y eliminar un registro:

www.compumusicmania.com

```
<?php
// intenta establecer una conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("ERROR: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta INSERT
$sql = "INSERT INTO artistas (artista_nombre, artista_pais) VALUES
('Luciano Pavarotti', 'IT')";
$ret = $pdo->exec($sql);
if ($ret === false) {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
} else {
  $id = $pdo->lastInsertId();
  echo 'Nuevo artista con id: ' . $id . 'ha sido añadido.';
}

// crea y ejecuta un consulta DELETE
$sql = "DELETE FROM artistas WHERE artista_pais = 'IT'";
$ret = pdo->exec($sql);
if ($ret === false) {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
} else {
  echo 'Eliminados ' . $ret . ' registros.';
}

// cierra conexión
unset($pdo);
?>
```

Este código utiliza el método `exec()` dos veces, primero para insertar un nuevo registro y luego para eliminar registros que coincidan con una condición específica. Si la consulta transmitida a `exec()` falla, `excec()` regresa un valor falso; de lo contrario, regresa la cantidad de registros afectados por la consulta. Advierte también que el script utiliza el método `lastInsertId()` del objeto PDO, el cual regresa el ID generado por el último comando `INSERT` en caso de que la tabla contenga un campo de autoincremento.

TIP

Si no hay registros afectados por la ejecución de la consulta realizada por el método `exec()`, éste regresará un valor igual a cero. No confundas este valor con el valor booleano "false" (falso) regresado por `exec()` cuando falla la consulta. Para evitar confusiones, el manual de PHP recomienda siempre probar el valor de retorno de `exec()` con el operador === en lugar de utilizar el operador ==.

Utilizar declaraciones preparadas

PDO también da soporte a declaraciones preparadas, mediante sus métodos `prepare()` y `execute()`. El siguiente ejemplo lo ilustra, utilizando una declaración preparada para añadir varias canciones a la base de datos:

```
<?php
// define los valores que serán insertados
$canciones = array(
  array('Voulez-Vous', 2, 5),
  array('Take a Chance on Me', 2, 3),
  array('I Have a Dream', 2, 4),
  array('Thank You For The Music', 2, 4),
);

// intenta establecer una conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("ERROR: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta SELECT
sql = "INSERT INTO canciones (cancion_titulo, ex_cancion_artista, ex_
cancion_rating) VALUES (?, ?, ?)";
if ($stmt = $pdo->prepare($sql)) {
  foreach ($canciones as $s) {
    $stmt->bindParam(1, $s[0]);
    $stmt->bindParam(2, $s[1]);
    $stmt->bindParam(3, $s[2]);
    if ($stmt->execute()) {
      echo "Nueva canción con id: " . $pdo->lastInsertId() . "ha sido
añadida. \n";
    } else {
      echo "ERROR: No fue posible ejecutar consulta: $sql. " . print_r
($pdo->errorInfo());
    }
  }
} else {
  echo "ERROR: No fue posible preparar consulta: $sql. " . print_r($pdo
->errorInfo());
}

// cierra conexión
unset($pdo);
?>
```

Si comparas este último script con uno similar de MySQLi en las secciones anteriores, verás una similitud muy marcada. Como antes, este script también define una consulta SQL modelo que contiene una declaración `INSERT` formada por contenedores en lugar de valores, y luego convierte este modelo en una declaración preparada utilizando el método `prepare()` del objeto PDO.

En caso de tener éxito, `prepare()` regresa un objeto PDOStatement que representa la declaración preparada. Los valores que serán interpolados en la declaración se unen entonces a los contenedores invocando repetidamente el método `bindParam()` del objeto PDOStatement con dos argumentos, la posición del contenedor y la variable a la que será unida. Una vez que las variables están unidas, el método `execute()` del objeto PDOStatement se ocupa de ejecutar la declaración preparada con los valores correctos.

NOTA

Cuando utilices declaraciones preparadas con PDO, es importante considerar que los beneficios de este tipo de declaraciones sólo estarán disponibles si la base de datos con la que trabajas soporta estas declaraciones de forma nativa. Para bases de datos que no soportan las declaraciones preparadas, PDO convertirá internamente las invocaciones para `prepare()` y `execute()` en declaraciones SQL estándar y en estos casos no obtendrás ningún beneficio.

Manejar errores

Cuando se realizan operaciones de bases de datos con PDO, pueden ocurrir errores durante la fase de conexión o durante la ejecución de la consulta. PDO ofrece herramientas robustas para manejar ambos tipos de errores.

Errores de conexión

Si PDO no puede conectarse con la base de datos especificada utilizando el DSN, nombre de usuario y contraseña proporcionados, automáticamente generará una PDOException. Esta excepción puede capturarse utilizando el mecanismo estándar de manejo de excepciones de PHP (abordado con detalle en el capítulo 10), y es posible desplegar un mensaje de error explicando las causas del mismo en el objeto excepción.

Errores de ejecución de la consulta

Si ocurre un error durante la preparación o ejecución de la consulta, PDO proporciona información sobre el mismo con el método `errorInfo()`. Regresa una matriz que contiene tres elementos: el código de error SQL, el código de error de la base de datos y un mensaje de error legible para los humanos (también generado por la base de datos en uso). Es fácil procesar esta matriz y presentar los elementos apropiados que contiene desde la sección de manejo de errores de tu script.

www.compumusicmania.com

Prueba esto 7-4 Construir un formulario de inicio de sesión

Pongamos ahora en práctica PDO con otra aplicación práctica, una que encontrarás una y otra vez durante tu desarrollo como programador PHP: un formulario de ingreso. Esta aplicación generará un formulario Web para que los usuarios ingresen su nombre de usuario y contraseña; luego verificará estos datos de entrada contra los almacenados en la base de datos y permitirá o rechazará su ingreso. Primero, la aplicación utilizará la base de datos MySQL; sin embargo, después verás qué tan portátil es el código PDO, cuando conmutemos la aplicación hacia una base de datos SQLite.

Utilizar una base de datos MySQL

Para comenzar, arranca el programa cliente de línea de comandos de MySQL y crea una tabla que contenga los nombres de usuario y las contraseñas, como sigue:

```
mysql> CREATE DATABASE app;
Query OK, 0 rows affected (0.07 sec)
mysql> USE app;
Query OK, 0 rows affected (0.07 sec)
mysql> CREATE TABLE usuarios (
    -> id int(4) NOT NULL AUTO_INCREMENT,
    -> nombredeusuario VARCHAR(255) NOT NULL,
    -> contrasena VARCHAR(255) NOT NULL,
    -> PRIMARY KEY (id));
Query OK, 0 rows affected (0.07 sec)
```

En este punto es buena idea "sembrar" la tabla ingresando algunos nombres de usuario y contraseñas. Para simplificarlo, los nombres de usuario serán iguales a las contraseñas, pero cifradas para que estén a salvo de mirones casuales (y de hackers no tan casuales):

```
mysql> INSERT INTO usuarios (nombredeusuario, contrasena)
    -> VALUES ('john', '$1$Tk0.gh4.$42EZDbQ4mOfmXMq.0m1tS1');
Query OK, 1 row affected (0.21 sec)

mysql> INSERT INTO usuarios (nombredeusuario, contrasena)
    -> VALUES ('jane', '$1$.15.tR/.$XK1KW1Wzqy0UuMFKQDHH00');
Query OK, 1 row affected (0.21 sec)
```

Ahora todo lo que se necesita es un formulario de inicio de sesión, y algo de código PHP para leer los valores ingresados en el formulario y compararlos contra los valores almacenados en la base de datos. He aquí el código (*ingreso.php*):

(*continúa*)

www.compumusicmania.com

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-4: Construir un formulario de ingreso</title>
  </head>
  <body>
    <h2>Proyecto 7-4: Construir un formulario de ingreso</h2>
<?php
    // si el formulario no ha sido enviado
    // genera un nuevo formulario
    if (!isset($_POST['submit'])) {
?>
    <form method="post" action="ingreso.php">
      Nombre de Usuario: <br />
      <input type="text" name="nombredeusuario" />
      <p>
      Contraseña: <br />
      <input type="contraword" name="contrasena" />
      <p>
      <input type="submit" name="submit" value="Ingresar" />
    </form>

<?php
    // si el formulario ha sido enviado
    // verifica los datos proporcionados
    // contra la base de datos
    } else {
      $nombredeusuario = $_POST['nombredeusuario'];
      $contrasena = $_POST['contrasena'];

      // verifica datos de entrada
      if(empty($nombredeusuario)) {
        die('ERROR: Por favor escriba su nombre de usuario');
      }
      if(empty($contrasena)) {
        die('ERROR: Por favor escriba su contraseña');
      }

      // intenta establecer conexión con la base de datos
      try {
         $pdo = new PDO('mysql:dbname=app;host=localhost', 'usuario',
'contra');
      } catch (PDOException $e) {
         die("ERROR: No fue posible conectar: " . $e->getMessage());
      }

      // limpia los caracteres especiales de los datos de entrada
      $nombredeusuario = $pdo->quote($nombredeusuario);
```

www.compumusicamania.com

```
      // verifica si existe el nombre de usuario
      $sql = "SELECT COUNT(*) FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
      if($result = $pdo->query($sql)) {
        $row = $result->fetch();
        // si es positivo, busca la contraseña cifrada
        if($row[0] == 1) {
          $sql = "SELECT contrasena FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
          // cifra la contraseña ingresada en el formulario
          // la verifica contra la contraseña cifrada que reside en la
base de datos
          // si ambas coinciden, la contraseña es correcta
          if ($result = $pdo->query($sql)) {
            $row = $result->fetch();
            $salt = $row[0];
            if (crypt($contrasena, $salt) == $salt) {
              echo 'Sus credenciales de acceso fueron verificadas
positivamente.';
            } else {
              echo 'Ha ingresado una contraseña incorrecta.';
            }
          } else {
            echo "ERROR: No fue posible ejecutar $sql. " . print_r
($pdo->errorInfo());
          }
        } else {
          echo 'Ha ingresado un nombre de usuario incorrecto.';
        }
      } else {
        echo "ERROR: No fue posible ejecutar $sql. " . print_r
($pdo->errorInfo());
      }

      // cierra conexión
      unset($pdo);
    }
?>
  </body>
</html>
```

La figura 7-12 muestra la apariencia del formulario de ingreso.

(*continúa*)

www.compumusicamania.com

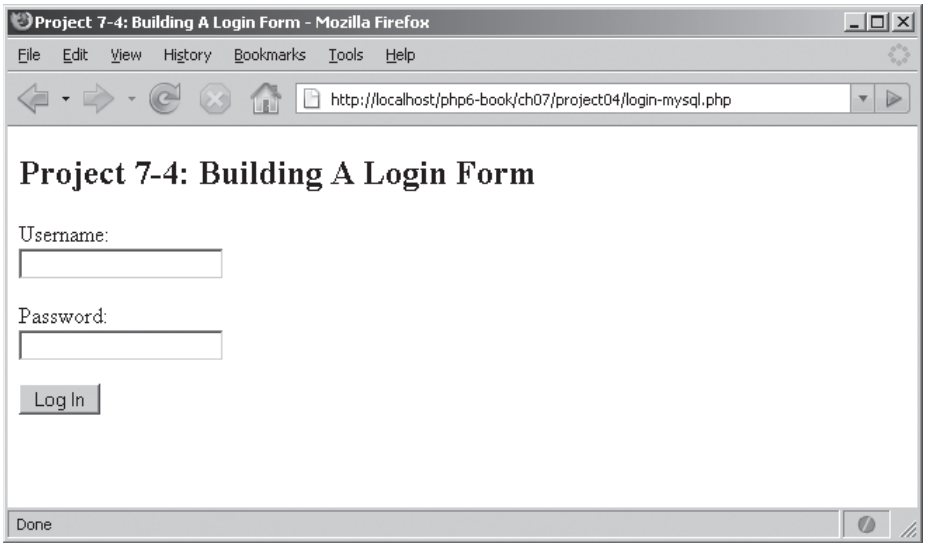


Figura 7-12 Formulario de ingreso en una página Web

Cuando se transmite este formulario, la segunda mitad del script entra en juego. Se realizan varios pasos:

1. Verifica los datos de entrada del formulario para asegurar que el nombre de usuario y la contraseña estén presentes, y detiene la ejecución del script con un mensaje de error si falta cualquiera de los dos. También limpia los caracteres especiales de los datos de entrada utilizando el método `quote()`.
2. Abre una conexión a la base de datos y ejecuta la consulta `SELECT` para verificar si existe una coincidencia con la información almacenada en la base de datos. En caso de que esta verificación dé como resultado un valor falso, genera el mensaje de error "Ha ingresado un nombre de usuario incorrecto".
3. Si el nombre de usuario existe, entonces el script procede a revisar la contraseña. Dado que ésta se cifró utilizando un esquema de cifrado de una sola vía, esta verificación no puede realizarse directamente; la única manera de realizar la tarea es volver a cifrar la contraseña del usuario, a partir de la manera en que ha sido ingresada en el formulario, y comparar esta versión contra la que se encuentra almacenada en la base de datos. En caso de que ambas cadenas de caracteres cifradas coincidan, significa que la contraseña ingresada es correcta.

www.compumusicmania.com

Figura 7-13 El resultado de ingresar una contraseña incorrecta en el formulario de registro

4. Dependiendo del resultado de la prueba, el script genera el mensaje '`Ha ingresado una contraseña incorrecta.`' o bien '`Sus credenciales de acceso fueron verificadas positivamente`'.

La figura 7-13 muestra el resultado de ingresar una contraseña incorrecta y la figura 7-14 muestra el resultado de ingresar correctamente la contraseña.

Figura 7-14 El resultado de ingresar una contraseña válida en el formulario de registro

www.comunidadmania.com

Pregunta al experto

P: ¿Cómo genero la contraseña cifrada que se utiliza en el ejemplo?

R: Las cadenas de contraseña cifradas que se utilizan en el ejemplo fueron generadas por la función PHP `crypt()`, que utiliza un esquema de cifrado en un solo sentido que aplica a cualquier cadena de caracteres transmitida. El cifrado en un solo sentido hace que la contraseña original sea irrecuperable (de ahí el término "en un solo sentido"). Este cifrado se basa en una clave única o falseada (*salt*), que puede proporcionarse opcionalmente a la función como segundo argumento; en ausencia de ésta, PHP genera de manera automática una clave falseada (*salt*).

La contraseña original ya no es recuperable después de pasar por la función `crypt()`, por lo que realizar después la validación de la contraseña contra el valor proporcionado por el usuario es un proceso de dos pasos: primero, volver a cifrar el valor proporcionado por el usuario con la misma clave falseada utilizada en el proceso original de cifrado y luego comprobar si las dos cadenas cifradas coinciden. Esto es precisamente lo que hace el formulario del ejemplo cuando procesa la información.

Conmutar a una base de datos diferente

Ahora, supongamos que por causas de fuerza mayor te ves forzado a cambiar de MySQL a SQLite. Lo primero que necesitas hacer es crear una tabla en la base de datos para almacenar toda la información de las cuentas de usuario. Así que arranca tu programa cliente SQLite y replica la base de datos creada en la sección anterior (nombra el archivo de base de datos *app.db*):

```
sqlite> CREATE TABLE usuarios (
     -> id INTEGER PRIMARY KEY,
     -> nombredeusuario TEXT NOT NULL,
     -> contrasena TEXT NOT NULL,
     -> );

sqlite> INSERT INTO usuarios (nombredeusuario, contrasena)
     -> VALUES ('john', '$1$Tk0.gh4.$42EZDbQ4mOfmXMq.0m1tS1');
sqlite> INSERT INTO usuarios (nombredeusuario, contrasena)
     -> VALUES ('jane', '$1$.15.tR/.$XK1KW1Wzqy0UuMFQDHH00');
```

El siguiente paso consiste en actualizar el código de tu aplicación para que se comunique con esta base de datos en lugar de hacerlo con la base de MySQL. Como estás utilizando PDO, este cambio consiste en cambiar una (así es, *una*) línea en tu script PHP: el DSN transmitido al constructor del objeto PDO. He aquí el segmento relevante del código:

```
<?php
     // intenta establecer conexión con la base de datos
     try {
```

www.computrucismania.com

```
        $pdo = new PDO('sqlite:app.db');
      } catch (PDOException $e) {
         die("Error: No fue posible conectar: " . $e->getMessage());
      }
?>
```

Y ahora, cuando ingreses, tu aplicación trabajará exactamente como lo hizo antes, excepto que ahora utilizará la base de datos SQLite en lugar de MySQL. ¡Inténtalo por ti mismo y lo verás!

Esta portabilidad es precisamente la razón por la que PDO está ganando popularidad entre los desarrolladores de PHP; hace que todo el proceso de conmutar entre bases de datos sea demasiado sencillo, reduciendo tanto el desarrollo y el tiempo de prueba, como los costos asociados a esas tareas.

Resumen

El soporte para bases de datos de PHP es una de las grandes razones de su popularidad, y este capítulo cubrió todo el terreno necesario para que comiences a trabajar con esta importante característica del lenguaje. El capítulo comenzó presentándote una introducción a los conceptos básicos de las bases de datos, y enseñándote las bases del lenguaje estructurado de consultas (SQL). Pasamos rápidamente a presentar las extensiones PHP para bases de datos más populares: las extensiones MySQLi y las extensiones SQLite, además de la extensión de objetos de datos de PHP (PDO). Los temas abordados incluyeron información práctica y ejemplos de código para realizar consultas y modificaciones a las bases de datos con PHP, con el fin de que comiences a utilizar estas extensiones en tu programación diaria.

Para leer más acerca de los temas abordados en este capítulo, te recomiendo que visites los siguientes vínculos:

- Conceptos de bases de datos, en **www.melonfire.com/community/columns/trog/article.php?id=52**
- Bases de SQL, en **www.melonfire.com/community/columns/trog/article.php?id=39** y **www.melonfire.com/community/columns/trog/article.php?id=44**
- Más información sobre fusiones SQL, en **www.melonfire.com/community/columns/trog/article.php?id=148**
- El sitio Web de MySQL, en **www.mysql.com**
- El sitio Web de SQLite, en **www.sqlite.org**
- Extensiones PHP MySQLi, en **www.php.net/mysqli**
- Extensiones PHP SQLite, en **www.php.net/sqlite**
- Extensiones PHP PDO, en **www.php.net/pdo**

www.compumusicmania.com

Autoexamen Capítulo 7

1. Marca las siguientes declaraciones como verdaderas o falsas:

- **A** Las fusiones de tablas con SQL sólo se realizan entre los campos de llave primaria y llave externa.
- **B** El soporte de PHP para MySQL es reciente.
- **C** La cláusula `ORDER BY` es utilizada para ordenar los campos de una colección de resultados SQL.
- **D** Los campos `PRIMARY KEY` pueden aceptar valores nulos.
- **E** Es posible reescribir el valor generado por SQLite para un campo `INTEGER PRIMARY KEY`.
- **F** Las declaraciones preparadas pueden utilizarse únicamente para las operaciones `INSERT`.

2. Identifica correctamente el comando SQL para cada una de las siguientes operaciones de base de datos:

- **A** Borrar una base de datos.
- **B** Actualizar un registro.
- **C** Borrar un registro.
- **D** Crear una tabla.
- **E** Seleccionar una base de datos para ser utilizada.

3. ¿Qué significa normalizar una base de datos y por qué es útil?

4. Menciona una ventaja y una desventaja de utilizar una biblioteca de abstracción como PDO en lugar de utilizar extensiones nativas de la base de datos.

5. Utilizando la extensión PDO, escribe un script PHP para añadir nuevas canciones a la tabla *canciones* desarrollada en este capítulo. Permite que los usuarios seleccionen el artista de la canción y el rating de una lista de selección desplegable, cuyo contenido sea alimentado por las tablas *artista* y *ratings*.

6. Utilizando la extensión MySQLi de PHP, escribe un script para crear una nueva tabla de base de datos con cuatro campos que tú selecciones. Después, realiza la misma tarea con una base de datos SQLite utilizando la extensión SQLite de PHP.

7. Alimenta manualmente la base de datos MySQL creada en la pregunta anterior con 7 o 10 registros de tu elección. Después, escribe un script basado en PDO que lea el contenido de esta tabla y que migre el contenido encontrado en ella a la tabla de la base de datos SQLite también creada en la pregunta anterior. Utiliza una declaración preparada.

www.compumusicmania.com

Capítulo 8

Trabajar con XML

www.compumusicmania.com

Habilidades y conceptos clave

- Comprender las tecnologías y los conceptos básicos de XML
- Comprender las extensiones de PHP SimpleXML y DOM
- Acceder a documentos XML con PHP y manipularlos
- Crear nuevos documentos XML desde cero usando PHP
- Integrar informes RSS de terceros en una aplicación PHP
- Convertir entre SQL y XML utilizando PHP

El lenguaje de marcado extensible (XML, eXtensible Markup Language) es un estándar muy aceptado para intercambio y descripción de datos. Permite que los autores de contenidos "marquen" sus datos con etiquetas personales comprensibles para las computadoras, y por lo mismo, facilita la clasificación y búsqueda de los datos. XML también obliga a dar una estructura formal al contenido, y proporciona un formato portátil que se utiliza para intercambiar información fácilmente entre diferentes sistemas.

PHP incluye soporte a XML desde la versión 4, pero es sólo en la versión 5 cuando las diferentes extensiones XML fueron estandarizadas en PHP para utilizar herramientas de segmentación comunes. Este capítulo te presenta una introducción a dos de las más útiles y poderosas extensiones de procesamiento XML de PHP: SimpleXML y DOM, e incluye numerosos ejemplos de código y ejercicios prácticos para utilizar XML en combinación con las aplicaciones basadas en PHP.

Introducción a XML

Antes de entrar en los detalles de la manipulación de archivos XML con PHP, resultará instructivo ocupar un tiempo familiarizándose con XML. Si eres nuevo en XML, la siguiente sección presenta los fundamentos básicos, incluida una visión panorámica de sus conceptos y tecnologías. Esta información será de utilidad para comprender el material más avanzado en las siguientes secciones.

Aspectos básicos de XML

Comencemos con una pregunta muy básica: ¿qué es XML y por qué es útil?

XML es un lenguaje que ayuda a los autores de documentos a describir datos dentro de éstos, al "marcarlos" con etiquetas personales. Estas etiquetas no provienen de una lista predefinida; en lugar de ello, XML alienta a los autores a crear sus propias etiquetas y su propia

www.compumusicmania.com

estructura, acoplada a sus necesidades, como una manera de incrementar la flexibilidad y capacidad de uso. Este hecho, junto con las recomendaciones de W3C en 1998, han servido para hacer de XLM una de las maneras más populares para describir y almacenar información estructurada dentro y fuera de Web.

Los datos XML están almacenados en archivos de texto. Esto hace que esos documentos sean muy portátiles, porque todas las computadoras pueden leer y procesar archivos de texto. Esto no sólo facilita la distribución de información, también permite que XML se utilice en una gran cantidad de aplicaciones. Por ejemplo, los formatos de RSS y Atom Weblog son alimentados mediante una estructura XML, lo mismo que JavaScript y XML asincrónico (AJAX, Asynchronous JavaScript and XML) y el protocolo de acceso a objetos simples (SOAP, Simple Object Access Protocol).

Pregunta al experto

P: ¿Qué programas puedo utilizar para crear o ver un archivo XML?

R: En sistemas UNIX/Linux, tanto `vi` como `emacs` son útiles para crear documentos XML, mientras que el Bloc de notas sigue siendo el favorito en los sistemas Windows. Tanto Microsoft Internet Explorer como Mozilla Firefox tienen soporte integrado para XML y pueden leer y desplegar un documento XML como vista de árbol jerárquico.

Anatomía de un documento XML

Internamente, un documento XML está integrado por varios componentes, cada uno de ellos sirve a un propósito específico. Para comprender estos componentes, examina el siguiente documento XML, que contiene una receta para cocinar espagueti a la boloñesa:

```
 1.  <?xml version='1.0'?>
 2.  <receta>
 3.    <ingredientes>
 4.      <ingrediente cantidad="250" unidades="gm">Carne de res picada</
ingrediente>
 5.      <ingrediente cantidad="200" unidades="gm">Cebollas</ingrediente>
 6.      <ingrediente cantidad="75" unidades="ml">Vino rojo</ingrediente>
 7.      <ingrediente cantidad="12">Tomates</ingrediente>
 8.      <ingrediente cantidad="2" unidades="tbsp">Queso parmesano</ingre-
diente>
 9.      <ingrediente cantidad="200" unidades="gm">Espagueti</ingrediente>
10.    </ingredientes>
11.    <método>
```

www.compumusicmania.com

```
12.       <paso número="1">Corte y fría las cebollas.</paso>
13.       <paso número="2">Añada la carne picada a las cebollas fritas
&amp; continúe friendo.</paso>
14.       <paso número="3">Haga puré los tomates y combínelos con la mezcla
junto con el vino rojo.</paso>
15.       <paso número="4">Hiérvalos durante una hora.</paso>
16.       <paso número="5">Sírvalos encima de una pasta cocinada con queso
parmesano.</paso>
17.     </método>
18.   </receta>
```

Este documento XML contiene una receta, dividida en diferentes secciones; cada una de ellas está "marcada" con etiquetas descriptivas para identificar con toda precisión el tipo de dato que contiene en su interior. Veamos cada una en detalle.

1. Cada documento XML debe iniciar con una declaración que especifique la versión XML que se está utilizando; esta declaración es conocida como *prólogo del documento*, y puede verse en la línea 1 del documento XML anterior. Además del número de la versión, este prólogo del documento puede contener información sobre la codificación de caracteres y la referencia a la definición del tipo de documento (DTD) (para la validación de datos).
2. El prólogo del documento es seguido por una serie anidada de *elementos* (líneas 2 a la 18). Los elementos son básicamente unidades de XML; por lo general, están integrados por un par de etiquetas, una de apertura y otra de cierre, que incluyen cierto contenido en forma de texto. Los nombres de los elementos son definidos por el usuario; deben elegirse con cuidado, porque su intención es describir el contenido que se encuentra entre ellos. Los nombres de los elementos son sensibles a las mayúsculas y deben iniciar con una letra, opcionalmente seguidos por más letras o números. El primer elemento en un documento XML (en el ejemplo anterior, el elemento llamado `<receta>` que ocupa la línea 2) es conocido como *elemento del documento* o *elemento raíz*.
3. Los datos en forma de texto encerrados dentro de los elementos son conocidos, en la terminología de XML, como *datos literales*. Son cadenas de caracteres, números y caracteres especiales (con algunas excepciones: los corchetes angulados(< >) y los signos de unión (&) dentro del texto deben reemplazarse con sus códigos `&lt;`, `&gt;` y `&amp;`, respectivamente, para evitar confusiones en el analizador sintáctico XML cuando lea el documento). La línea 13, por ejemplo, utiliza el código `&amp;` para representar el signo de unión dentro de los datos literales.
4. Por último, los elementos también pueden contener *atributos*, que son pares nombre-valor que contienen información adicional sobre el elemento. Los nombres de los atributos son sensibles a las mayúsculas y siguen las mismas reglas que los de elementos. El mismo

www.compumusicamania.com

nombre de atributo no puede utilizarse más de una vez en el mismo elemento, y los valores del atributo deben ir siempre encerrados entre comillas dobles. De la línea 4 a la 9 y de la 12 a la 16 en el documento de ejemplo se muestra el uso de los atributos para proporcionar información descriptiva adicional sobre el elemento al que están adjuntados; por ejemplo, el atributo '`unidades`' especifica la unidad de medida para cada ingrediente.

Los elementos XML también pueden contener otros componentes: nombres de espacios, instrucciones de procesamiento y bloques CDATA. Éstos son un poco más complejos y no los verás en ninguno de los ejemplos utilizados en este capítulo; sin embargo, si estás interesado en saber más sobre ellos, visita los sitios recomendados al final de este capítulo para conocer mayor información al respecto y para conseguir algunos ejemplos.

Pregunta al experto

P: ¿Puedo crear elementos que no contengan nada?

R: Seguro. La especificación XML da soporte a elementos sin contenido y, por lo mismo, no requiere una etiqueta de cierre. Para cerrar estos elementos, simplemente añade una diagonal al final de la etiqueta de apertura, como en el siguiente código:

```
La línea se rompe <br /> aquí.
```

XML bien formado y válido

La especificación XML hace una importante distinción entre los documentos bien formados y los válidos.

- Un *documento bien formado* sigue todas las reglas para los nombres de elementos y atributos, contiene todas las declaraciones esenciales, incluye un (y sólo un) elemento raíz y sigue correctamente la jerarquía de elementos anidados debajo de este elemento. Todos los documentos XML que verás en este capítulo son documentos bien formados.
- Un *documento válido* es aquel que respeta todas las condiciones de un documento bien formado y además se apega a las reglas adicionales expresadas en una definición de tipo de documento (DTD) o esquema XML. Este capítulo no aborda ni las DTD ni los esquemas XML en detalle, por lo que no verás ningún ejemplo de este tipo de documento; sin embargo, encontrarás muchos ejemplos en línea y en los vínculos sugeridos al final de este capítulo.

www.compumusicmania.com

Métodos de segmentación de XML

Por lo general, un documento XML es procesado por un programa de aplicación llamado analizador sintáctico XML. Éste lee el documento XML usando uno de dos métodos: simple API para XML (SAX) o el modelo de objeto de documento (DOM, Document Object Model):

- Un analizador sintáctico SAX opera recorriendo de manera secuencial un documento XML de principio a fin, e invocando funciones específicas definidas por el usuario mientras encuentra diferentes tipos de constructores XML. Así, por ejemplo, un analizador sintáctico SAX puede estar programado para invocar una función que procese un atributo, otro para procesar un elemento de arranque y un tercero para procesar los datos literales. Las funciones invocadas de esta manera son las responsables de procesar, en realidad, el constructor XML encontrado y cualquier información almacenada ahí.
- Un analizador sintáctico DOM, por otra parte, funciona leyendo el documento XML completo de una sola vez y convirtiéndolo en un árbol jerárquico estructurado en la memoria. Luego, es posible programar el analizador sintáctico para recorrer el árbol, brincando entre las diferentes "ramas" para acceder a piezas de información específicas.

Ambos métodos tienen pros y contras: SAX lee los datos XML en "fragmentos" y es eficiente para archivos grandes, pero requiere que el programador cree funciones personalizadas para manejar los diferentes elementos en un archivo XML. DOM requiere menos personalización, pero puede consumir mucha memoria en poco tiempo por sus acciones y, por lo mismo, no es recomendable para documentos XML grandes. La elección de uno u otro método depende en gran medida de las necesidades de la aplicación de que se trate.

Tecnologías XML

Al tiempo que aumenta la popularidad del lenguaje XML, también lo hace la lista de tecnologías que lo utilizan. He aquí unas cuantas de ellas de las que seguramente has escuchado:

- **Esquema XML** Define la estructura y el formato de documentos XML, permitiendo mayor flexibilidad en la validación y soporte de los tipos de datos, la herencia, el agrupamiento y la vinculación con bases de datos.
- **XLink** Especificación para vincular entre sí estructuras de datos XML. Permite la aplicación de tipos de vínculos más sofisticados que los hipervínculos regulares de HTML, incluidos vínculos con varias etiquetas.
- **XPointer** Especificación para navegar por el árbol jerárquico estructurado de un documento XML, para localizar con facilidad elementos, atributos y otras estructuras de datos dentro del documento.

www.compumusicmania.com

- **XLS** El Lenguaje extensible de hojas de estilo (XLS, eXtensible Styleshet Language) aplica reglas de formato a los documentos XML y los "transforma" de un formato a otro.
- **XHTML** Combina la precisión de marcado XML con las sencillas etiquetas HTML para crear una versión nueva y más estándares de compilación para HTML.
- **XForms** Separa la información recopilada en un formulario Web de la apariencia del formulario, con lo que permite una validación más rígida, además de que permite el fácil reciclaje de formularios en diferentes medios.
- **XML Query** Permite que los desarrolladores apliquen consultas a un documento XML y generen colecciones de resultados, de manera muy similar a la aplicación de SQL para recuperar registros de una base de datos.
- **XML Encryption y XML Signature** La primera representa una serie de medios para cifrar y descifrar documentos XML, además de representar los datos resultantes. Está relacionada muy estrechamente con la segunda, XML Signature, que proporciona un medio para representar y verificar firmas digitales con XML.
- **SVG** Scalable Vector Graphics o imágenes vectoriales escalables utiliza XML para describir vectores o convertir información en imágenes, con soporte para máscaras alfa, filtros, rutas de acceso y transformaciones.
- **MathML** Utiliza XML para describir expresiones o fórmulas matemáticas, con el fin de representarlas fácilmente en exploradores Web.
- **SOAP** El protocolo de acceso a objetos simples utiliza XML para codificar solicitudes y respuestas entre servidores anfitrión utilizando HTTP.

Pregunta al experto

P: ¿Cuándo debo utilizar un atributo y cuándo un elemento?

R: Tanto los atributos como los elementos contienen datos descriptivos, por lo que es cuestión de criterio decidir si es mejor almacenar una pieza de información en particular como elemento o como atributo. En casi todos los casos, si la información está estructurada jerárquicamente, los elementos son contenedores más apropiados; por otra parte, los atributos son mejores para información subordinada o que no tiene en sí una estructura formal.

Para encontrar un análisis formal sobre el tema, visita el artículo de IBM, developerWorks (trabajo de desarrollo) en www.ibm.com/developerswork/xml/library/x-eleatt.html, que aborda el tema con gran detalle.

www.compumusicmania.com

Prueba esto 8-1 Crear un documento XML

Ahora que conoces las bases de XML, pongamos esos conocimientos en práctica creando un documento XML bien formado y viéndolo en un explorador Web. Este documento presentará una colección de libros utilizando XML. Cada parte del documento contendrá información sobre título, autor, género y número de páginas de cada volumen.

Para crear este documento XML abre tu editor de texto favorito y escribe el siguiente código (*biblioteca.xml*):

```
<?xml version="1.0"?>
<biblioteca>
  <libro id="1" genero="horror" rating="5">
    <titulo>The Shining</titulo>
    <autor>Stephen King</autor>
    <paginas>673</paginas>
  </libro>
  <libro id="2" genero="suspenso" rating="4">
    <titulo>Shutter Island</titulo>
    <autor>Dennis Lehane</autor>
    <paginas>390</paginas>
  </libro>
  <libro id="3" genero="fantasía" rating="5">
    <titulo>The Lord of The Rings</titulo>
    <autor>J. R .R. Tolkien</autor>
    <paginas>3489</paginas>
  </libro>
  <libro id="4" genero="suspenso" rating="3">
    <titulo>Double Cross</titulo>
    <autor>James Patterson</autor>
    <paginas>308</paginas>
  </libro>
  <libro id="5" genero="horror" rating="4">
    <titulo>Ghost Story</titulo>
    <autor>Peter Straub</autor>
    <paginas>389</paginas>
  </libro>
  <libro id="6" genero="fantasía" rating="3">
    <titulo>Glory Road</titulo>
    <autor>Robert Heinlein</autor>
    <paginas>489</paginas>
  </libro>
  <libro id="7" genero="horror" rating="3">
    <titulo>The Exorcist</titulo>
    <autor>William Blatty</autor>
    <paginas>301</paginas>
  </libro>
  <libro id="8" genero="suspenso" rating="2">
```

www.compumusicmania.com

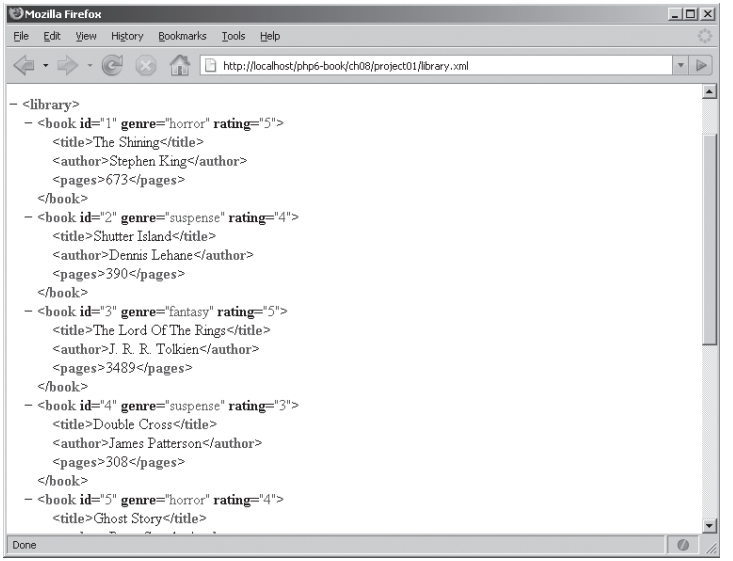


Figura 8-1 Documento XML, desplegado en Mozilla Firefox

```
    <titulo>The Camel Club</titulo>
    <autor>David Baldacci</autor>
    <paginas>403</paginas>
  </libro>
</biblioteca>
```

Guarda este archivo en la carpeta raíz para documentos de tu servidor Web y asígnale el nombre de *biblioteca.xml*. A continuación, abre tu explorador Web y escribe el URL correspondiente a la ubicación del archivo. Debes ver algo semejante a lo que aparece en la figura 8-1.

Utilizar las extensiones SimpleXML de PHP

Aunque PHP da soporte a DOM y SAX como métodos de segmentación, la manera más fácil de trabajar con XML en PHP es, por mucho, la extensión SimpleXML. Esta extensión, que está disponible por defecto en la versión 5 de PHP, proporciona una interfaz de usuario amigable e intuitiva para leer y procesar datos XML en aplicaciones PHP.

Aunque es posible guardar el texto con formato Unicode y hacer que el analizador sintáctico de todos los exploradores lo respete, aquí nos apegaremos a la práctica de no incluir acentos ni eñes, para evitar cualquier conflicto con la recuperación de datos en PHP. Si deseas usarlos, puedes recurrir a caracteres de escape para representarlos, pero su uso está más allá del alcance de este libro.

www.computrucionario.com

Trabajar con elementos

SimpleXML representa cada documento XML como un objeto y convierte sus elementos internos en un conjunto jerárquico de objetos y propiedades de objeto. Accesar un elemento se convierte así en cuestión de utilizar la notación `principal->secundario` para recorrer el árbol de objetos hasta alcanzar el elemento buscado.

Para mostrar cómo funciona en la práctica, examina el siguiente archivo XML (*dirección.xml*):

```
<?xml version='1.0'?>
<dirección>
  <calle>13 High Street</calle>
  <condado>Oxfordshire</condado>
  <ciudad>
    <nombre>Oxford</nombre>
    <cp>OX1 1BA</cp>
  </ciudad>
  <país>UK</país>
</dirección>
```

He aquí un script PHP que utiliza SimpleXML para leer este archivo y recuperar el nombre de la ciudad y el código postal:

```
<?php
// carga archivo XML
$xml = simplexml_load_file('dirección.xml') or die ("No fue posible
cargar XML!");

// accesa datos XML
// datos de salida: 'Ciudad: Oxford \n Código Postal: OX1 1BA\n'
echo "Ciudad: " . $xml->ciudad->nombre . "\n";
echo "Código Postal:" . $xml->ciudad->cp . "\n";
?>
```

Para leer un archivo XML con SimpleXML, utiliza la función `simplexml_load_file()` y pasa la ruta de acceso al disco del archivo como argumento. Entonces, esta función leerá e interpretará el archivo XML y, dando por hecho que está bien formado, regresará un objeto SimpleXML representando el elemento raíz del documento. Este objeto es sólo el nivel superior del árbol jerárquico, que es un espejo de la estructura interna de los datos XML: los elementos debajo del elemento raíz son representados como propiedades u objetos secundarios y así pueden accederse utilizando la notación `objetoPrincipal->objetoSecundario`.

TIP

Si tus datos XML están en una variable de cadena de caracteres y no en un archivo, utiliza la función `simplexml_load_string()` para leerla en un objeto SimpleXML.

www.compumusicmania.com

Diferentes instancias del mismo elemento, ubicadas en el mismo nivel del documento dentro del árbol XML, son representadas como matrices. Es posible procesar estas instancias fácilmente utilizando un constructor PHP de bucles. Para aclararlo, examina el siguiente ejemplo, que lee el archivo *biblioteca.xml*, desarrollado en la sección anterior, y presenta los títulos y nombre de autores que encuentra en él:

```
<?php
// carga archivo XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");
// reitera sobre los datos XML como una matriz
// presenta los títulos y autores de los libros
// datos de salida: 'The Shining fue escrito por Stephen King. \n...'
foreach ($xml->libro as $libro) {
  echo $libro->titulo . " fue escrito por " . $libro->autor . ".\n";
}
?>
```

Aquí, un bucle `foreach` itera sobre los objetos `<libro>` generados a partir de los datos XML, presentando las propiedades '`titulo`' y '`autor`' de cada uno.

También puedes contar la cantidad de elementos de un nivel en particular en el documento XML invocando la función `count()`. El siguiente código lo ejemplifica, contando la cantidad de `<libros>` en el documento XML:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");

// recorre en bucle los datos XML como una matriz
// presenta la cantidad de libros
// datos de salida: '8 libro(s) encontrados.'
echo count($xml->libro) . ' libro(s) encontrados.';
?>
```

Trabajar con atributos

Si un elemento XML contiene atributos, SimpleXML cuenta con un medio para recuperarlos con igual facilidad: los atributos y los valores son transformados en llaves y valores de una matriz asociativa PHP y pueden accederse como elementos regulares de la matriz.

Para dejarlo en claro, analiza el siguiente ejemplo, que lee el archivo *biblioteca.xml* de la sección anterior y presenta cada título encontrado, junto con su '`genero`' y '`rating`':

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");
```

www.compumusicamania.com

```
// accesa los datos XML
// para cada libro
// recupera y presenta los atributos 'genero' y 'rating'
// datos de salida: 'The Shining \n Género: horror \n Rating: 5 \n\n...'
foreach ($xml->libro as $libro) {
  echo $libro->título . "\n";
  echo "Género: " . $libro['genero'] . "\n";
  echo "Rating: " . $libro['rating'] . "\n\n";
}
?>
```

En este ejemplo, un bucle `foreach` itera sobre el elemento `<libro>` en los datos XML, convirtiendo cada uno de ellos en un objeto. Los atributos del elemento libro se representan como elementos de una matriz asociativa, de modo que una llave puede acceder a ellos: la llave '`genero`' regresa el valor del atributo '`genero`' y la llave '`rating`' regresa el valor del atributo '`rating`'.

Prueba esto 8-2 Convertir XML a SQL

Ahora que sabes leer elementos XML y atributos, veamos un ejemplo práctico de SimpleXML en acción. El siguiente programa lee un archivo XML y convierte los datos que contiene en una serie de declaraciones SQL, que pueden ser utilizadas para transferir los datos a MySQL o cualquier otro compilador de base de datos.

He aquí el archivo ejemplo XML (*inventario.xml*):

```
<?xml version='1.0'?>
<items>
  <item sku="123">
    <nombre>Queso cheddar</nombre>
    <precio>3.99</precio>
  </item>
  <item sku="124">
    <nombre>Queso azul</nombre>
    <precio>5.49</precio>
  </item>
  <item sku="125">
    <nombre>Tocino ahumado (paquete de 6 piezas)</nombre>
    <precio>1.99</precio>
  </item>
  <item sku="126">
    <nombre>Tocino ahumado (paquete de 12 piezas)</nombre>
    <precio>2.49</precio>
  </item>
  <item sku="127">
    <nombre>Paté de ganso</nombre>
    <precio>7.99</precio>
```

www.compumusicamania.com

```
  </item>
  <item sku="128">
    <nombre>Paté de pato</nombre>
    <precio>6.49</precio>
  </item>
</items>
```

Y aquí está el código PHP que convierte estos datos XML en declaraciones SQL (*xmlAsql.php*):

```
<?php
// carga el documento XML
$xml = simplexml_load_file('inventario.xml') or die ("No fue posible
cargar XML!");

// recorre en bucle los elementos XML <item>
// accede a nodos secundarios e interpola con declaraciones SQL
foreach($xml as $item) {
  echo "INSERT INTO ítems (sku, nombre, precio) VALUES ('".
addslashes($item['sku']) . "','" . addslashes($item->nombre) . "','" .
addslashes($item->precio) . "');\n";
}
?>
```

Este script debe ser fácil de comprender si has seguido las lecciones: itera sobre todos los elementos <item> del documento XML, utilizando la notación `objeto->propiedad` para acceder a cada elemento <nombre> y <precio> de item. Se accede al atributo 'sku' de cada <item> de manera similar con la llave 'sku' de cada atributo item de la matriz. Los valores recuperados de esta manera son entonces interpolados en una declaración SQL `INSERT`.

Entonces, esta declaración puede ser proporcionada de manera normal a una función como `mysql_query()` o `sqlite_query()` para insertarlas en una base de datos MySQL o SQLite; para este ejemplo, es más sencillo desplegarlas en pantalla.

La figura 8-2 muestra los datos de salida de este script.

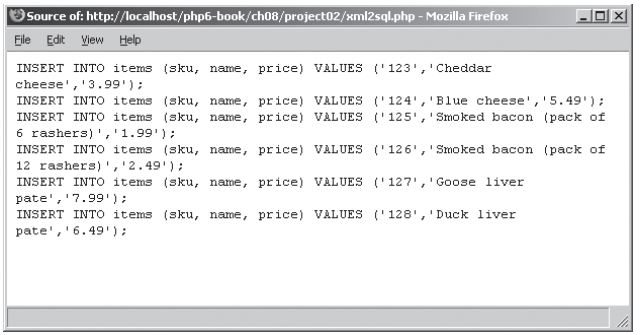


Figura 8-2 Conversión de XML a SQL con SimpleXML

www.compumusicmania.com

Alterar elementos y valores de atributos

Con SimpleXML es fácil cambiar el contenido en un archivo XML: simplemente asigna un nuevo valor a la propiedad correspondiente del objeto utilizando el operador PHP de asignación (=). Para comprenderlo, examina el siguiente script PHP, que modifica el título y el autor del segundo libro en *biblioteca.xml* y después presenta el documento XML modificado:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("¡No fue posible
cargar XML!");

// cambia valores de elementos
// escribe un nuevo título y autor para el segundo libro
$xml->libro[1]->titulo = 'Invisible Prey';
$xml->libro[1]->autor = 'John Sandford';

// datos de salida de la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Aquí, se utiliza SimpleXML para acceder al segundo elemento `<libro>` por su índice, y los valores de los elementos `<titulo>` y `<autor>` son alterados suministrando nuevos valores para sus correspondientes propiedades de objeto. Pon atención al método `asXML()`, que es nuevo en este ejemplo: convierte la jerarquía anidada de objetos SimpleXML y las propiedades de los objetos en una cadena estándar de XML.

Cambiar los valores de los atributos es igual de fácil: asigna un valor nuevo a la llave correspondiente del atributo en la matriz. He aquí un ejemplo, que cambia el `'rating'` del sexto libro y presenta el resultado:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");

// cambia los valores de atributos
// escribe un nuevo rating para el sexto libro
$xml->libro[5]{'rating'} = 5;

// muestra la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

www.compumusicamania.com

Añadir nuevos elementos y atributos

Además de modificar los valores de elementos y atributos, SimpleXML también te permite añadir dinámicamente nuevos elementos y atributos a un documento XML existente. Para ejemplificarlo, examina el siguiente script, que añade un nuevo <libro> a los datos XML de *biblioteca.xml*:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");

// obtiene el último 'id' de los libros
$numLibros = count($xml->libro);
$lastID = $xml->libro[($numLibros-1)]{'id'};

// añade un nuevo elemento <libro>
$libro = $xml->addChild('libro');

// obtiene el atributo 'id'
// para el nuevo elemento <libro>
// incrementando 1 a $lastID
$libro->addAttribute('id', (lastID+1));

// añade atributos 'rating' y 'genero'
$libro-> addAttribute('genero', 'viajes');
$libro-> addAttribute('rating', 5);

// añade elementos <titulo>, <autor>, <página>
$título = $libro->addChild('titulo', 'Frommer\'s Italy 2007');
$autor = $libro->addChild('autor', 'Varios');
$página = $libro->addChild('paginas', 820);

// muestra la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Cada objeto SimpleXML presenta un método `addChild()`, para añadir nuevos elementos secundarios, y un método `addAttribute()`, para añadir nuevos atributos. Ambos métodos aceptan un nombre y un valor, generando el elemento o atributo correspondiente, y adjuntándolo al objeto principal dentro de la jerarquía XML.

Estos métodos se ilustran en el ejemplo anterior, que comienza leyendo el documento XML existente en un objeto SimpleXML. El elemento raíz de este documento se almacena en el objeto `$xml` de PHP. Entonces, el programa necesita calcular el ID que será asignado al nuevo elemento <libro>, y lo hace contando la cantidad de elementos que ya están presentes en el documento XML, accede al último de esos elementos, recupera su atributo 'id' y le suma 1.

www.compumusicmania.com

Pasada esa formalidad, el programa se dedica a la creación de elementos y atributos:

1. Comienza adjuntando un nuevo elemento <libro> al elemento raíz, invocando el método addChild() del objeto $xml. Este método acepta el nombre del elemento que será creado y (opcionalmente) un valor para ese elemento. El objeto XML resultante se almacena en el objeto PHP $libro.
2. Con el elemento creado, es tiempo de establecer sus atributos 'id', 'genero' y 'rating'. Esto se hace con el método addAttribute() del objeto $libro, que también acepta dos argumentos: el nombre del atributo y su valor, y establece las correspondientes llaves de la matriz asociativa.
3. Una vez que el último elemento <libro> se ha definido por completo, es momento de añadir los elementos <titulo>, <autor> y <página> como secundarios del elemento <libro>. Esto se realiza fácilmente invocando de nuevo al método addChild(), esta vez del objeto $libro.
4. Una vez que los objetos secundarios están definidos, la jerarquía del objeto se transforma en una cadena del documento XML con el método asXML().

La figura 8-3 muestra el resultado.

Crear nuevos documentos XML

También puedes utilizar SimpleXML para crear un nuevo documento XML de la nada, inicializando un objeto SimpleXML vacío a partir de una cadena de caracteres XML y utilizando después los métodos addChild() y addAttribute() para construir el resto del árbol XML. Examina el siguiente ejemplo, que muestra el proceso:

```
<?php
// carga XML a partir de una cadena de caracteres
$xmlStr = "<?xml version='1.0'?><persona></persona>";
$xml = simplexml_load_string($xmlStr);

// añade atributos
$xml->addAttribute('edad', '18');
$xml->addAttribute('sexo', 'masculino');

// añade elementos secundarios
$xml->addChild('nombre', 'Juan Pineda');
$xml->addChild('fdn', '04-04-1989');

// añade Segundo nivel de elementos secundarios
$direccion = $xml->addChild('direccion');
$direccion->addChild('calle', '12 A Road');
$direccion->addChild('ciudad', 'Londres');
```

www.compumusicmania.com

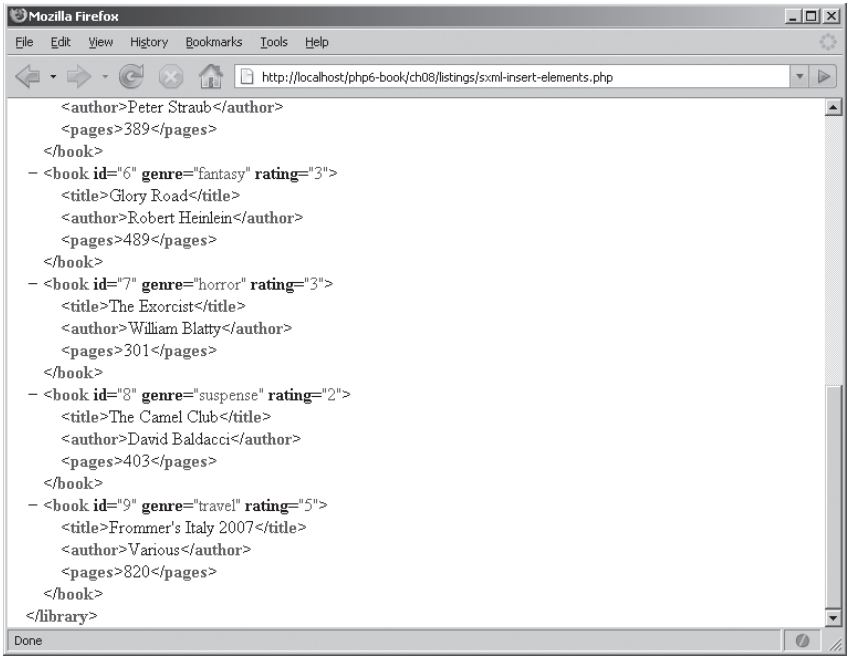


Figura 8-3 Insertar elementos en un árbol XML con SimpleXML

```
// añade un tercer nivel de elementos hijo
$pais = $direccion->addChild('pais', 'Reino Unido');
$pais->addAttribute('codigo', 'UK');

// muestra la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Este script PHP es similar a los que has visto en secciones anteriores, con una diferencia importante: en lugar de añadir nuevos elementos y atributos a un árbol XML ya existente, ¡este script lo genera por completo y de la nada!

El script comienza por inicializar una variable de cadena de texto para contener el prólogo y el elemento raíz del documento XML. El método `simplexml_load_string()` se encarga de convertir esta cadena en un objeto SimpleXML que representa el elemento raíz del documento. Una vez que este objeto se ha inicializado, sólo es cuestión de añadirle elementos secundarios y atributos, y construir el resto del árbol XML de manera programática. La figura 8-4 muestra el árbol XML resultante.

www.computeciencia.com

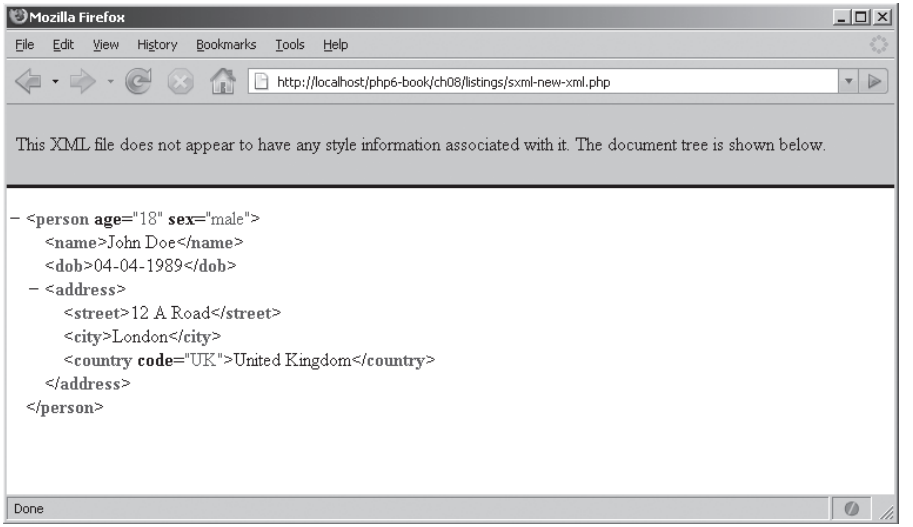


Figura 8-4 Generación dinámica de un nuevo documento XML con SimpleXML

Prueba esto 8-3 Leer informes RSS

RSS es un formato basado en XML, originalmente utilizado por Netscape para distribuir información sobre su contenido en el portal My.Netscape.com. Hoy en día, RSS es un medio muy popular en Web para distribuir información; muchos sitios Web ofrecen "informes" RSS que contienen vínculos y fragmentos de sus noticias más recientes y las actualizaciones de su contenido; casi todos los exploradores Web vienen con lectores RSS integrados, que se utilizan para leer o "suscribirse" a esos informes.

Un documento RSS sigue todas las reglas de marcado propias de XML y, por lo general, contiene una lista de recursos (URL), marcados con metadatos descriptivos. He aquí un ejemplo:

```
<?xml version='1.0' encoding="utf-8"?>
<rss>
  <channel>
    <title>El título del informe se escribe aquí</title>
    <link>El URL del informe se escribe aquí</link>
    <description>La descripción del informe ocupa este espacio</
description>
    <item>
```

www.computeciencia.com

```
      <title>Título de un tema particular</title>
      <description>Descripción del tema</description>
      <link>Liga al tema</link>
      <pubDate>Fecha de la publicación en formato de sello temporal</
       pubDate>
    </item>
    </item>
     ...
    <item>
  </channel>
</rss>
```

Como lo muestra este ejemplo, un documento RSS abre y cierra con un elemento `<rss>`. Un bloque `<channel>` contiene la información general sobre el sitio Web que proporciona el informe; esto es seguido por varios elementos `<item>`; cada uno de ellos representa una unidad de contenido diferente o una noticia particular. Todos los elementos `<item>` contienen su propio título, un URL y su respectiva descripción.

Dada esta estructura jerárquica bien definida, analizar sintácticamente el informe RSS con SimpleXML es de lo más sencillo. Eso es lo que hace el siguiente script: conecta un informe RSS activo a un URL anfitrión, recupera los datos del informe codificados en XML, los analiza y los convierte en una página HTML que puede desplegarse en cualquier explorador Web. He aquí el código (*rss2html.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 8-3: Lee un informe RSS</title>
    <style type="text/css">
    div.heading{
      font-weight: bolder;
    }
    div.story {
      background-color: white;
      border: 1px solid black;
      width: 320px;
      height: 200px;
      margin: 20px;
    }
    div.headline a {
      font-weight: bolder;
      color: orange;
      margin: 5px;
    }
    div.body {
      margin: 5px;
```

(*continúa*)

www.compumusicmania.com

```
    }
    div.timestamp {
      font-size: smaller;
      font-style: italic;
      margin: 5px;
    }
    ul {
       list-style-type: none;
    }
    li {
       float: left;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 8-3: Lee un informe RSS</h2>
<?php
// lee el informe RSS newsvine.com sobre avances tecnológicos
$xml = simplexml_load_file("http://www.newsvine.com/_feeds/rss2/
 tag?id=technology") or die ("ERROR: No es posible leer el informe RSS");
?>
    <h3 style="heading"><?php echo $xml->channel->title; ?></h3>
    <ul>
<?php
// reitera sobre la lista de noticias
// presenta el título de cada noticia, el URL y el sello cronológico
// y luego el cuerpo de la misma
foreach ($xml->channel->item as $item) {
?>
      <li>
        <div class="story">
          <div class="headline">
            <a href="<?php echo $item->link; ?>">
              <?php echo $item->title; ?>
            </a>
          </div>
          <div class="timestamp"><?php echo $item->pubDate; ?></div>
          <div class="body"><?php echo $item->description; ?></div>
        </div>
      </li>
<?php
}
?>
    </ul>
  </body>
</html>
```

Este script comienza utilizando el método `simplexml_load_file()` de SimpleXML para conectar un URL remoto (en este caso, un reporte RSS almacenado en NewsVi-

www.compumusicamania.com

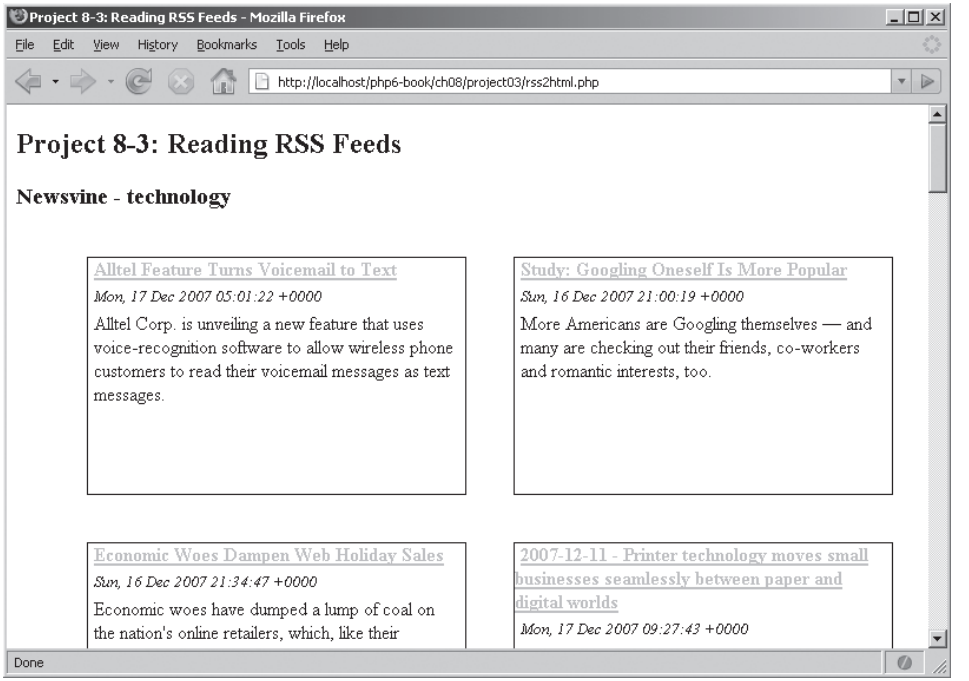


Figura 8-5 Interpretación de un informe RSS con SimpleXML

ne.com) y convertir el código XML encontrado ahí en un objeto SimpleXML. Después utiliza la capacidad de SimpleXML para hacer un bucle sobre la colección de nodos para recuperar rápidamente cada título, URL, sello cronológico y cuerpo de noticias; marca estas piezas de información con etiquetas HTML y las presenta en una página Web.

La figura 8-5 muestra los datos de salida.

Utilizar la extensión DOM de PHP

Ahora bien, la extensión de PHP SimpleXML es fácil de utilizar y comprender, pero no es útil para otra cosa que no sea el manejo básico de datos XML. Para operaciones más complejas con este lenguaje es necesario ver al siguiente paso, la extensión DOM de PHP. Esta extensión, que también está disponible por defecto desde la versión 5 de PHP, proporciona una serie de herramientas complejas que se ajusta con el estándar de nivel 3 DOM y proporciona a PHP capacidades completas para el análisis sintáctico de datos XML.

www.compumusicmania.com

Trabajar con elementos

El analizador sintáctico DOM funciona leyendo un documento XML y creando elementos que representen las diferentes partes del mismo. Cada uno de estos objetos incluye métodos y propiedades específicas, que pueden utilizarse para acceder a su información intrínseca y manipularla. De esta manera, todo el documento XML se representa como un "árbol" de estos objetos, y el analizador sintáctico DOM proporciona una sencilla API para moverse por las diferentes ramas del árbol.

Para mostrar la manera en que funciona, revisemos el archivo *direccion.xml* de la sección anterior:

```
<?xml version='1.0'?>
<direccion>
  <calle>13 High Street</calle>
  <condado>Oxfordshire</condado>
  <ciudad>
    <nombre>Oxford</nombre>
    <cp>OX1 1BA</cp>
  </ciudad>
  <pais>UK</pais>
</direccion>
```

A continuación aparece el script PHP que utiliza la extensión DOM para analizar sintácticamente este archivo y recuperar varios componentes de la dirección:

```
<?php
// inicializa el nuevo documento
$doc = new DOMdocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('direccion.xml');

// obtiene el elemento raíz
$root = $doc->firstChild;
// obtiene el nodo 'UK'
echo "País: " . $root->childNodes->item(3)->nodeValue . "\n";

// obtiene el nodo 'Oxford'
echo "Ciudad: " . $root->childNodes->item(2)-> childNodes->item(0)
->nodeValue . "\n";
```

www.compumusicamania.com

```
// obtiene el nodo 'OX1 1BA'
echo "Código Postal: " . $root->childNodes->item(2)-> childNodes-
>item(1)->nodeValue . "\n";

// datos de salida: 'País: UK \n Ciudad: Ofxord \n Codigo Postal: OX1
1BA'
?>
```

A primera vista se nota claramente que ya no estamos en el territorio de SimpleXML. Con la extensión DOM de PHP el primer paso es siempre inicializar una instancia del objeto DOMDocument, que representa al documento XML. Una vez que este objeto se ha inicializado, puede utilizarse para analizar sintácticamente un archivo XML con su método `load()`, que acepta la ruta de acceso en disco del archivo XML de objetivo.

El resultado del método `load()` es un árbol que contiene objetos DOMNode, donde cada objeto presenta varias propiedades y métodos para acceder a sus nodos principal y secundarios. Por ejemplo, cada objeto DOMNode presenta la propiedad `parentNode`, que se utiliza para acceder al nodo principal que corresponde al nodo en uso, lo mismo que la propiedad `childNodes`, que regresa la colección de nodos secundarios pertenecientes al nodo en uso. De manera similar, cada objeto DOMNode también presenta las propiedades `nodeName` y `nodeValue`, utilizadas para acceder, obviamente, al nombre y el valor del nodo, respectivamente. De esta manera, resulta muy fácil navegar de nodo en nodo por el árbol, recuperando valores de nodo en cada estrato.

Para ilustrar el proceso, revisa con cuidado el ejemplo anterior. Una vez que el documento XML ha sido cargado [`load()`], invoca la propiedad `firstChild` del objeto DOMDocument, que regresa un objeto DOMNode que representa el elemento raíz `<direccion>`. Este objeto DOMNode, a su vez, cuenta con la propiedad `childNodes`, que regresa una colección con todos los elementos secundarios de `<direccion>`. Se accede a los elementos individuales de esta colección a través de su ubicación en el índice utilizando el método `item()`, donde el índice comienza a partir de 0. Estos elementos son representados de nuevo como objetos DOMNode; de tal manera que sus nombres y valores son accesibles a través de las propiedades `nodeName` y `nodeValue`.

Por tanto, el elemento `<pais>`, que es el cuarto de `<direccion>`, resulta accesible a través de la ruta de acceso `$root->childNodes->item(3)`, y el valor de este elemento, `'UK'`, es accesible a través de la ruta de acceso `$root->childNodes->item(3) ->nodeValue`. De manera similar, el elemento `<nombre>`, que es el primer hijo del elemento `<ciudad>`, es accesible a través de la ruta de acceso `$root->childNodes->item(2)->childNodes->item(0)`, y el valor `'Oxford'` es accesible a través de la ruta de acceso `$root->childNodes->item(2)->childNodes->item(0)-> nodeValue`.

www.compumusicamania.com

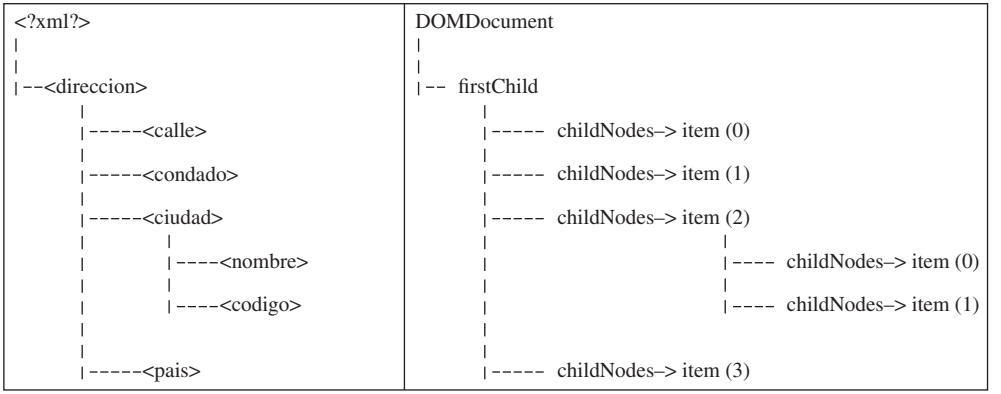


Figura 8-6 Relaciones DOM

La figura 8-6 debe aclarar estas relaciones; presenta un mapa del árbol XML *direccion.xml* con los métodos y propiedades DOM utilizados en esta sección.

Pregunta al experto

P: Cuando proceso un documento XML utilizando DOM, se muestran lo que parecen ser nodos de texto adicionales en cada una de las colecciones de nodos. Sin embargo, cuando accedo a estos nodos parece que están vacíos. ¿Qué está sucediendo?

R: Por la especificación DOM, todos los espacios en blanco del documento, incluyendo los retornos de carro, deben tratarse como nodos de texto. Si tu documento XML contiene espacios en blanco adicionales, o si tus elementos XML están limpiamente formados e incluyen sangrías con líneas separadas, esos espacios en blanco serán representados en tu colección de nodos como nodos de texto aparentemente vacíos. En la API DOM de PHP, puedes inhibir este comportamiento al establecer la propiedad `DOMDocument->preserveWhiteSpace` como `'false'`, como en los ejemplos de esta sección.

Un método opcional (y que resulta muy útil cuando te enfrentas a un árbol XML muy anidado) consiste en utilizar el método `getElementsByTagName()`, del objeto DOMDocument, para recuperar todos los elementos con un nombre en particular. Los datos de salida que presenta este método son una colección de objetos DOMNode que coinciden con ciertos criterios; de ahí es fácil hacer reiteraciones con un bucle `foreach` y recuperar el valor de cada nodo.

www.computrucionmania.com

Si resulta que tu documento sólo cuenta con una instancia de cada elemento (como en el caso de *direccion.xml*), utilizar el método `getElementsByTagName()` puede servir como un atajo efectivo en comparación con la navegación tradicional por el árbol XML. Examina el siguiente ejemplo, que produce los mismos datos de salida que el ejemplo anterior, sólo que en este caso se utiliza un atajo:

```
<?php
// inicializa el nuevo documento
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('direccion.xml');

// obtiene la colección de elementos <pais>
$pais = $doc->getElementsByTagName('pais');
echo "Pais: " . $pais->item(0)->nodeValue . "\n";

// obtiene la colección de elementos <nombre>
$ciudad = $doc->getElementsByTagName('ciudad');
echo "Ciudad: " . $ciudad->item(0)->nodeValue . "\n";

// obtiene la colección de elementos <cp>
$cp = $doc->getElementsByTagName('cp');
echo "Codigo Postal: " . $cp->item(0)->nodeValue . "\n";

// datos de salida: 'País: UK \n Ciudad: Oxford \n Código Postal: OXI IBA'
?>
```

En este ejemplo, el método `getElementsByTagName()` es utilizado para regresar una colección DOMNode que representa todos los elementos con el nombre `<pais>` en la primera instancia. Desde el árbol XML resulta claro que esta colección contendrá un solo objeto DOMNode. Para acceder al valor de este nodo basta con invocar al método `item()` de la colección con el argumento 0 (la primera posición en el índice) para traer el objeto DOMNode y después leer su propiedad `nodeValue`.

Sin embargo, en casi todos los casos tu documento XML no tendrá una sola instancia de cada elemento. Toma por ejemplo el archivo *biblioteca.xml* que estudiaste en secciones anteriores y que contiene varias instancias del elemento `<libro>`. Incluso en tales situaciones, el método `getElementsByTagName()` es útil para crear con eficiencia y rapidez un subgrupo de nodos que coincidan con cierto criterio, mismos que pueden procesarse con un bucle PHP. Para dejarlo en claro, examina el siguiente ejemplo, que lee el archivo *biblioteca.xml* y presenta los títulos y autores que encuentra:

www.compumusicamania.com

```
<?php
// inicializa el nuevo documento
$doc = new DOMdocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// el bucle foreach <libro> obtiene el valor de los elementos <titulo> y
<autor>
// datos de salida: 'The Shining fue escrito por Stephen King. \n ...'
$libros = $doc->getElementsByTagName('libro');
foreach ($libros as $libro){
  $titulo = $libro->getElementsByTagName('titulo')->item(0)->nodeValue;
  $autor = $libro->getElementsByTagName('autor')->item(0)->nodeValue;
echo "$titulo fue escrito por $autor. \n";
}
?>
```

En este caso, la primera invocación a `getElementsByTagName()` regresa una colección que representa todos los elementos `<libro>` del documento XML. Después es fácil hacer reiteraciones sobre esta colección con el bucle `foreach()`, procesando así cada objeto DOMNode y recuperando el valor correspondiente a los elementos `<titulo>` y `<autor>` con posteriores invocaciones a `getElementsByTagName()`.

TIP

Puedes recuperar una colección con todos los elementos de un documento con la invocación `DOMDocument -> getElementsByTagName(*)`.

Para saber cuántos elementos fueron regresados por `getElementsByTagName()`, utiliza la propiedad `length` de la colección resultante. He aquí un ejemplo:

```
<?php
// inicializa el nuevo documento
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;
// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// regresa un conteo del total de elementos <libro>
// datos de salida: '8 libro(s) encontrados.'
```

www.compumusicamania.com

```
$libros = $doc->getElementsByTagName('libro');
echo $libros->length . ' libro(s) encontrados.';
?>
```

Trabajar con atributos

DOM también incluye amplio soporte a los atributos: cada objeto DOMElement viene con un método `getAttribute()`, que acepta un nombre de atributo y regresa el valor respectivo. He aquí un ejemplo, que presenta cada `rating` y `genero` de libro del documento *biblioteca.xml*:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// por cada libro
// recupera y presenta los atributos 'genero' y 'rating'
// datos de salida: 'The Shining \n Genero: horror \n Rating: 5 \n\n ...'
$libros = $doc->getElementsByTagName('libro');
foreach ($libros as $libro) {
  $titulo = $libro->getElementsByTagName('titulo')->item(0)->nodeValue;
  $rating = $libro->getAttribute('rating');
  $genero = $libro->getAttribute('genero');
  echo "$titulo\n";
  echo "Género: $genero\n";
  echo "Rating: $rating\n\n";
}
?>
```

¿Qué sucede si no conoces el nombre del atributo y simplemente quieres procesar todos los atributos de un elemento? Bueno, cada DOMElement tiene una propiedad `attributes`, que regresa una colección con todos los atributos del elemento. Es fácil hacer reiteraciones sobre esta colección para recuperar cada `name` y `value` del atributo.

El siguiente ejemplo muestra el uso de esta propiedad, con una variación sobre el código anterior:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();
```

www.compumusicmania.com

```
// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// por cada libro
// recupera y presenta todos los atributos
// datos de salida: 'The Shining \n id: 1 \n genero: horror \n rating: 5
\n\n ...'
$libros = $doc->getElementsByTagName('libro');
foreach ($libros as $libro) {
  $titulo = $libro->getElementsByTagName('titulo')->item(0)->nodeValue;
  echo "$titulo\n";
  foreach ($libro->attributes as $attr) {
    echo "$attr->name: $attr->value \n";
  }
  echo "\n";
}
?>
```

Prueba esto 8-4 Procesar recursivamente un documento árbol de XML

Si planeas trabajar con XML y PHP en el futuro, con toda seguridad el siguiente proyecto te será de utilidad algún día: es un programa sencillo que comienza en la raíz del documento árbol XML y recorre todas sus ramas, procesando cada elemento y atributo que encuentra en el camino. Dada la naturaleza de tipo árbol del documento XML, la manera más eficiente de realizar esta tarea es con una función recursiva, y dada la riqueza informativa proporcionada por DOM, escribir la función es una tarea sencilla.

Supongamos por un momento que el documento XML que habrá de procesarse tiene el siguiente (*inventario.xml*):

```
<?xml version='1.0'?>
<objetos>
  <objeto color="rojo" forma="cuadrado">
  <longitud unidades="cm">5</longitud>
</objeto>
<objeto color="rojo" forma="círculo">
  <radio unidades="px">7</radio>
</objeto>
<objeto color="verde" forma="triángulo">
  <base unidades="in">1</base>
```

www.compumusicmania.com

```
  <altura unidades="in">2</altura>
  </objeto>
  <objeto color="azul" forma="triángulo">
    <base unidades="mm">100</base>
    <altura unidades="mm">50</altura>
  </objeto>
  <objeto color="amarillo" forma="círculo">
    <radio unidades="cm">18</radio>
  </objeto>
</objetos>
```

Y aquí está el codigo PHP para procesar recursivamente este (o cualquier otro) documento XML utilizando DOM:

```
<!DOCTYPE html PUBLIC "-//W3C// DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 8-4: Procesa recursivamente un documento XML</title>
  </head>
  <body>
    <h2>Proyecto 8-4: Procesa recursivamente un documento XML</h2>
    <pre>
<?php
// función recursiva para procesar una colección de nodos XML
function xmlProcess($node, $depthMarker) {

  // procesa los hijos de este nodo
  foreach ($node->childNodes as $n) {
    switch ($n->nodeType) {

      // para los elementos, presenta el nombre del elemento
      case XML_ELEMENT_NODE:
        echo "$depthMarker <b>$n->nodeName</b> \n";
        // si el elemento tiene atributos
        // lista sus nombres y valores
        if ($n->attributes->length > 0) {
          foreach ($n->attributes as $attr) {
            echo "$depthMarker <i>attr</i>: $attr->name => $attr->
value \n";
          }
      }
      break;
      // para datos de texto, presenta valores
      case XML_TEXT_NODE:
        echo "depthMarker <i>text</i>: \"$n->nodeValue\" \n";
```

(*continúa*)

www.compumusicamania.com

```
        break;
    }

    // si este nodo tiene un nivel inferior o subnodos
    // incrementa el marcador de profundidad
    // lo ejecuta recursivamente
    if ($n->hasChildNodes()) {
      xmlProcess($n, $depthMarker . DEPTH_CHAR);
    }
  }
}
// finaliza definición de función

// define el carácter utilizado para la indentación
define ('DEPTH_CHAR', ' ');

// inicializa DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('objetos.xml');

// invoca la función recursiva con el elemento raíz
xmlProcess($doc->firstChild, DEPTH_CHAR);
?>
    </pre>
  </body>
</html>
```

En este programa, la función personalizada `xmlProcess()` es una función recursiva que acepta un objeto DOMNode, recupera la colección de elementos secundarios de este nodo al leer la propiedad `childNodes` del objeto, e itera sobre esta colección con un bucle `foreach`. Si el nodo en uso es un nodo elemento, presenta el nombre del nodo y si es un nodo de texto, presenta su valor. En caso de que se trate de un nodo de elemento, el programa realiza un paso adicional para verificar los atributos y presentarlos, si es necesario. Utiliza una instrucción "cadena inferior" para indicar la posición jerárquica del nodo en los datos de salida; esta cadena se incrementa automáticamente cada vez que se ejecuta el bucle.

Cuando finaliza todas esas tareas, la última acción de la función es verificar si el nodo en uso tiene elementos secuandarios; en caso positivo, se invoca recursivamente para procesar el siguiente nivel del árbol de nodos. El proceso continúa hasta que se agotan los nodos que habrán de procesarse.

La figura 8-7 presenta los datos de salida del programa cuando se invoca `xmlProcess()` con el elemento raíz del documento como argumento de inicio.

www.computrucionmania.com

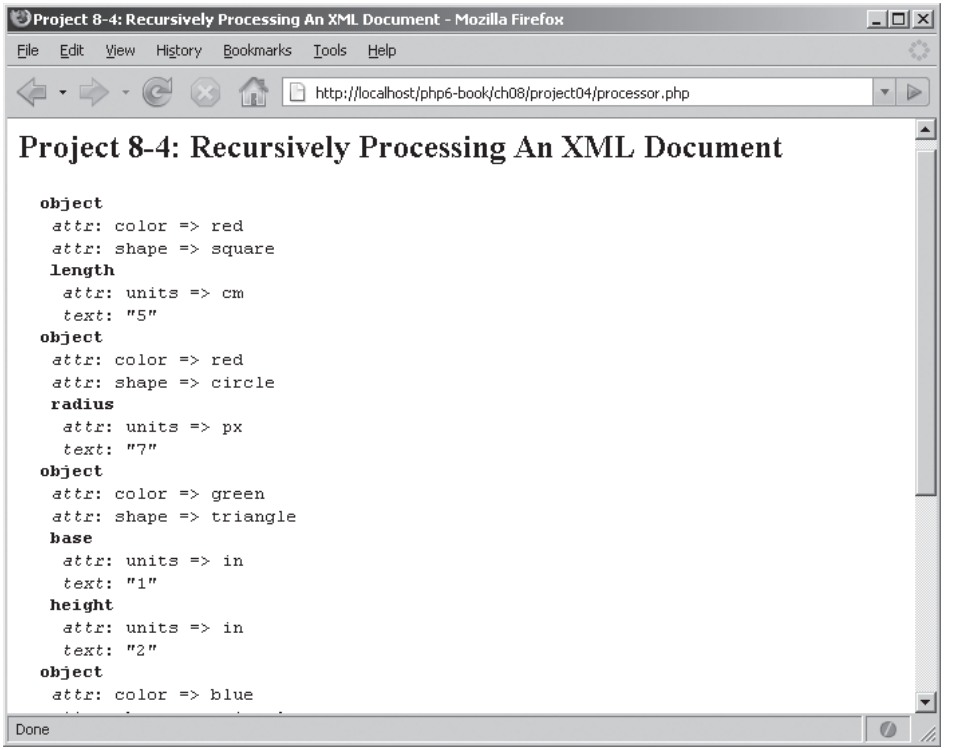


Figura 8-7 Procesamiento recursivo de un documento XML con DOM

Alterar elementos y valores de atributos

Con DOM, cambiar el valor de un elemento XML es muy sencillo: navega hasta el objeto DOMNode que representa el elemento y altera su propiedad nodeValue para insertar el nuevo valor. Como ejemplo, considera el siguiente script PHP, que cambia el título y el autor del segundo libro en *biblioteca.xml* y luego presenta el documento modificado:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');
```

www.comunidadmania.com

```
// obtiene la colección de elementos <libro>
$libros = $doc->getElementsByTagName('libro');

// cambia el elemento <titulo> del segundo <libro>
$libros->item(1)->getElementsByTagName('titulo')->item(0)->nodeValue =
'Invisible Prey';

// cambia el elemento <autor> del segundo <libro>
$libros->item(1)->getElementsByTagName('autor')->item(0)->nodeValue =
'John Sandford';

// presenta la nueva cadena XML
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

Aquí, el método `getElementsByTagName()` es utilizado primero para obtener la colección de elementos `<libro>` y para navegar hacia el segundo elemento de esa colección (cuyo lugar en el índice es 1). Después se vuelve a utilizar para obtener referencias para el objeto DOMNode que representa los elementos `<titulo>` y `<autor>`. A las propiedades `nodeValue` de esos objetos se les asigna entonces un nuevo valor utilizando el operador de asignación de PHP, y el árbol XML modificado es transformado de nueva cuenta a una cadena con el método `saveXML()` del objeto DOMDocument.

Cambiar valores de atributo es igual de fácil: asigna un nuevo valor a un atributo utilizando el método `setAttribute()` del objeto DOMDocument. He aquí un ejemplo, que cambia el '`genero`' del quinto libro y presenta el resultado:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
$libros = $doc->getElementsByTagName('libro');

// cambia el elemento 'genero' del quinto <libro>
$libros->item(4)->setAttribute('genero', 'horror-suspenso');

// presenta la nueva cadena XML
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

www.compumusicamania.com

Crear nuevos documentos XML

DOM contiene un API completamente marcado para crear nuevos documentos XML o para señalar elementos, atributos y otras estructuras similares ya existentes en un árbol XML. Esta API, que es mucho más compleja que la ofrecida por SimpleXML, debe ser tu primera opción cuando crees o modifiques dinámicamente un árbol XML con PHP.

La mejor manera de ejemplificar esta API es con un ejemplo. Examina el siguiente listado, que crea un archivo XML desde cero:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument('1.0');

// crea y adjunta el elemento raíz <programa>
$root = $doc->createElement('programa');
$programa = $doc->appendChild($root);

// crea y adjunta el elemento <curso> bajo <programa>
$curso = $doc->createElement('curso');
$programa->appendChild($curso);

// crea y adjunta el elemento <materia> bajo <curso>
// y añade un valor para el elemento <materia>
$materia = $doc->createElement('materia');
$materiaData = $doc->createTextNode('Macroeconomía');
$curso->appendChild($materia);
$materia->appendChild($materiaData);

// crea y adjunta el elemento <maestro> bajo <curso>
// y añade un valor para el elemento <maestro>
$maestro = $doc->createElement('maestro');
$maestroData = $doc->createTextNode('Profesor Q. Draw');
$curso->appendChild($maestro);
$maestro->appendChild($maestroData);

// crea y adjunta el elemento <creditos> bajo <curso>
// y añade un valor para el elemento <creditos>
$creditos = $doc->createElement('creditos');
$creditosData = $doc->createTextNode('4');
$curso->appendChild($creditos);
$creditos->appendChild($creditosData);

// adjunta un atributo 'transferible' al elemento <creditos>
// establece un valor para el atributo
$transferible = $doc->createAttribute('transferible');
```

www.compumusicamania.com

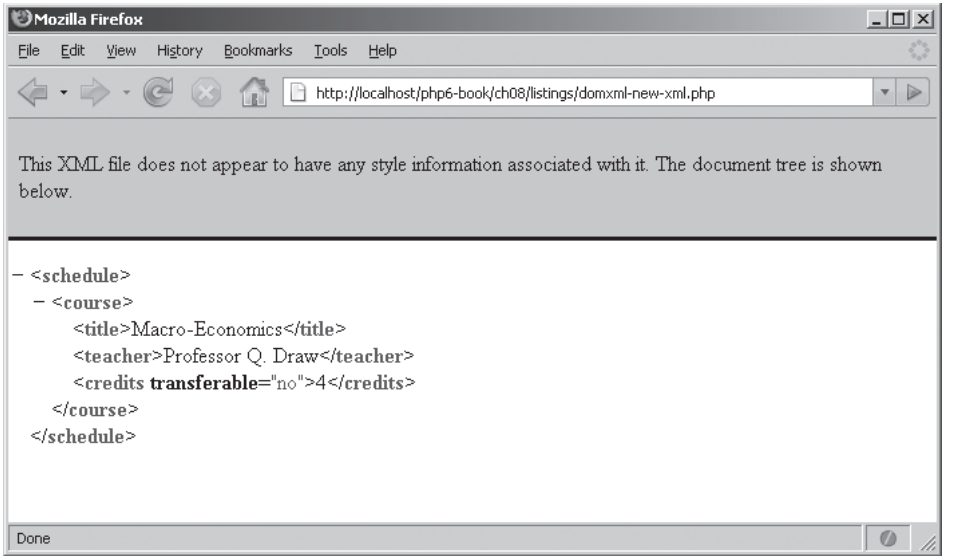


Figura 8-8 Generación dinámica de un nuevo documento XML con DOM

```
$creditos->appendChild($transferible);
$creditos->setAttribute('transferible', 'no');

// forma los datos XML de salida
$doc->formatOutput = true;

// presenta el documento XML
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

La figura 8-8 muestra el documento XML generado por el script.

Este script presenta nuevos métodos, todos ellos relacionados con la creación dinámica de nodos y la manera de adjuntarlos al árbol XML. El proceso presentado requiere dos pasos básicos:

1. Crear un objeto que representa la estructura XML que quieres añadir. La base del objeto DOMDocument presenta los métodos `create...()` correspondientes a cada una de las estructuras primarias XML: `createElement()` para elementos, `createAttribute()` para atributos y `createTextNode()` para datos de texto.

www.comunidadmania.com

2. Adjuntar un nuevo objeto en el punto apropiado dentro del árbol, invocando el método del padre `appendChild()`.

El ejemplo anterior presenta estos pasos, siguiendo secuencias específicas para conseguir el árbol resultante que se muestra en la figura 8-8.

1. Comienza inicializando un objeto DOMDocument llamado `$doc`, para luego invocar su método `createElement()` y generar un nuevo elemento DOMElement llamado `$programa`. Este objeto representa el elemento raíz del documento; como tal, se adjunta a la base del árbol DOM invocando el método `$doc->appendChild()`.
2. Un nivel debajo del elemento `<programa>` viene el elemento `<curso>`. En términos DOM, esto se realiza creando un nuevo objeto DOMElement llamado `$curso` con el método `createElement()` del objeto DOMDocument, y luego adjuntando este objeto al árbol debajo del elemento `<programa>` al invocar `$programa->appendChild()`.
3. Un nivel abajo del elemento `<curso>` viene el elemento `<materia>`. De nuevo, esto se lleva a cabo creando un objeto DOMElement llamado `$materia` y adjuntándolo bajo `<curso>` con la invocación `$curso->appendChild()`. Sin embargo, aquí hay una variación: el elemento `<materia>` contiene un valor de texto llamado '`Macroeconomía`'. Para crear este valor de texto, el script crea un nuevo objeto DomTextNode a través del objeto `createTextNode()`, lo rellena con la cadena de texto y luego lo adjunta como hijo del elemento `<materia>` con la invocación `$materia->appendChild()`.
4. Lo mismo sucede un poco más adelante, cuando se crea el elemento `<creditos>`. Una vez que el elemento y su valor de texto han sido definidos y adjuntados al árbol debajo del elemento `<curso>`, se utiliza el método `createAttribute()` para crear un nuevo objeto DOMAttr para representar el atributo '`transferible`'. Luego, este atributo es adjuntado al elemento `<creditos>` con la invocación `$creditos->appendChild()`, y se le asigna un valor al atributo de manera normal, invocando `$creditos->setAttribute()`.

Conversión entre DOM y SimpleXML

Una característica interesante de PHP es su capacidad para convertir datos XML entre DOM y SimpleXML. Esto se realiza con dos funciones: `simplexml_import_dom()`, que acepta un objeto DOMElement y regresa un objeto SimpleXML, y la función `dom_import_simplexml()`, que hace lo inverso. El siguiente ejemplo muestra esta interpolaridad:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();
```

www.compumusicmania.com

```
// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
$libros = $doc->getElementsByTagName('libro');

// convierte el sexto <libro> en objeto SimpleXML
// presenta el titulo del sexto libro
// datos de salida: 'Glory Road'
$sxml = simplexml_import_dom($libros->item(5));
echo $sxml->titulo;
?>
```

Prueba esto 8-5 Leer y escribir archivos de configuración XML

Ahora que ya sabes leer y crear documentos XML de manera programática, utilicemos este conocimiento en una aplicación que se está haciendo muy popular en estos días: archivos de configuración basados en XML, que utiliza este lenguaje para marcar los datos de configuración de una aplicación.

El siguiente ejemplo lo pone en acción: genera un formulario Web que permite a los usuarios configurar un horno en línea, ingresando datos de configuración sobre temperatura, modo y fuente de calor. Cuando el formulario se envía, los datos proporcionados por el usuario son convertidos en XML y guardados en un archivo de disco. Cuando los usuarios vuelven a ver el formulario, los datos guardados con anterioridad se utilizan para llenar los campos del formulario.

He aquí el código (*configura.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 8-5: Leer y escribir archivos de configuración XML</
title>
  </head>
  <body>
    <h2>Proyecto 8-5: Leer y escribir archivos de configuración XML</h2>
    <h3 style="background-color: silver">Configuración de un horno</h3>
<?php
```

www.compumusicmania.com

```
    // define el nombre y la ruta de acceso del archivo de configuración
    $configFile = 'config.xml';

    // si el formulario no ha sido enviado
    // despliega el formulario
    if (!isset($_POST['submit'])) {

      // establece matriz con parámetros por defecto
      $datos = array();
      $datos['modo'] = null;
      $datos['temperatura'] = null;
      $datos['duración'] = null;
      $datos['direccion'] = null;
      $datos['autoapagado'] = null;

      // lee los valores de configuración en uso
      // utiliza los valores para rellenar el formulario
      if (file_exists($configFile)) {
        $doc = new DOMDocument();
        $doc->preserveWhiteSpaces = false;
        $doc->load($configFile);
        $horno = $doc->getElementsByTagName('horno');
        foreach ($horno->item(0)->childNodes as $nodo) {
          $datos[$nodo->nodeName] = $nodo->nodeValue;
        }
      }
?>
    <form method="post" action="configura.php">
      Modo: <br />
      <select name="data[modo]">
        <option value="asado" <?php echo ($datos['modo'] == 'asado') ?
'selected' : null; ?>>Asado</option>
        <option value="cocido" <?php echo ($datos['modo'] == 'cocido') ?
'selected' : null; ?>>Cocido</option>
        <option value="tostado" <?php echo ($datos['modo'] == 'tostado')
? 'selected' : null; ?>>Tostado</option>
      </select>

      <p>
      Temperatura: <br />
      <input type="text" size="2" name="data[temperatura]" value="<?php
echo $datos['temperatura']; ?>"/>

      <p>
      Duración (minutos): <br />
      <input type="text" size="2" name="data[duración]" value="<?php echo
$datos['duración']; ?>"/>
```

(*continúa*)

```
      <p>

      Fuente del calor y direccion: <br />
      <input type="radio" name="data[direccion]" value="arriba-abajo"
<?php echo ($datos['direccion'] == 'arriba-abajo') ? 'checked' : null;
?>>De arriba hacia abajo</input>
      <input type="radio" name="data[direccion]" value="abajo-arriba"
<?php echo ($datos['direccion'] == 'abajo-arriba') ? 'checked' : null;
?>>De abajo hacia arriba</input>
      <input type="radio" name="data[direccion]" value="ambos" <?php echo
($datos['direccion'] == 'ambos') ? 'checked' : null; ?>>Ambos</input>

      <p>

      Apagar automáticamente cuando termine:
      <input type="checkbox" name="data[autoapagado]" value="yes" <?php
echo ($datos['autoapagado'] == 'yes') ? 'checked' : null; ?>/>

      <p>

      <input type="submit" name="submit" value="Enviar" />
      </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada
    } else {
      // lee los datos enviados
      $config = $_POST['data'];

      // valida los datos enviados como sea necesario

      if ((trim($config['temperatura']) == '') || (trim($config
['temperatura']) != '' && (int)$config['temperatura'] <= 0)) {
        die ('ERROR: Por favor ingrese una temperatura de horno válida');
      }

      if ((trim($config['duración']) == '') || (trim($config['duración'])
!= '' && (int)$config['duración'] <= 0)) {
        die ('ERROR: Por favor ingrese una duración válida');
      }

      // genera un nuevo documento XML
      $doc = new DOMDocument();

      // crea y adjunta el elemento raíz <configuración>
      $root = $doc->createElement('configuración');
      $configuración = $doc->appendChild($root);
      // crea y adjunta elemento <horno> bajo <configuración>
      $horno = $doc->createElement('horno');
      $configuración->appendChild($horno);
```

www.compumusicamania.com

```
      // escribe cada valor de configuración en el archivo
      foreach ($config as $key => $value) {
        if (trim ($value) != '') {
          $elem = $doc->createElement($key);
          $texto = $doc->createTextNode($value);
          $horno->appendChild($elem);
          $elem->appendChild($texto);
        }
      }

      // forma datos de salida XML
      // guarda el archivo XML
      $doc->formatOutput = true;
      $doc->save($configFile) or die ('ERROR: No fue posible escribir el
archivo de configuración');
      echo 'Los datos de configuración se escribieron correctamente en el
archivo.';
    }
?>
  </body>
</html>
```

La figura 8-9 muestra el formulario Web generado por el script.

Una vez que el formulario se ha enviado, los datos ingresados llegan en forma de una matriz asociativa, cuyas claves corresponden a los nombres de los elementos XML. Primero, estos datos son validados, y después la API DOM es utilizada para generar un nuevo árbol XML que contiene esos elementos y sus valores. Una vez que el árbol se genera completamente, la función `save()` del objeto DOMDocument se utiliza para escribir el archivo XML en disco.

He aquí un ejemplo de la apariencia que tendría el documento *config.xml* después de haber enviado el formulario de la figura 8-9:

```
<?xml version="1.0"?>
<configuración>
  <horno>
    <modo>tostado</modo>
    <temperatura>22</temperatura>
    <duración>1</duración>
    <direccion>De abajo hacia arriba</direccion>
    <autoapagado>yes</autoapagado>
  </horno>
</configuración>
```

Si un usuario vuelve a visitar el formulario Web, el script primero verifica la existencia de un archivo de configuración llamado *config.xml*; en caso positivo, se leen los datos XML del

(*continúa*)

www.compumusicmania.com

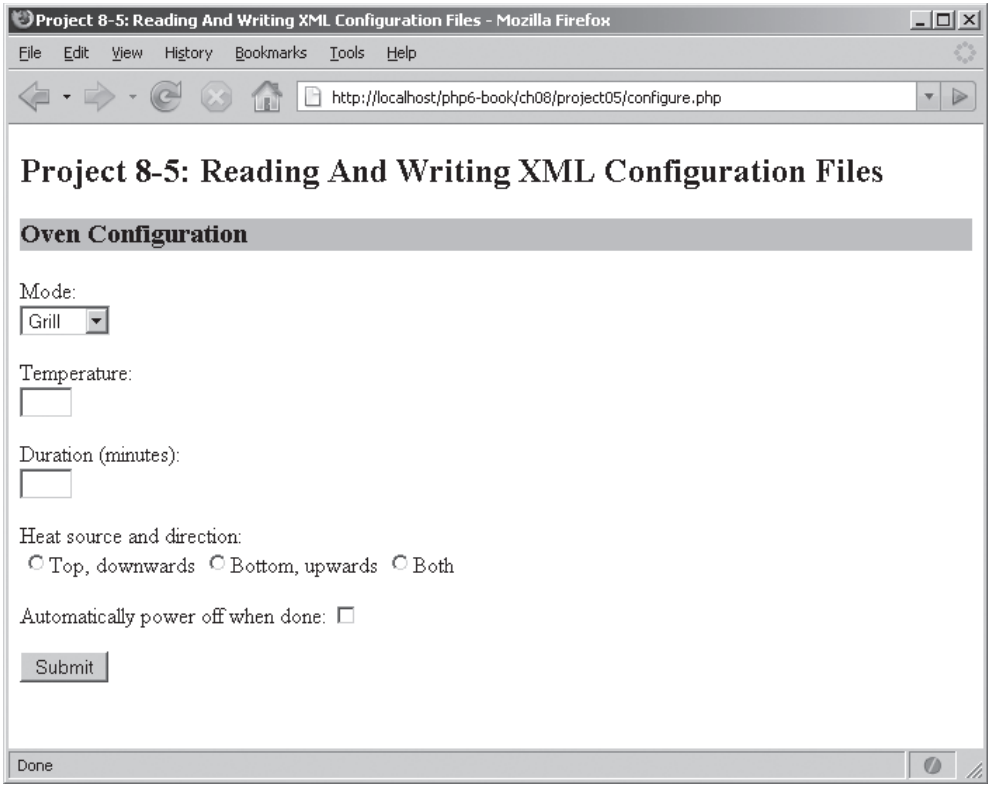


Figura 8-9 Formulario Web para datos de configuración

archivo en un nuevo objeto DOMDocument con el método `load()` y se convierten en una matriz asociativa para iterar sobre la lista de nodos secundarios con un bucle. Los diferentes botones de opción, casillas de verificación y listas de selección en el formulario ya están activados o preseleccionados, dependiendo de los valores de la matriz.

La figura 8-10 muestra el formulario precompletado con los datos leídos del archivo de configuración XML.

Si el usuario envía el formulario Web con nuevos datos, éstos serán codificados de nuevo en formato XML y utilizados para reescribir el archivo de configuración. Ya que la configuración está expresada en XML, todo explorador Web que tenga capacidad para interpretar este lenguaje puede leer y utilizar los datos. Cuando se utiliza de esta manera, XML proporciona un medio para transferir información entre aplicaciones, aunque éstas sean escritas en diferentes lenguajes de programación o se ejecuten en sistemas operativos incompatibles.

www.compumusicmania.com

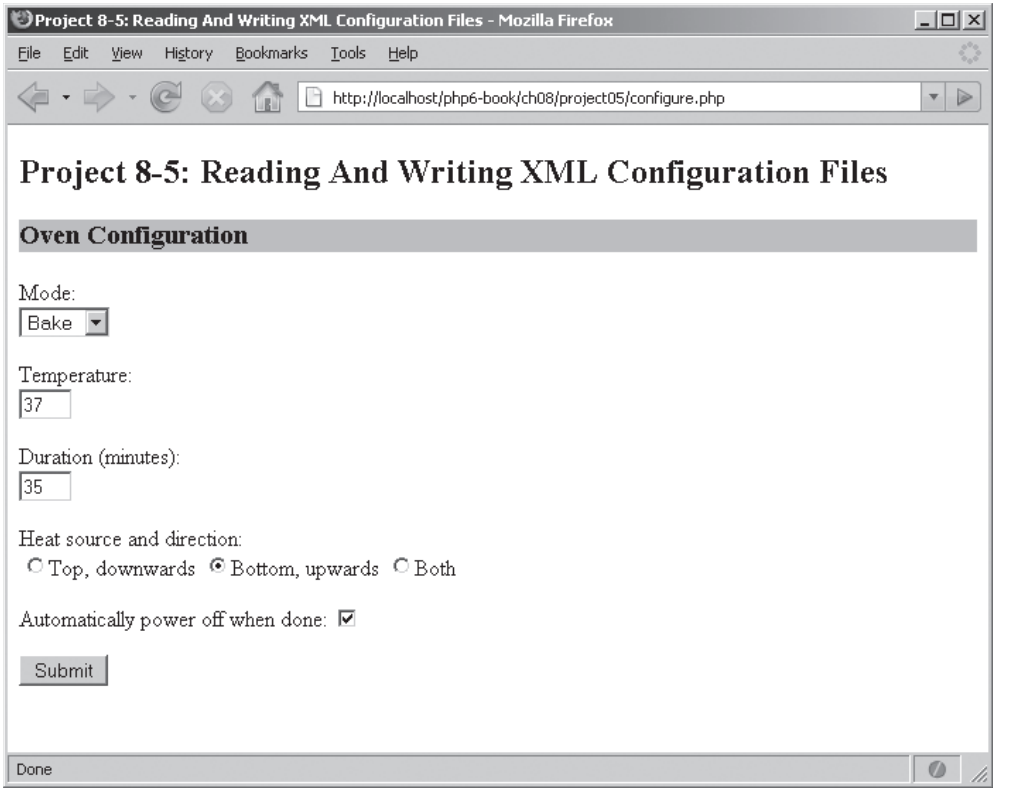


Figura 8-10 El mismo formulario Web, precompletado con los datos de configuración

Resumen

Al finalizar este capítulo ya debes saber lo suficiente para comenzar a escribir programas PHP que puedan interactuar correctamente con datos XML. Este capítulo comenzó con una introducción a XML, explicando sus estructuras básicas como elementos, atributos y datos de carácter, además de proporcionarte un curso relámpago sobre las tecnologías XML y los métodos de análisis sintáctico. Después, siguió una explicación sobre las dos extensiones PHP más populares para procesar datos XML: SimpleXML y DOM, y se explicó la manera en que cada una de ellas puede utilizarse para accesar valores de elemento y atributo, crear colecciones de nodos, además de generar o modificar de manera programática árboles de documento XML. Se utilizaron varios proyectos, desde convertir de XML a SQL hasta la lectura de documentos RSS, para ilustrar de modo práctico la interacción entre XML y PHP.

www.computrucionmania.com

XML es un tema muy extenso y el material en este libro apenas delinea un esbozo de la superficie. Sin embargo, hay muchos tutoriales y artículos excelentes sobre XML y PHP en Web; aquí se presentan algunos vínculos hacia ellos; consúltalos si quieres saber más sobre este tema tan interesante y en continuo cambio:

- Bases de XML, en **www.melonfire.com/community/columns/trog/article.php?id=78**
- Bases de XPath, en **www.melonfire.com/community/columns/trog/article.php?id=83**
- Bases de XSL, en **www.melonfire.com/community/columns/trog/article.php?id=82**
- Funciones SimpleXML, en **www.php.net/simplexml**
- Funciones API DOM en PHP, en **www.php.net/dom**
- La especificación DOM, en **www.w3.org/DOM/**
- Construcción de documentos XML utilizando PHP y PEAR, en **www.melonfire.com/community/columns/trog/article.php?id=180**
- Serialización de XML, en **www.melonfire.com/community/columns/trog/article.php?id=244**
- Realizar Procedimiento de Invocación Remota (RCP) con PHP, en **www.melonfire.com/community/columns/trog/article.php?id=274**

✓ Autoexamen Capítulo 8

1. ¿Cuáles son los dos métodos para analizar sintácticamente un documento XML y en qué se diferencian?
2. Nombra dos características de un documento XML bien formado.
3. Dado el siguiente documento XML (*email.xml*), escribe un programa para recuperar y presentar todas las direcciones de correo electrónico del documento utilizando SimpleXML:

```
<?xml version='1.0'?>
<data>
  <persona>
    <nombre>Clon Uno</nombre>
    <email>uno@domain.com</email>
  </persona>
  <persona>
    <nombre>Clon Sesenta y cuatro</nombre>
    <email>sesentaycuatro@domain.com</email>
  </persona>
```

www.compumusicmania.com

```
  <persona>
    <nombre>Clon Tres</nombre>
    <email>tres@domain.com</email>
  </persona>
  <persona>
    <nombre>Clon Noventa y Nueve </nombre>
    <email>noventaynueve@domain.com</email>
  </persona>
</data>
```

4. Dado el siguiente documento XML (*arbol.xml*), sugiere tres maneras diferentes para recuperar el texto con valor `'John'` utilizando DOM:

```
<?xml version='1.0'?>
<arbol>
  <persona type="abuelo" />
  <persona type="abuela" />
  <hijos>
    <persona type="papá" />
    <persona type="mamá" />
    <hijos>
      <persona type="hermano">
        <nombre>John</nombre>
      </persona>
    <persona type="hermana">
      <nombre>Jane</nombre>
    </persona>
    </hijos>
  </hijos>
</arbol>
```

5. Escribe un programa que cuente el número de elementos en un archivo XML. Utiliza DOM.

6. Escribe un programa que procese el archivo *biblioteca.xml* que aparece al principio de este capítulo, para incrementar en 1 el rating de cada libro; presenta el resultado de la modificación. Utiliza SimpleXML.

7. Escribe un programa que se conecte a una base de datos MySQL y recupere el contenido de cualquier tabla como un archivo XML. Utiliza DOM.

www.computrucionmania.com

Capítulo 9

Trabajar con cookies, sesiones y encabezados

www.compumusicmania.com

Habilidades y conceptos clave

- Comprender cómo funcionan las cookies
- Escribir y utilizar tus propias cookies para crear páginas "desprendibles"
- Compartir datos entre páginas con sesiones y variables de sesión
- Manipular el explorador del usuario enviándole encabezados HTTP personalizados

---

Tal vez ya sepas que el protocolo de transferencia de hipertexto (http, Hypertext Transfer Protocol) es el protocolo estándar para transferir datos entre tu explorador y los diferentes sitios Web que visitas. Sin embargo, lo que tal vez no sepas es que HTTP es un protocolo "sin estado", que trata cada solicitud de página Web como una transacción única e independiente, sin ninguna relación con la transacción antecedente. Para solucionar este inconveniente, casi todas las estaciones Web utilizan cookies o sesiones para "preservar el estado", y poder ofrecer mejores servicios a los usuarios, como las transacciones comerciales en línea y la restauración automática de configuración personal de la página.

PHP incluye soporte completo a cookies y sesiones. Al utilizar este soporte es fácil crear ambas, almacenar y recuperar los datos específicos del usuario que se encuentran en ellas, manipularlas desde tu aplicación PHP e incluso enviar encabezados personalizados al explorador del usuario para alterar su comportamiento estándar. Este capítulo te enseñará cómo hacerlo, con algunos ejemplos prácticos que muestran lo útil que pueden ser estas características cuando construyes sitios y aplicaciones Web.

Trabajar con cookies

Las cookies no son difíciles de comprender, pero hay algunos aspectos básicos con los que necesitas familiarizarte antes de que comiences a escribir el código que las maneje. La siguiente sección te presentará una introducción a estos aspectos básicos.

Aspectos básicos de las cookies

En su forma más sencilla, una *cookie* es un archivo de texto guardado en el equipo del usuario por un sitio Web. El archivo contiene información que el sitio puede recuperar durante la siguiente visita del usuario, con lo que le permite "reconocerlo" para proporcionarle un conjunto enriquecido de características personalizadas para ese usuario específico. Ejemplos comunes de estas características son: mostrar el contenido del sitio de modo personalizado, de acuerdo con

www.compumusicmania.com

las preferencias del usuario; mantener un registro del contenido visitado, además de integrar información personal del usuario en la disposición de los elementos mostrados en el sitio.

Como las cookies facilitan la transferencia de datos entre el usuario y el sitio Web remoto, se les ha vilipendiado en los medios de información masiva tildándolas de "inseguras" y "malas". La verdad es que se trata de una exageración: las cookies (como otras tecnologías) pueden ser mal utilizadas; la mayor parte de los sitios Web las ocupan sin causar daño y pueden vincularse directamente para mejorar la experiencia del usuario al navegar por Web. Además, las cookies tienen importantes características de seguridad, como las que se muestran a continuación:

- Una cookie sólo puede ser leída por el sitio Web o el dominio que la creó.
- Un solo dominio no puede colocar más de 20 cookies.
- Una sola cookie no puede exceder de 4 kilobytes de tamaño.
- El número máximo de cookies que se pueden establecer en el equipo de un usuario es de 300.

PRECAUCIÓN

Como las cookies se almacenan en el disco duro del usuario, los desarrolladores tienen poco control sobre ellas. Si el usuario decide "apagar" el soporte a cookies en su explorador, tus cookies simplemente no almacenarán. Por ello, si la persistencia de los datos es una característica importante en tu sitio Web, debes tener listo un plan de respaldo (como cookies del lado del servidor o trabajar con sesiones).

Atributos de las cookies

Una cookie típica de un sitio Web contiene menos ingredientes que una galleta cocinada; para ser preciso, son cinco. La tabla 9-1 presenta la lista.

1. Cada cookie contiene una pareja nombre-valor, que representa el nombre de la variable y su correspondiente valor que debe ser almacenado en la cookie.

Atributo	Descripción	Ejemplo
`name=`*`valor`*	El nombre y valor de la cookie	`'email=yo@algún.dominio.com'`
`expires`	La validación de la cookie	`'expires=Viernes, 25-Ene-08 23:59:50 IST'`
`domain`	El dominio asociado con la cookie	`'domain=estesitioweb.com'`
`path`	La ruta de acceso del dominio asociado con la cookie	`'path=/'`
`secure`	Si está presente, se requiere una conexión segura HTTP para leer la cookie	`'secure'`

Tabla 9-1 Atributos de una cookie

www.compumusicmania.com

2. El atributo `'expires'` de una cookie define su tiempo válido de duración. Si se establece este atributo con una fecha atrasada, el explorador borrará la cookie.
3. El atributo `'domain'` define el nombre de dominio asociado con la cookie. Sólo este dominio podrá tener acceso a la información almacenada por la cookie.
4. El atributo `'path'` define cuáles secciones del dominio especificado en el atributo `'domain'` pueden tener acceso a la cookie. Al establecerlo en la raíz del servidor (`/`) se permite que todo el dominio tenga acceso a la información almacenada en la cookie.
5. El atributo `'secure'` indica si una conexión HTTP segura es indispensable, antes de que se tenga acceso a la cookie.

Encabezados de cookies

Las cookies se transmiten entre el explorador del usuario y el sitio Web remoto a través de encabezados HTTP. Por ejemplo, al establecer una cookie, el sitio Web debe enviar al explorador cliente un encabezado `'Set-Cookie:'` que contenga los atributos necesarios. El siguiente código ejemplifica los encabezados enviados para crear dos cookies para un dominio:

```
Set-Cookie: username=john; path=/; domain=.estesitio.com;
expires=Friday, 25-Jan-08 23:59:50 IST
Set-Cookie: location=UK; path=/; domain=.estesitio.com;
expires=Friday, 25-Jan-08 23:59:50 IST
```

De manera similar, si una cookie en particular es válida para un sitio Web y su ruta de acceso, el explorador del usuario automáticamente incluirá la información de esta cookie en un encabezado `'Cookie:'` cuando solicite el URL del sitio en cuestión. En el ejemplo anterior, la siguiente ocasión que el usuario visite el sitio "estesitio.com" su explorador incluirá automáticamente el siguiente encabezado en la solicitud:

```
Cookie: username=john; location=UK
```

Pregunta al experto

P: **¿Puedo leer las cookies almacenadas en mi computadora?**

R: Las cookies son archivos de texto almacenados en tu computadora y, como tales, puedes leerlos con cualquier procesador de texto. La ubicación exacta del lugar donde se almacenan depende del explorador y el sistema operativo que estés utilizando. Por ejemplo, en Microsoft Windows, Internet Explorer almacena las cookies como archivos independientes en la ruta de acceso *C:/Documents and Settings/[nombreusuario]/cookies*, mientras Mozilla Firefox almacena todas sus cookies en un solo archivo en *C:/Documents and Settings/[nombreusuario]/Application Data/Mozilla/Firefox/Profiles/[nombreperfil]/cookies.txt*.

www.compumusicmania.com

Establecer cookies

La API de PHP para manipular cookies es muy sencilla; consiste en una sola función: `setcookie()`, que puede utilizarse para establecer y eliminar cookies. Esta función acepta seis argumentos: el nombre y valor de la cookie, la fecha de expiración en formato sello cronológico de UNIX, el dominio y ruta de acceso y un valor booleano para establecer el atributo `'secure'` como verdadero o falso. Esta función regresa un valor verdadero si el encabezado de la cookie fue transmitido con éxito a la computadora del usuario; sin embargo, esto no es indicación de que la cookie haya sido establecida con éxito (en caso de que la computadora del usuario haya sido configurada para rechazar todas las cookies, el encabezado pudo ser transmitido con éxito pero es posible que la cookie en sí no haya sido establecida en la computadora del usuario).

He aquí un ejemplo, que establece una cookie que contiene el correo electrónico del usuario:

```
<?php
// establece cookie
setcookie('email', 'john@sitioweb.com', mktime()+129600, '/');
?>
```

Es posible establecer múltiples cookies, invocando una función `setcookie()` para cada una de ellas. He aquí un ejemplo que establece tres cookies con diferentes periodos de validez y rutas de acceso:

```
<?php
// establece múltiples cookies
setcookie('username', 'ballenablanca', mktime()+129600, '/');
setcookie('email', 'john@sitioweb.com', mktime()+86400, '/');
setcookie('puesto', 'moderador', mktime()+3600, '/admin');
?>
```

PRECAUCIÓN

Dado que las cookies se establecen con el uso de encabezados HTTP, la invocación `setcookie()` debe anteceder a cualquier dato de salida generado por tu script. Cualquier violación a esta regla no sólo impedirá que la cookie se establezca en el equipo del usuario; además generará una serie de mensajes de error PHP.

Leer cookies

Las cookies establecidas por un dominio pueden ser leídas con la matriz asociativa especial `$_COOKIE` utilizando los scripts PHP que se ejecutan en dicho dominio. Es posible acceder a estas cookies con el uso de la notación de matrices estándar. He aquí un ejemplo:

```
<?php
// leer cookie
if(isset($_COOKIE['email'])) {
  echo 'Bienvenido de nuevo, ' . $_COOKIE['email'] . '!';
```

www.compumusicamania.com

```
} else {
  echo '¡Hola, nuevo usuario!';
}
?>
```

Eliminar cookies

Para eliminar una cookie utiliza `setcookie()` con su nombre para establecer su fecha de validez con un valor pasado, como se muestra aquí:

```
<?php
// eliminar cookie
$ret = setcookie('puesto', 'moderador', mktime()-1600, '/admin');
if ($ret) {
  echo 'Encabezados de cookie transmitidos con éxito.';
}
?>
```

Prueba esto 9-1 Guardar y restablecer preferencias del usuario

Construyamos ahora una aplicación sencilla que utilice cookies para guardar y restaurar las preferencias de un usuario. El formulario Web del siguiente ejemplo solicita al usuario que seleccione sus preferencias para un viaje largo en avión y guarda estas preferencias en una cookie en el equipo del usuario. Cuando regresa a esta página, las preferencias establecidas con anterioridad son leídas de la cookie y restauradas automáticamente.

He aquí el código (*preferencias-vuelo.php*):

```
<?php
// si la forma ya ha sido enviada
// escribe la cookie con la configuración
if(isset($_POST['submit'])) {
  $ret1 = (isset($_POST['nombre'])) ? setcookie('nombre',
$_POST['nombre'], mktime() + 36400, '/') : null;
  $ret2 = (isset($_POST['asiento'])) ? setcookie('asiento',
$_POST['asiento'], mktime() + 36400, '/') : null;
  $ret3 = (isset($_POST['comida'])) ? setcookie('comida',
$_POST['comida'], mktime() + 36400, '/') : null;
  $ret4 = (isset($_POST['ofertas'])) ? setcookie('ofertas',
implode(',',$_POST['ofertas']), mktime() + 36400, '/') : null;
}

// lee la cookie y asigna sus valores
//  a variables PHP
$nombre = isset($_COOKIE['nombre']) ? $_COOKIE['nombre'] : '';
```

www.compumusicamania.com

```
$asiento = isset($_COOKIE['asiento']) ? $_COOKIE['asiento'] : '';
$comida = isset($_COOKIE['comida']) ? $_COOKIE['comida'] : '';
$ofertas = isset($_COOKIE['ofertas']) ? explode (',',$COOKIE['ofertas'])
: array();
?>
<!DOCTYPE html PUBLIC «-//W3C//DTD XHTML 1.0 Transitional//EN»
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 9-1: Guardar y restaurar preferencias del usuario</
title>
  </head>
  <body>
    <h2>Proyecto 9-1: Guardar y restaurar preferencias del usuario</h2>
    <h3>Seleccione sus Preferencias de Vuelo</h3>
    <?php
    // si el formulario ya ha sido enviado
    // despliega mensaje de éxito
    if (isset($_POST['submit'])) {
    ?>
    Gracias por su selección.
    <?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    } else {
    ?>
    <form method="post" action="preferencias-vuelo.php">
      Nombre: <br />
      <input type="text" size="20" name="nombre" value"<?php echo $name;
?>" />

      <p>

      Selección de asiento: <br />
      <input type="radio" name="asiento" value="pasillo" <?php echo
($asiento == 'pasillo') ? 'checked' : ''; ?>>Pasillo</input>
      <input type="radio" name="asiento" value="ventana" <?php echo
($asiento == 'ventana') ? 'checked' : ''; ?>>Ventana</input>
      <input type="radio" name="asiento" value="centro" <?php echo
($asiento == 'centro') ? 'checked' : ''; ?>>Centro</input>

      <p>

      Selección de comida: <br />
      <input type="radio" name="comida" value="normal-veg" <?php echo
($comida == 'normal-veg') ? 'checked' : ''; ?>>Vegetariana</input>
      <input type="radio" name="comida" value="normal-nveg" <?php echo
($comida == 'normal-nveg') ? 'checked' : ''; ?>>No Vegetariana</input>
      <input type="radio" name="comida" value="diabética" <?php echo
```

(*continúa*)

www.compumusicmania.com

```
($comida == 'diabética') ? 'checked' : ''; ?>>Diabética</input>
      <input type="radio" name="comida" value="niño" <?php echo
($comida == 'niño') ? 'checked' : ''; ?>>Niño</input>
      <p>

      Estoy interesado en ofertas especiales de los vuelos a: <br />
      <input type="checkbox" name="ofertas[]" value="LHR" <?php echo in_
array('LHR', $ofertas) ? 'checked' : ''; ?>>Londres (Heathrow)</input>
      <input type="checkbox" name="ofertas[]" value="CDG" <?php echo in_
array('CDG', $ofertas) ? 'checked' : ''; ?>>París</input>
      <input type="checkbox" name="ofertas[]" value="CIA" <?php echo in_
array('CIA', $ofertas) ? 'checked' : ''; ?>>Roma (Ciampino)</input>
      <input type="checkbox" name="ofertas[]" value="IBZ" <?php echo in_
array('IBZ', $ofertas) ? 'checked' : ''; ?>>Ibiza</input>
      <input type="checkbox" name="ofertas[]" value="SIN" <?php echo in_
array('SIN', $ofertas) ? 'checked' : ''; ?>>Singapur</input>
      <input type="checkbox" name="ofertas[]" value="HKG" <?php echo in_
array('HKG', $ofertas) ? 'checked' : ''; ?>>Hong Kong</input>
      <input type="checkbox" name="ofertas[]" value="MLA" <?php echo in_
array('MLA', $ofertas) ? 'checked' : ''; ?>>Malta</input>
      <input type="checkbox" name="ofertas[]" value="BOM" <?php echo in_
array('BOM', $ofertas) ? 'checked' : ''; ?>>Bombay</input>

      <p>

      <input type="submit" name="submit" value="Enviar" />
    </form>
    <?php
   }
   ?>
  </body>
</html>
```

Este formulario Web contiene varios campos para que el usuario ingrese su nombre, seleccione un asiento y tipo de comida y acepte recibir ofertas especiales. La figura 9-1 muestra su apariencia.

Cuando este formulario es enviado, las opciones seleccionadas por el usuario son guardadas en cookies en su computadora. Como resultado, cada vez que el usuario vuelve a visitar el formulario, los datos de las cookies son leídos por PHP en `$_COOKIE`. El formulario Web puede usar entonces estos datos de cookies para llenar de manera automática los campos de acuerdo con el último envío del usuario. De esta manera, la página parece "recordar" las preferencias del usuario en cada visita.

Las cookies son almacenadas en el equipo del usuario y pueden ser vistas utilizando cualquier procesador de textos. Algunos exploradores también te permiten ver las cookies con ayuda de herramientas integradas. Por ejemplo, en Mozilla Firefox, puedes ver el contenido de todas las cookies guardadas en tu computadora con el menú Herramientas | Opciones | Privacidad | Mostrar Cookies, como se presenta en la figura 9-2.

www.compumusicmania.com

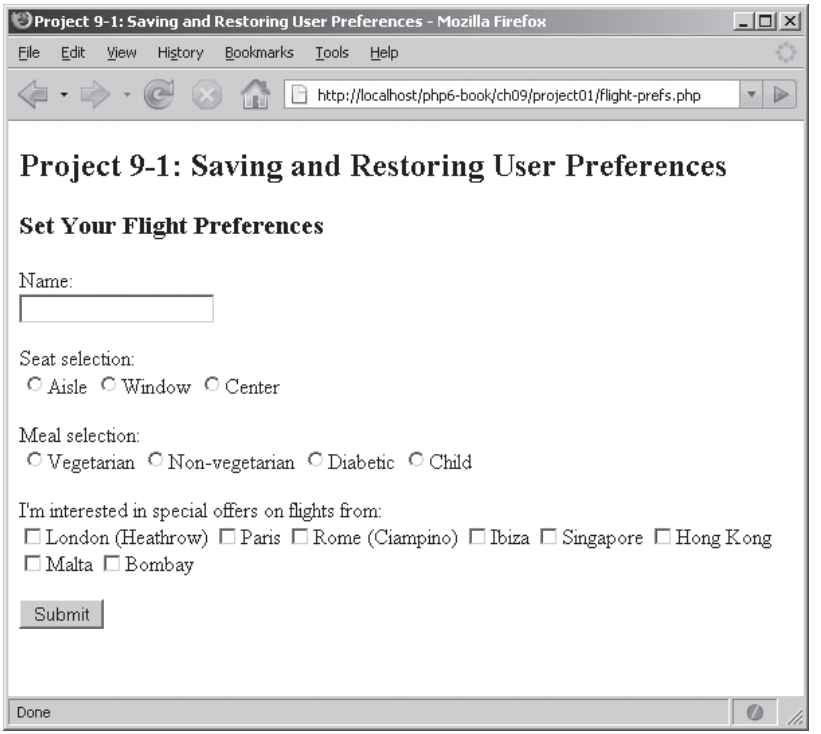


Figura 9-1 Formulario Web para que el usuario ingrese sus preferencias para el vuelo

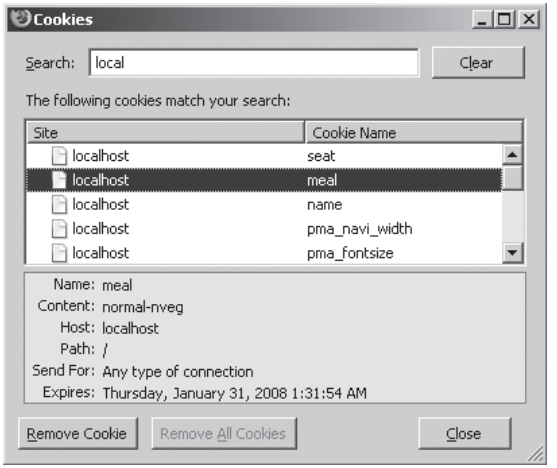


Figura 9-2 Ver cookies en Mozilla Firefox

www.compumusicmania.com

Trabajar con sesiones

Como las cookies, las sesiones también ofrecen un método para mantener el estado de un sitio Web o una aplicación, sólo que utiliza un método algo diferente. Las siguientes secciones explican el funcionamiento de las sesiones con PHP, además de las funciones de PHP para crear y utilizar sesiones dentro de una aplicación Web.

Aspectos básicos de las sesiones

Ahora ya sabes lo que son las cookies: archivos de texto almacenados en el equipo del usuario que ayudan al sitio Web o la aplicación a reconocer al usuario y recuperar información específica sobre el mismo. El problema con las cookies es que no son muy seguras; como se almacenan en el equipo cliente, es posible que el usuario abra el archivo de las cookies, las lea o modifique la información que contienen, en ocasiones con malas intenciones.

Por ello muchos sitios Web prefieren utilizar *sesiones*. Las sesiones funcionan de manera muy similar a las cookies, salvo que la información utilizada para mantener el estado se almacena en el servidor y no en el equipo cliente. En un ambiente basado en sesiones, cada cliente es identificado por un número único (llamado *identificador de sesión*) y este número se utiliza para vincular a cada cliente con su información almacenada en el servidor. Cada vez que el cliente visita el sitio Web o despliega la aplicación, el sitio lee el identificador de sesión del cliente y restaura la información correspondiente desde un repositorio de datos ubicado en el servidor.

Bajo este sistema, la información se almacena en una base de datos SQL o en un archivo de texto en el servidor; como resultado, los usuarios no pueden tener acceso ni modificar la información, por lo que el sistema entero es mucho más seguro. El *identificador de sesión* en sí puede almacenarse en el equipo del usuario como cookie, o puede pasarse de página en página en el URL. Con PHP, esta cookie es llamada `PHPSESSID`.

Las siguientes secciones abordan las funciones PHP para crear sesiones, registrar y utilizar variables de sesión y destruir sesiones.

Crear sesiones y variables de sesión

Es fácil iniciar una nueva sesión con PHP: simplemente invoca la función `session_start()` para crear una nueva sesión y generar un ID de sesión para el cliente. Una vez que se ha creado la sesión, es posible crear y adjuntar cualquier número de *variables de sesión* para ella; éstas son variables regulares, en el sentido de que pueden almacenar información en forma de texto o números, pero también son especiales porque sólo existen durante la sesión, mientras el usuario navega por el sitio.

Las variables de sesión son "registradas" al guardarlas como pareja clave-valor en una matriz asociativa `$_SESSION`. Como `$_POST` y `$_GET`, esta matriz siempre está disponible de manera global y puede accederse directamente desde cualquier punto de tu script PHP.

www.compumusicmania.com

Para ver cómo funcionan las sesiones y las variables de sesión, examina el siguiente script, que crea una nueva sesión de cliente y registra dos variables de sesión:

```
<?php
// inicia sesión
session_start();

// registra variables de sesión
$_SESSION['nombre'] = 'Ronald';
$_SESSION['especie'] = 'Conejo';
?>
```

PRECAUCIÓN

Por lo general, la función `session_start()` establece una cookie que contiene el ID de sesión en el equipo cliente. Por ello, al igual que con la función `setcookie()`, la invocación `session_start()` debe anteceder a cualquier dato de salida generado por el script. Esto se debe a restricciones en el protocolo HTTP que requiere que los encabezados de las cookies se envíen antes que los datos de salida del script.

Ahora es posible acceder a estas variables de sesión desde cualquier página del mismo dominio Web. Para ello, crea un nuevo script PHP, recrea la sesión invocando `session_start()` y luego trata de accesar los valores de la matriz asociativa `$_SESSION`, como en el siguiente ejemplo:

```
<?php
// inicia sesión
session_start();

// lee las variables de sesión
// datos de salida: 'Bienvenido de nuevo, Ronald Conejo'
echo 'Bienvenido de nuevo, ' . $_SESSION['nombre'] . ' ' . $_
SESSION['especie'] ;
?>
```

Pregunta al experto

P: ¡Ayuda! Mis sesiones de variables no se están guardando. ¿Qué hago ahora?

R: Si una variable de sesión no se está registrando correctamente, existen un par de posibilidades que debes revisar antes de darte por vencido y llamar a la policía de sesiones:

- Por defecto, PHP guarda los datos de sesión en el directorio */tmp* del servidor. Sin embargo, si estás utilizando Microsoft Windows, este directorio no existe por defecto y tus sesiones no se almacenarán correctamente. Para rectificarlo, abre el archivo de configuración de PHP, *php.ini*, y edita la variable `'session.save_path'` para escribir el directorio temporal de tu computadora o servidor.

(*continúa*)

www.compumusicmania.com

- Por defecto, PHP establece una cookie en el equipo cliente con el identificador de sesión. Si el explorador del usuario está configurado para rechazar todas las cookies, este identificador de sesión no se almacenará y la información de la sesión no se mantendrá de página en página. Para corregir esto, puedes transmitir el identificador de sesión de una página a otra como parte de la cadena URL (aunque esto es menos seguro) estableciendo como verdadera la variable `'session.use_trans_sid'` en el archivo de configuración de PHP.
- Cada sesión PHP tiene un *valor de término* (la duración, medida en segundos) de la sesión en ausencia de cualquier actividad por parte del usuario. En algunos casos, este valor de término puede establecerse demasiado bajo para la aplicación, y el resultado es que la aplicación se destruye demasiado pronto. Puedes ajustar este valor desde la variable `'session.gc_maxlifetime'` en el archivo de configuración PHP.

Eliminar sesiones y variables de sesión

Para eliminar una variable de sesión específica, simplemente aplica la función `unset()` a la correspondiente clave dentro de la matriz `$_SESSION`:

```
<?php
// inicia sesión
session_start();

// elimina variable de sesión
unset($_SESSION['nombre']);
    ?>
```

Como opción, para eliminar todas las variables de sesión y la sesión misma, utiliza la función `session_destroy()`:

```
<?php
// inicia sesión
session_start();

// eliminar sesión
session_destroy();
?>
```

Es importante advertir que antes de destruir la sesión con `session_destroy()`, primero necesitas recrear el ambiente de sesión (para que haya algo que destruir) con la función `session_start()`.

www.compumusicmania.com

Prueba esto 9-2 Rastrear visitas previas a la página

Pongamos ahora toda esta teoría en contexto, construyendo una aplicación sencilla que muestre el funcionamiento de las sesiones. El siguiente ejemplo utiliza una sesión para registrar cada visita hecha por un usuario particular a una página Web. En cada visita, el script presenta las fechas y horas de todas las visitas anteriores y añade a la sesión un registro para la visita actual.

He aquí el código (*visitas.php*):

```
<?php
// inicia sesión
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 9-2: Rastrear visitas previas a la página</title>
  </head>
  <body>
    <h2>Proyecto 9-2: Rastrear visitas previas a la página</h2>
    <?php
    if (!isset($_SESSION['visitas'])) {
      echo 'Esta es su primera visita.';
    } else {
      echo 'Visitó esta página en: <br />';
      foreach ($_SESSION['visitas'] as $v) {
        echo date('d M Y h:i:s', $v) . '<br />';
      }
    }
    ?>
  </body>
</html>
<?php
// añade el sello temporal de la visita actual
$_SESSION['visitas'] [] = mktime();
?>
```

Para ver este script en acción, visita la página Web unas cuantas veces y verás una lista creciente que contiene los registros de las anteriores visitas. Esta lista se mantiene disponible mientras no cierres el explorador, de manera que aunque visites algunos otros sitios y luego regreses al script, seguirás viendo los registros de tus visitas anteriores. Sin embargo, una vez que cierres el explorador, la cookie de sesión será destruida y la lista comenzará de nuevo.

(*continúa*)

www.compumusicmania.com

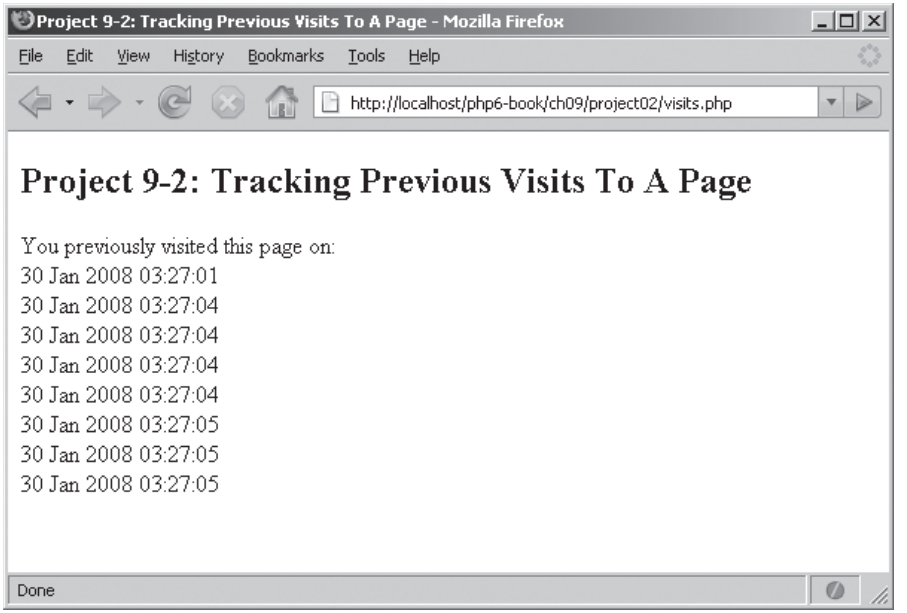


Figura 9-3 Rastreo de visitas previas a una página Web

La figura 9-3 muestra un ejemplo de lo que probablemente verás.

Este script funciona creando una sesión cada vez que el usuario visita la página y almacena el sello cronológico de la visita en la variable de sesión `$_SESSION['visitas']`. En cada visita subsecuente la sesión se recrea, la matriz que contiene el sello temporal de las visitas previas se restaura y se utiliza un bucle `foreach` para iterar sobre la matriz y presentar los registros en formato de fecha y hora legible para los humanos.

Utilizar encabezados HTTP

En secciones anteriores viste que PHP envía automáticamente encabezados al explorador cliente para establecer cookies. Como desarrollador PHP, éstos no son los únicos encabezados que puedes enviar. PHP te permite enviar al explorador cliente cualquier encabezado soportado por el protocolo HTTP, a través de la función `header()`.

www.compumusicmania.com

Quizás el encabezado de mayor uso sea `'Location:'`, utilizado para redireccionar el explorador del usuario a un URL diferente sin que el usuario lo note. He aquí un ejemplo:

```
<?php
// redireccionar a www.php.net
header('Location: http://www.php.net');
?>
```

También puedes enviar otros encabezados, por ejemplo: `'Cache-Control'` o `'Content-Encoding'`, como en el siguiente ejemplo:

```
<?php
// controla el caché
header('Cache-Control: no-cache');

// establece el tipo de contenido
header('Content-type: text/xml');

// establece la codificación del contenido
echo "<?xml version='1.0'?><doc><element/></doc>";
?>
```

Como sabes, enviar un encabezado HTTP después de que el script haya generado datos de salida producirá un error. Puedes evitar este error probando primero si algún encabezado ya ha sido enviado; para ello utiliza la función de PHP `headers_sent()`. Examina el siguiente ejemplo, que lo ilustra:

```
<?php
// comprueba si algún encabezado ha sido enviado
// de no ser así, envía los encabezados
// en caso de envío anterior muestra un mensaje de error
if (!headers_sent()) {
  header('Cache-Control: no-cache');
  header('Content-type: text/plain');
  echo 'Encabezados enviados.';
} else {
  die ('ERROR: ¡No es posible enviar encabezados!');
}
?>
```

TIP

Para una lista completa de los encabezados que soporta el protocolo HTTP, consulta la especificación del mismo en www.w3.org/Protocols/rfc2616/rfc2616.html, o en Wikipedia en.wikipedia.org/wiki/List_of_HTTP_headers.

www.compumusicmania.com

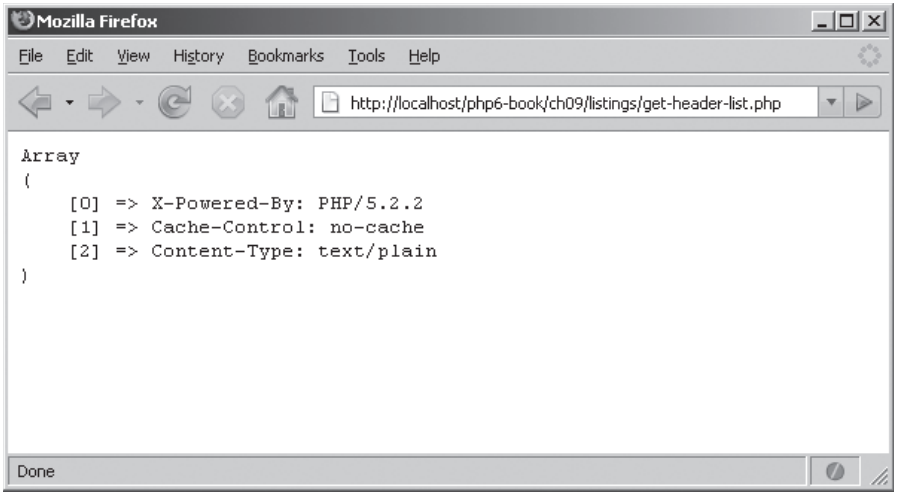


Figura 9-4 Lista de los encabezados HTTP que serán enviados al cliente

Para obtener una lista completa de los encabezados que serán enviados al explorador del usuario, utiliza la función `headers_list()`. He aquí un ejemplo:

```
<?php
// controla el caché
header('Cache-Control: no-cache');

// establece el tipo de contenido
header('Content-type: text/plain');

// presenta la lista de encabezados
print_r(headers_list());
?>
```

La figura 9-4 muestra los datos de salida de este script.

Prueba esto 9-3 Construir un formulario de ingreso mejorado

En el capítulo 7 construiste un formulario de inicio de sesión que actuaba dinámicamente con la base de datos MySQL para verificar los permisos del usuario. Ahora, mejoremos un poco ese formulario con sesiones, cookies y encabezados. El siguiente ejemplo enriquecerá el formulario creado en el capítulo 7 para recordar el nombre de usuario que se haya ingresado y para restringir el acceso a ciertas páginas a las que pueden acceder sólo los usuarios que han iniciado sesión.

www.computeciencia.com

Dando por hecho que configuraste la base de datos MySQL de acuerdo con las instrucciones del capítulo 7, aquí está el formulario de ingreso modificado (*ingreso.php*):

```
<?php
// si el formulario no ha sido enviado
// muestra el formulario
if (!isset($_POST['submit'])) {
  $nombredeusuario = (isset($_COOKIE['nombre'])) ? $_COOKIE['nombre'] :
'';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 9-3: Construir un formulario de ingreso mejorado</title>
  </head>
  <body>
    <h2>Proyecto 9-3: Construir un formulario de ingreso mejorado</h2>
    <form method="post" action="ingreso.php">
      Nombre de Usuario: <br />
      <input type="text" name="nombredeusuario" value="<?php echo
$nombredeusuario; ?>"/>
      <p>
      Contraseña: <br />
      <input type="password" name="contrasena" />
      <p>
      <input type="checkbox" name="rastreo" checked />
      Recuérdame
      <p>
      <input type="submit" name="submit" value="Ingresar" />
    </form>
  </body>
</html>
<?php
// si el formulario ha sido enviado
// verifica los datos proporcionados
// contra la base de datos
} else {
  $nombredeusuario = $_POST['nombredeusuario'];
  $contrasena = $_POST['contrasena'];

  // verifica datos de entrada
  if (empty($nombredeusuario)) {
    die('ERROR: Por favor escriba su nombre de usuario');
  }
  if (empty($contrasena)) {
    die('ERROR: Por favor escriba su contraseña');
  }
```

(*continúa*)

www.compumusicamania.com

```
  // intenta establecer conexión con la base de datos
  try {
     $pdo = new PDO('mysql:dbname=app;host=localhost', 'user', 'pass');
  } catch (PDOException $e) {
     die("Error: No fue posible conectar: " . $e->getMessage());
  }

  // limpia los caracteres especiales de los datos de entrada
$nombredeusuario = $pdo->quote($nombredeusuario);

  // verifica si existe el nombre de usuario
  $sql = "SELECT COUNT(*) FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
  if($result = $pdo->query($sql)) {
    $row = $result->fetch();
    // si es positivo, busca la contraseña cifrada
    if($row[0] == 1) {
      $sql = "SELECT contrasena FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
      // cifra la contraseña ingresada en el formulario
      // la verifica contra la contraseña cifrada que reside en la base
de datos
      // si ambas coinciden, la contraseña es correcta
      if ($result = $pdo->query($sql)) {
        $row = $result->fetch();
        $salt = $row[0];
        if (crypt($contrasena, $salt) == $salt) {
          // contraseña correcta
          // inicia una nueva sesión
          // guarda el nombre del usuario para la sesión
          // de requerirse, establece una cookie con el nombre de usuario
          // redirecciona el explorador a la página principal de la
aplicación
          session_start();
          $_SESSION['nombredeusuario'] = $nombredeusuario;
          if ($_POST['rastreo']) {
            setcookie('nombre', $_POST['nombredeusuario'],
mktime()+86400);
          }
          header('Location: principal.php');
        } else {
          echo 'Ha ingresado una contraseña incorrecta.';
        }
      } else {
          echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo-
>errorInfo());
      }
    } else {
      echo 'Ha ingresado un nombre de usuario incorrecto.';
    }
```

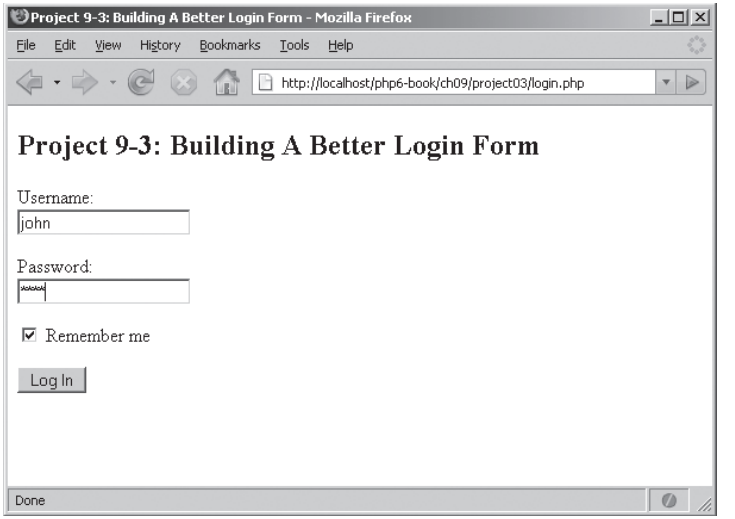


Figura 9-5 Un formulario Web para ingresar en una aplicación

```
  } else {
    echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo-
>errorInfo());
  }

  // cierra conexión
  unset($pdo);
}
?>
```

La figura 9-5 muestra el formulario de ingreso.

Cuando este formulario es enviado, la segunda mitad del script verifica el nombre de usuario y la contraseña contra los valores almacenados en la base de datos, utilizando el procedimiento explicado en el capítulo 7. Sin embargo, hay diferencias importantes: en esta versión, en lugar de simplemente generar un mensaje de éxito en caso de que el nombre de usuario y la contraseña sean válidos, el script comienza una nueva sesión y registra el nombre de usuario en una variable de sesión.

A continuación, el script verifica si se ha seleccionado la opción "Recuérdame" del formulario Web. De ser así, el script coloca una cookie en el equipo cliente con el fin de almacenar el nombre del usuario para posteriores visitas. La siguiente vez que el usuario visite esta página, la cookie establecida en el paso anterior será leída automáticamente y aparecerá el nombre de usuario en el campo correspondiente dentro del formulario. Una vez que se han establecido la sesión y la cookie, el script utiliza la función `header()` para redireccionar el explorador del usuario a la página principal de la aplicación, *principal.php*.

www.sampumusicmania.com

Pero esto es sólo la mitad de la historia. Con el fin de restringir el acceso sólo a los usuarios que han iniciado sesión, es necesario verificar la existencia de una sesión válida en otras páginas del sitio. Para ejemplificarlo, observa la página principal de la aplicación (*principal.php*), que implementa esta verificación:

```
<?php
// recrea sesión
// verifica si el usuario firmadota iniciado sesión
// de no ser así, muestra un mensaje de error y detiene el proceso
session_start();
if (!isset($_SESSION['nombredeusuario'])) {
  die('ERROR: Ha intentado ingresar a una página restringida. Por favor
<a href="login.php">Regístrese</a>.');
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 9-3: Construir un formulario de ingreso mejorado</title>
  </head>
  <body>
    <h2>Proyecto 9-3: Construir un formulario de ingreso mejorado</h2>
    Ésta es la página principal de la aplicación.
    <p/>
    Verás esta página después de ingresar exitosamente.
    <p/>
  </body>
</html>
```

La figura 9-6 muestra lo que verán los usuarios que ingresan con éxito.

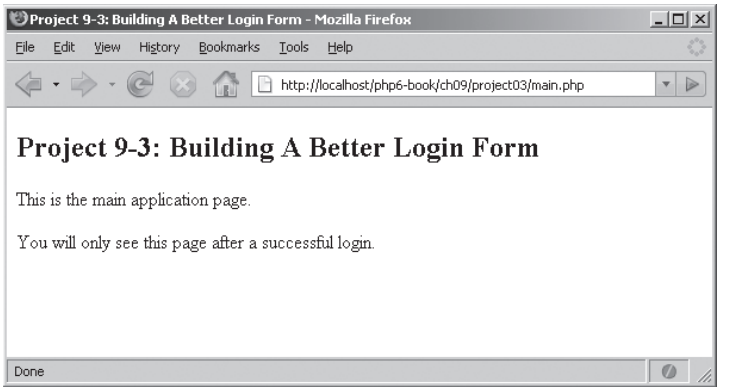


Figura 9-6 El resultado de acceder con éxito

www.compumusicmania.com

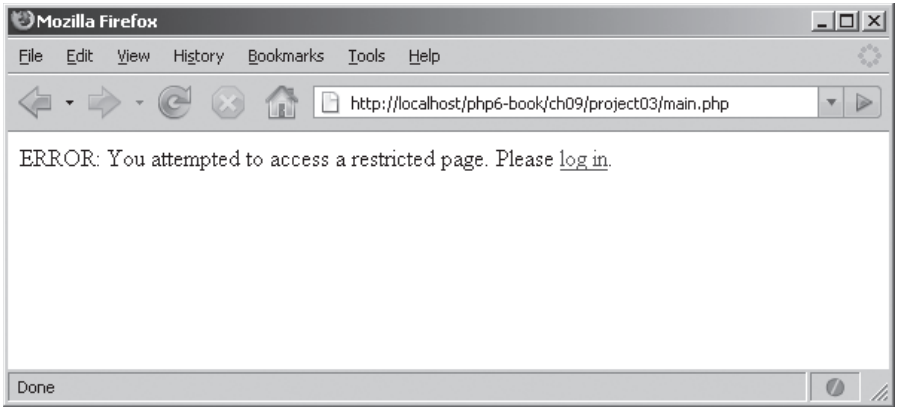


Figura 9-7 El resultado de acceder a una página segura sin haber firmado el acceso con anterioridad

La figura 9-7 muestra lo que verán los usuarios que intentan ingresar a la página sin haber iniciado sesión.

Resumen

Este capítulo mostró dos de las características más útiles y refinadas de PHP: la capacidad de trabajar con restricciones en el protocolo HTTP y la facultad de mantener el estatus a través de sesiones y cookies. Estas características suelen ser utilizadas por los desarrolladores para mejorar y enriquecer la experiencia de los usuarios en los sitios basados en PHP, y como comprobaste en las páginas anteriores, son muy fáciles de implementar.

Además de la información teórica sobre las sesiones, cookies y encabezados, este capítulo también incluyó ejemplos prácticos para poner en contexto real la teoría. Una tarea común en Web (permitir el acceso a ciertas partes del sitio sólo a los usuarios que han iniciado sesión) se utilizó para demostrar la manera en que pueden utilizarse sesiones, cookies y encabezados de manera conjunta para construir una aplicación segura y funcional.

Para leer más sobre los temas abordados en este capítulo, visita los siguientes sitios:

- Cookies en PHP, **www.php.net/setcookie**
- Sesiones en PHP, **www.php.net/session**
- Encabezados HTTP en PHP, **www.php.net/header**

www.compumusicmania.com

Autoexamen Capítulo 9

1. ¿Cuál es la diferencia entre una sesión y una cookie?
2. ¿Cómo eliminas una cookie establecida con anterioridad?
3. ¿Cómo se registra una variable de sesión? ¿Y cómo se accede a su valor desde una página diferente?
4. ¿Qué errores tiene el siguiente script PHP? Sin ejecutarlo, imagina cuáles serían los datos de salida.

```
<?php
echo 'Redireccionándolo ...';
header('Location: http://www.php.net');
?>
```

5. Escribe un programa que determine cuántas veces ha visitado una página en particular un usuario:
   - **A** Dentro de la misma sesión.
   - **B** En diferentes sesiones.
6. Revisa el último proyecto de este capítulo. Después, enriquécelo con un script PHP que saque a los usuarios de la aplicación y los redireccione al formulario de inicio de sesión.

www.compumusicmania.com

Parte III

Seguridad y solución de problemas

www.compumusicmania.com

Capítulo 10

Manejo de errores

www.compumusicmania.com

Habilidades y conceptos clave

- Comprender los niveles de error de PHP
- Controlar cuáles errores son desplegados en tu script PHP
- Desviar el manejador de errores por defecto de PHP y desviar los errores a una función personalizada
- Comprender cómo generar y manejar excepciones
- Enrutar automáticamente los errores a un archivo o una dirección de correo electrónico
- Generar una ruta alterna para depurar errores de script

Una mala interpretación común, sobre todo entre los desarrolladores poco experimentados, es concebir que un "buen" programa es el que funciona sin errores. En realidad, esto no es completamente cierto; una mejor definición sería que un buen programa es el que anticipa todas las posibles condiciones que provocarían errores y lidia con estas posibilidades de manera consistente y correcta.

Escribir programas "inteligentes" conforme a esta última definición es, a la vez, un arte y una ciencia. Experiencia e imaginación desempeñan un importante papel en la anticipación de causas potenciales de error y su respectiva acción correctiva, pero no menos importante es el lenguaje de programación, que define las herramientas y funciones que están disponibles para atrapar y corregir los errores.

Por fortuna, PHP no es perezoso en este aspecto: el lenguaje viene acompañado de un conjunto de sofisticadas herramientas que ayuda a los desarrolladores a capturar los errores y ponerles remedio. Este capítulo te presenta una introducción a este conjunto de herramientas; te mostrará el modelo de excepciones de PHP 5.3 y te enseñará a crear rutinas personalizadas para el manejo de errores a la medida de las necesidades de tu aplicación PHP.

Manejo de errores de script

A medida que has recorrido los proyectos de este libro, sin duda alguna has tenido algunos accidentes: una llave mal colocada por aquí, un punto y coma omitido por allá, tal vez alguna invocación equívoca en algún otro lugar. Y habrás notado que PHP es muy efectivo para señalar estos errores. En algunos casos, habrá generado un mensaje de error pero siguió ejecutan-

www.compumusicmania.com

do tu script; en otros casos, más serios, habrá detenido la ejecución del script con un mensaje que indica el número de la línea que causa el error.

Los tipos de error descritos son de "nivel de script"; surgen cuando el motor de PHP encuentra defectos en la sintaxis o la estructura de un script PHP. Por lo general, sólo se hacen visibles una vez que PHP comienza con la segmentación y ejecuta el script. Para ejemplificarlo, intenta crear y ejecutar el siguiente script:

```
<?php
// intenta dividir entre cero
echo 45/0;

// intenta invocar una función no definida
echo unaFuncion();
?>
```

Los datos de salida de este script deben ser semejantes a lo que aparece en la figura 10-1.

Como muestra la figura 10-1, este script genera dos tipos de errores: una "advertencia" por el intento de dividir entre cero, y un "error fatal" por el intento de invocar una función indefinida. En realidad, los errores de PHP pueden clasificarse en gran medida en tres grandes categorías, que se presentan en la tabla 10-1.

Existe una clara jerarquía de los mensajes de error en PHP: las notificaciones son menos serias que las advertencias, que a su vez son menos serias que los errores fatales. Por defecto, PHP sólo muestra advertencias y errores fatales en los datos de salida del script (aunque, como verás en unos momentos, puedes cambiar este comportamiento por defecto de manera que aun las notificaciones sean visibles en los datos de salida del script). Los errores pueden surgir en varias etapas durante la vida del script (al inicio, en la segmentación, en la compila-

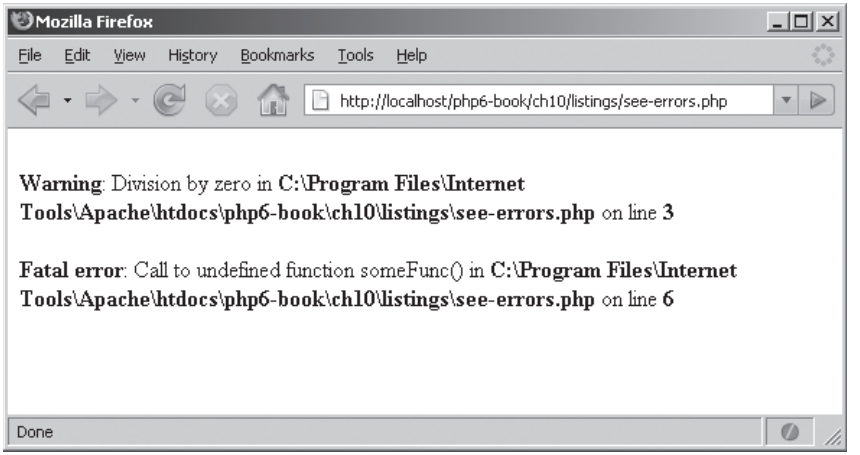


Figura 10-1 Ejemplo de página de error PHP

www.computeciencia.com

Tipo de error	Descripción	Ejemplo
Notificaciones	Errores no críticos que no detienen la ejecución del script	Acceder a una variable que no se ha inicializado
Advertencias	Errores más serios que requieren atención, pero no detienen la ejecución del script (aunque es posible que algunas partes del script no funcionen correctamente)	Leer un archivo que no existe en la ruta de acceso declarada
Errores fatales	Errores de sintaxis, o errores críticos que obligan a PHP a detener la ejecución del script	Crear una instancia de objeto de una clase indefinida

Tabla 10-1 Categorías de errores en PHP

ción o en la ejecución) y por lo mismo, PHP también hace distinciones internas de estas etapas. En conjunto, son doce diferentes niveles de error (más dos niveles "especiales"), representados por constantes. Puedes obtener una lista completa de estos niveles de error en www.php.net/manual/en/ref.errorfunc.php#errorfunc.constants; la tabla 10-2 presenta las constantes que verás con mayor frecuencia.

Es fácil entender casi todos estos niveles de error. Tal vez los únicos que presenten problemas sean los niveles `E_USER`, que se clasifican aparte de los errores personalizados en el nivel de la aplicación. No debes preocuparte por ellos, ya que fueron sustituidos por el nuevo modelo de excepciones introducido en PHP 5.

Nivel de error	Descripción
`E_PARSE`	Errores fatales de análisis sintáctico
`E_NOTICE`	Errores no fatales durante la etapa de ejecución (notificaciones)
`E_WARNING`	Errores no fatales durante la etapa de ejecución (advertencias)
`E_ERROR`	Errores fatales durante la etapa de ejecución que imponen la interrupción del script
`E_USER_NOTICE`	Errores no fatales definidos por el usuario (notificaciones)
`E_USER_WARNING`	Errores no fatales definidos por el usuario (advertencias)
`E_USER_ERROR`	Errores de aplicación fatal definidos por el usuario
`E_STRICT`	Errores no fatales durante la etapa de ejecución que surgen por errores de sintaxis PHP obsoleta
`E_ALL`	Todos los errores

Tabla 10-2 Niveles de error en PHP

www.compumusicmania.com

Controlar el reporte de errores

Puedes controlar cuáles errores mostrará el script con la función PHP integrada `error_reporting()`. Esta función acepta una o más de las constantes que aparecen en la tabla 10-2 y le indica al script únicamente los errores que coinciden con cierto tipo. Sin embargo, hay una excepción: los errores de segmentación (`E_PARSE`) que surgen por defectos en la sintaxis en el script PHP no pueden ocultarse con la función `error_reporting()`.

Para ver cómo funciona, considera el siguiente código modificado a partir de un ejemplo anterior; en este caso "oculta" los errores que no son fatales:

```
<?php
// muestra sólo los errores fatales
error_reporting(E_ERROR);
echo 45/0;
?>
```

En este caso, cuando el script se ejecuta no se generará ninguna advertencia, aunque esté intentando hacer una división entre cero.

Pregunta al experto

P: **¿Qué hace el nivel de error `E_STRICT`?**

R: En el nivel de error `E_STRICT`, PHP inspecciona tu código durante la ejecución y genera recomendaciones automáticas sobre la manera en que puede mejorarse. El uso de `E_STRICT` puede proporcionar recomendaciones sobre funciones que dejarán de existir en futuras versiones de PHP; aplicar tales recomendaciones puede mejorar el mantenimiento de tu código a largo plazo.

También puedes utilizar `error_reporting()` para evitar que aparezcan errores fatales durante la ejecución del script, como en el siguiente ejemplo:

```
<?php
//  muestra sólo las advertencias
error_reporting(E_WARNING);
echo unaFunción();
?>
```

Es importante destacar que `error_reporting()` no hace que el script quede libre de errores automáticamente; todo lo que hace es ocultar cierto tipo de errores. En el ejemplo anterior, aunque no se muestre un mensaje, se generará un error fatal y el script detendrá su ejecución en el punto donde se presenta el error.

www.compumusicmania.com

También puedes transmitir a `error_reporting()` una combinación de niveles de error, para personalizar aún más el sistema de reporte de errores de PHP. Observa el siguiente ejemplo, que sólo reporta notificaciones y errores fatales, pero no advertencias:

```
<?php
// notifica sólo notificaciones y errores fatales
error_reporting(E_NOTICE | E_ERROR);
echo $var; //  notificación
echo 45/0; // advertencia
echo unaFunción(); // fatal
?>
```

También es posible deshabilitar de forma selectiva el reporte de errores, con base en funciones específicas, insertando un prefijo en la invocación de la función con el operador @. Por ejemplo, en condiciones normales, el siguiente código generaría un error fatal porque `unaFunción()` no existe:

```
<?php
// invoca una función inexistente
echo unaFunción();
?>
```

Sin embargo, este error puede omitirse insertando el símbolo @ antes de invocar la función, de esta manera:

```
<?php
// invoca una función inexistente
@echo unaFunción();
?>
```

Utilizar un controlador de errores personalizado

Por defecto, cuando se dispara un error, el controlador de errores integrado de PHP identifica su tipo, muestra el mensaje de error apropiado [basándose en la configuración `error_reporting()`] y, como opción, detiene la ejecución del script (en caso de que el error sea fatal). El mensaje generado por el controlador de errores utiliza una hoja modelo estándar: indica el tipo de error, la razón del mismo, el nombre del archivo y el número de línea donde se generó (ver figura 10-1 como ejemplo).

Sin embargo, mientras tus aplicaciones PHP se hagan más complejas, este mecanismo para el manejo de errores puede terminar siendo inadecuado. Por ejemplo, tal vez quieras personalizar la hoja modelo utilizada por el controlador de errores para mostrar mayor o menor cantidad de información, o quizá desees enviar el error a un archivo o a una base de datos, en lugar de mostrarlo al usuario. Para todas estas situaciones, PHP ofrece la función `set_error_handler()`, que te permite reemplazar el controlador de errores integrado de PHP por uno propio.

www.compumusicmania.com

La función `set_error_handler()` acepta un solo argumento: el nombre de la función definida por el usuario que debe invocarse cuando ocurra el error. Esta función personalizada debe ser capaz de aceptar al menos dos argumentos obligatorios (el tipo de error y el mensaje descriptivo correspondiente) y un máximo de tres argumentos adicionales (el nombre del archivo, el número de línea donde ocurre el error y un lugar donde almacenar la variable en el momento del error).

NOTA

El controlador de errores personalizado no puede interceptar errores fatales `(E_ERROR)`, errores de análisis sintáctico `(E_PARSE)` ni notificaciones de sintaxis obsoleta `(E_STRICT)`.

Para mostrar cómo funciona, examina el siguiente ejemplo. Aquí, una función personalizada definida por el usuario reemplaza al controlador de errores integrado de PHP y genera dinámicamente una página de error personalizada:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
    <style type="text/css">
    .notice {
      font-weight: bolder;
      color: purple;
    }
    .warning {
      font-weight: bolder;
      font-size: larger;
      color: red;
    }
    </style>
  </head>
  <body>
<?php
// envía errores a un controlador personalizado
set_error_handler('miControlador');

// reporta todos los errores
error_reporting(E_ALL);

// genera algunos errores
echo $var;    // notificación
echo 23/0;    // advertencia
```

www.compumusicmania.com

```
// controlador de errores personalizado
function miControlador($type, $msg, $file, $line, $context) {
  $text = "Ha ocurrido un error en la línea $line mientras se procesaba
su solicitud. <p>
         Por favor visite nuestra <a href=http://www.dominio.com>página
de inicio</a> y vuelva a intentarlo.";
  switch($type) {
    case E_NOTICE:
      echo "<div class=\"notice\">$text</div><p>";
      break;

    case E_WARNING:
      echo "<div class=\"warning\">$text</div><p>";
      break;
  }
}
?>
  </body>
</html>
```

Aquí, la función `set_error_handler()` envía automáticamente todos los errores a la función `miControlador()` definida por el usuario. Cuando ocurre un error, esta función envía el tipo de error, mensaje, archivo, número de línea donde ocurrió el error, además de una variable de contexto. Luego muestra una página de errores personalizada que contiene el número de línea del error ya sea en color púrpura (notificaciones) o rojo (advertencias), y detiene manualmente la ejecución del script. La figura 10-2 muestra los datos de salida.

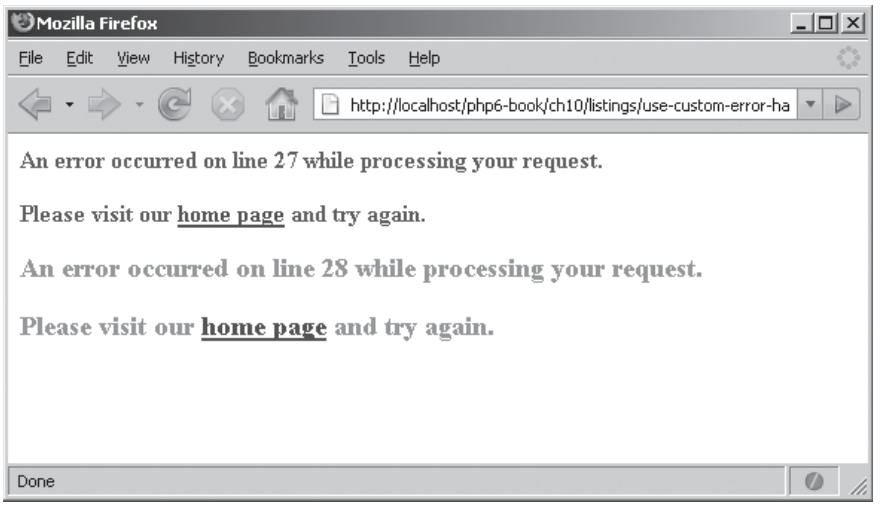


Figura 10-2 Página de error generada por un controlador de errores personalizado

www.comunidadmania.com

Pregunta al experto

P: ¿Cómo restauro el mecanismo controlador de errores por defecto de PHP una vez que haya invocado `set_error_handler()`?

R: Existen dos maneras. El método más sencillo es utilizar la función `restore_error_handler()`. Esta función restaura el último controlador de errores en uso antes de la invocación de `set_error_handler()`; en casi todos los casos, este controlador será el de PHP por defecto.

Otra opción es hacer que el controlador de errores personalizado regrese un valor falso; esto obligaría al error a transferirse de regreso al controlador PHP por defecto para ser procesado de manera normal. Esta técnica es muy útil si necesitas transmitir primero errores a través de una función personalizada para preprocesamiento o registro, y verás un ejemplo en la sección titulada "Registrar errores".

Prueba esto 10-1 Generar una página de errores legible

El controlador de errores de PHP sólo muestra información sobre un error. Muchas veces, este mensaje de error aparece mientras se genera la página que contiene los datos de salida, con lo que se destruye el diseño de la página y se crean confusión y estrés innecesarios para el usuario (ver figura 10-3 para conocer un ejemplo de este tipo de página). Sin embargo, al reemplazar el controlador de errores estándar por una función personalizada, es posible resolver este problema con facilidad generando una página de errores legible y enviar al mismo tiempo los errores del script a una base de datos para su posterior revisión. El siguiente ejemplo te muestra cómo hacerlo.

Para comenzar, crea una nueva base de datos SQLite, y una tabla que almacene los errores, como se muestra aquí:

```
shell> sqlite app.db
sqlite> CREATE TABLE errores (
...> id INTEGER PRIMARY KEY,
...> date TEXT NOT NULL,
...> error TEXT NOT NULL,
...> script TEXT NOT NULL,
...> line TEXT NOT NULL,
...>);
```

www.compumusicmania.com

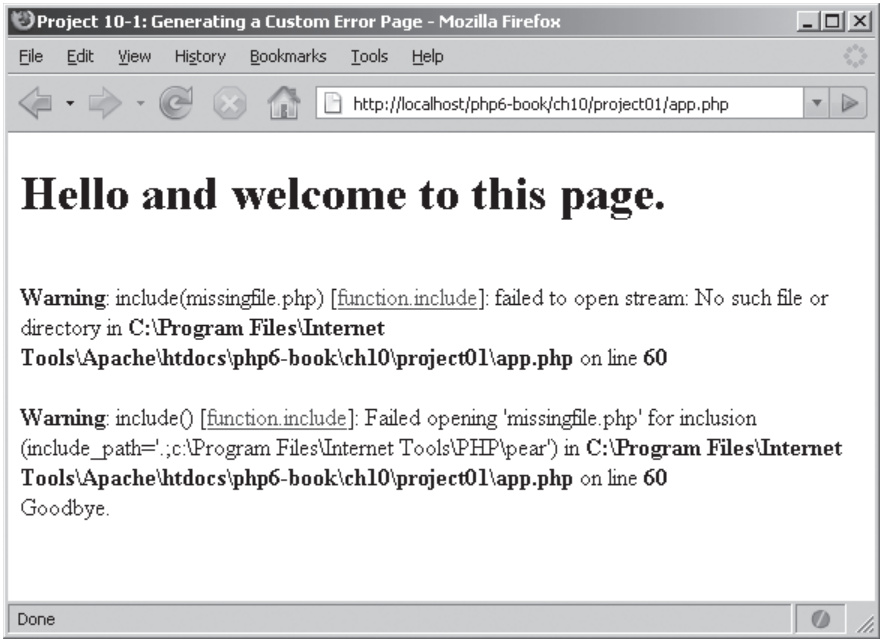


Figura 10-3 Datos de salida de errores entremezclada con el contenido del sitio

Después, define un controlador de errores personalizado que intercepte todos los errores del script y que los escriba en la tabla utilizando PDO, como en el siguiente script (*app.php*):

```
<?php
// reporta todos los errores
error_reporting(E_ALL);
// utiliza controlador personalizado
set_error_handler('miControlador');

// crea una región temporal de memoria (buffer)
ob_start();

// define un controlador personalizado
// que envíe los datos de error a la base de datos
// luego genera una página de error
function miControlador($type, $msg, $file, $line, $context) {
  // envía error a la base de datos
  $db = 'app.db';
  $pdo = new PDO("sqlite:$db");
  $msg = $pdo->quote($msg);
  $file = $pdo->quote($file);
  $line = $pdo->quote($line);
  $date = $pdo->quote(date('d-M-Y h:i:s', mktime()));
```

www.computeciencia.com

```
  $sql = "INSERT INTO errors (date, error, script, line) VALUES ($date,
$msg, $file, $line)";
  $pdo->exec($sql);

  // restablece y cierra el buffer
  // genera una nueva página de errores
  ob_end_clean();
  $errorPage = '
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "DTD/xhtml1-transitional.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <title>Página de Errores</title>
    </head>
    <body>
      <div style="border:solid 1px black; padding:10px; width:50%;
height:50%; margin:auto; top:0; bottom:0; right:0; left:0; position:
absolute">
        <h2>¡Perdón!</h2>
        Este script encontró un error interno y no es posible ejecutarlo.
        El error ya fue enviado y será rectificado a la brevedad.
        Hasta ese momento, por favor regrese a la página principal y
seleccione otra actividad.
      </div>
    </body>
  </html>';
  echo $errorPage;
  exit();
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 10-1: Generar una página de errores personalizada
</title>
  </head>
  <body>
<?php
// presenta algún texto de la página
echo "<h1>Hola y bienvenido a esta página.</h1>";

// genera advertencia (no se encuentra archivo)
include('archivofaltante.php');

// muestra alguna página de texto
echo "Adiós.";

// elimina el segmento de memoria reservado
ob_end_flush();
```

(*continúa*)

www.compumusicamania.com

```
?>
  </body>
</html>
```

Además de los componentes con los que ya estás familiarizado (PDO y controladores de errores personalizados), este script también presenta un nuevo elemento: funciones de control para los datos de salida de PHP. Como su nombre lo sugiere, estas funciones proporcionan un medio para que los desarrolladores apliquen un alto grado de control sobre los datos de salida generados por un script PHP.

Las funciones de control sobre los datos de salida funcionan desviando todos los datos de salida generados por el script hacia una *parte reservada de la memoria para datos de salida*, en lugar de mandarlos directamente al explorador cliente. El contenido de esta memoria reservada permanece activo hasta que los datos se hacen visibles de manera explícita para el usuario y pueden eliminarse al restablecer la memoria reservada para ellos.

Es necesario aprender tres funciones principales de la API de PHP sobre el control de los datos de salida:

- La función `ob_start()` inicializa la memoria reservada para datos de salida y se prepara para interceptarlos de un script. No es necesario decir que esta función debe invocarse antes de que se genere cualquier dato de salida por parte del script.
- La función `ob_end_flush()` termina con la memoria reservada para los datos de salida y envía su contenido a un dispositivo de salida (por lo general el explorador del usuario).
- La función `ob_end_clean()` termina con la memoria reservada para los datos de salida y limpia su contenido.

Con toda esta teoría en mente, regresemos al script anterior para ver cómo la memoria reservada para los datos de salida te ayuda a producir una página de errores más legible. Una rápida mirada al script y verás que comienza por inicializar una nueva memoria reservada para los datos de salida con `ob_start()`, y un controlador de errores personalizado con `set_error_handler()`. Ahora, todo error generado por el script será almacenado en la memoria reservada hasta que los datos se liberen hacia el cliente utilizando la invocación a `ob_end_flush()`.

Ahora, considera lo que sucede cuando ocurre un error en el script. Primero, el controlador de errores personalizado interceptará este error, abrirá un controlador PDO direccionado hacia la base de datos SQLite creada en el paso anterior, y convertirá el error (notificación o advertencia) en una consulta SQL `INSERT`. Luego, esta consulta se utilizará para guardar el error en la base de datos utilizando el método `exec()` de PDO, junto con la fecha y hora exactas. Después, la invocación de `ob_end_clean()` transmitirá la memoria reservada para los datos de salida, enviará un mensaje de error personalizado al explorador del usuario y terminará la ejecución del script.

www.compumusicmania.com

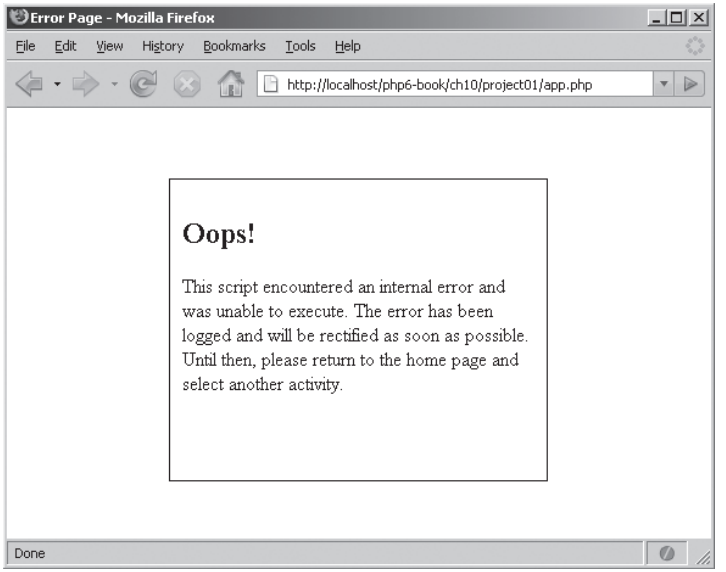


Figura 10-4 Página de errores legible

Como resultado de estas acciones, aunque existiera una página Web construida al vuelo cuando el error ocurre, nunca verá la luz del día, porque será descartada y reemplazada por la página de errores personalizada. Por otra parte, si el script se ejecuta sin errores, la invocación final a `ob_end_flush()` se encargará de enviar la página final completa al explorador.

Ahora, cuando ejecutes el script, en lugar de los datos de salida revueltos que se muestran en la figura 10-3, debes ver una página de errores legible. La figura 10-4 muestra lo que probablemente verás.

Mientras estás ahí, también mira la tabla de la base de datos SQLite. Debes encontrar los mensajes de error lanzados por el script, como se muestra a continuación:

```
1|13-Feb-2008 06:25:30|Undefined variable: myVar|/ch10/project01/app.
php|28
2|13-Feb-2008 06:25:30|include(archivofaltante.php): failed to open
steam: No such file or directory|/ch10/project01/app.php|31
3|13-Feb-2008 06:25:30| include(): Failed opening 'archivofaltante.php'
for inclusion (include_path='. ')|/ch10/project01/app.php|31
```

Una configuración como ésta facilita al administrador o desarrollador el mantenimiento de un registro permanente de los errores generados por el script y puede revisar el registro en cualquier momento para analizar o auditar errores; también asegura que la experiencia del usuario en el sitio sea consistente y sin complicaciones.

www.compumusicmania.com

Pregunta al experto

P: **¿Qué sucede si el propio controlador de errores personalizado contiene errores?**

R: Si el controlador de errores personalizado contiene errores, el mecanismo controlador de errores por defecto de PHP los manejará. De tal manera que, por ejemplo, si el código dentro del controlador personalizado de errores genera una advertencia, ésta será reportada dentro del nivel primario de PHP y será manejada por su controlador de errores por defecto.

Utilizar excepciones

Además de errores, PHP 5 también introduce un nuevo modelo de excepciones, similar al utilizado por otros lenguajes de programación como Java y Python. En este método basado en excepciones, el código del programa encerradote encierra en un bloque `try`, y las excepciones generadas por él son "atrapadas" y resueltas por uno o más bloques `catch`. Como es posible tener múltiples bloques `catch`, los desarrolladores pueden atrapar distintos tipos de excepciones y manejar cada una de ellas de manera diferente.

Para mostrar cómo funciona, examina el siguiente código, que intenta acceder a un elemento inexistente en una matriz utilizando un ArrayIterator:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi",
);

// intenta accesar a un elemento inexistente en la matriz
// genera un OutOfBoundsException
// datos de salida: 'Excepción: Posición 10 buscada y fuera de rango'
try{

  $iterator = newArrayIterator($ciudades);
  $iterator->seek(10);
} catch (Exception $e){
 echo 'ERROR: Ocurrió una excepción en tu script.';
}
?>
```

Cuando PHP encuentra código encerrado dentro de un bloque `try`, primero intenta ejecutar ese código. Si se procesa sin que se generen excepciones, el control se transfiere a las líneas que siguen al bloque `try-catch`. Sin embargo, si se genera una excepción mientras se

www.compumusicamania.com

ejecuta el código dentro del bloque `try` (como sucede en el ejemplo anterior), PHP detiene la ejecución del bloque en ese punto y comienza a verificar cada bloque `catch` para ver si hay un controlador para la excepción. Si se encuentra un controlador, el código dentro del bloque `catch` apropiado se ejecuta y luego se ejecutan las líneas que siguen al bloque `try`; en caso contrario, se genera un mensaje de error fatal y se detiene la ejecución del script en el punto donde se localiza el error.

Cada objeto Exception incluye información adicional que puede utilizarse para depurar la fuente del error. Es posible acceder a esta información mediante los métodos integrados en el objeto Exception e incluyen un mensaje de error descriptivo, un código de error, el nombre del archivo y el número de línea donde se encontró el error, así como un seguimiento de las invocaciones de la función que llevaron al error. La tabla 10-3 presenta una lista de estos métodos.

El siguiente código modificado a partir del ejemplo anterior muestra estos métodos en uso:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi"
);

// intenta accesar a un elemento inexistente en la matriz
// genera un OutOfBoundsException
try {

  $iterator = new ArrayIterator($ciudades);
  $iterator->seek(10);
```

Nombre del método	Lo que hace
`getMessage()`	Regresa un mensaje describiendo lo que salió mal
`getCode()`	Regresa un código de error numérico
`getFile()`	Regresa la ruta de acceso en disco y el nombre del script que generó la excepción
`getLine()`	Regresa el número de línea que generó la excepción
`getTrace()`	Regresa el seguimiento de las invocaciones que llevaron al error, como una matriz
`getTraceAsString()`	Regresa el seguimiento de las invocaciones que llevaron al error, como una cadena de texto

Tabla 10-3 Métodos del objeto de excepción PHP

www.compumusicmania.com

```
} catch (Exception $e) {
  echo "ERROR: ¡Algo salió mal!\n";
  echo "Error message: " . $e->getMessage() . "\n";
  echo "Error code: " . $e->getCode() . "\n";
  echo "File name: " . $e->getFile() . "\n";
  echo "Line: " . $e->getLine() . "\n";
  echo "Seguimiento: " . $e->getTraceAsString() . "\n";
}
?>
```

TIP

Es posible manejar diferentes tipos de excepciones de manera diferente, creando múltiples bloques `catch` y asignándoles diferentes acciones a cada uno. Verás un ejemplo de esto un poco más adelante en este mismo capítulo.

Ahora, si lo piensas, puedes estar tentado a creer que las excepciones son vino viejo en una botella nueva. Después de todo, parece que las capacidades descritas pueden replicarse fácilmente utilizando un controlador de errores personalizado, como se describió en la sección anterior. Sin embargo, en la práctica, este método basado en excepciones es mucho más sofisticado de lo que parece, porque ofrece los siguientes beneficios adicionales:

- En el modelo tradicional es necesario revisar el valor regresado de cada función invocada para verificar si ocurrió un error y realizar la acción correctiva. Esto puede producir código innecesariamente complicado, además de un anidamiento profundo de bloques de código. En el modelo basado en excepciones, puede utilizarse un solo bloque `catch` para atrapar cualquier error que ocurra mientras se procesa el bloque de código. Esto elimina la necesidad de múltiples cascadas de verificaciones para buscar errores, y produce un código más sencillo y fácil de leer.
- El modelo tradicional es prescriptivo por naturaleza: requiere que el desarrollador piense en todos los errores que pueden ocurrir, y que escriba el código que maneje cada una de estas posibilidades. En contraste, el método basado en excepciones es más flexible. Un controlador genérico de excepciones funciona como una red de seguridad, atrapando y manejando incluso los errores para los cuales no fue escrito ningún controlador específico. Esto ayuda a producir una aplicación más robusta y resistente a situaciones imprevistas.
- Dado que el modelo de excepciones utiliza un método orientado a objetos, los desarrolladores pueden utilizar conceptos de programación orientada a objetos como la herencia y extensibilidad para hacer subclases a partir del objeto Exception base y crear así diferentes objetos de excepción para múltiples tipos de excepciones. Esto hace posible distinguir entre diferentes tipos de errores, manejando cada uno de ellos de manera particular.
- El método basado en excepciones obliga a los desarrolladores a tomar decisiones cruciales sobre el manejo de diferentes tipos de errores. Al contrario del modelo tradicional, donde

www.compumusicamania.com

los desarrolladores pueden omitir fácilmente (por accidente o designio) las pruebas de verificación para los valores que regresa una función, las excepciones no son tan fáciles de ignorar. Al requerir que los desarrolladores creen y alimenten bloques `catch`, el modelo de excepciones los obliga a pensar en las causas y consecuencias de los errores y, a fin de cuentas, da como resultado un mejor diseño y una implementación más robusta.

¿El único inconveniente? Las excepciones se introdujeron hasta PHP 5 y, como resultado, son generadas de forma nativa sólo en las más recientes extensiones del lenguaje, como SimpleXML, objetos de datos PHP (PDO), objetos de servicio de datos (SDO, Service Data Objects) y la biblioteca estándar PHP (SPL, Standard PHP Library). Para el resto de las funciones de PHP es necesario verificar el valor que regresa la función para buscar errores y convertirlos manualmente en excepciones.

Crear, o *levantar*, manualmente excepciones de esta manera es una tarea para la declaración PHP `throw`. Ésta necesita transmitir un mensaje de error descriptivo y un código de error opcional, luego de lo cual genera un objeto Exception utilizando los parámetros y poniendo a la disposición del controlador de excepciones el mensaje y el código enviados. El proceso se ejemplifica en el siguiente código:

```
<?php
// establece el nombre del archivo
// intenta copiar y eliminar el archivo
$archivo = 'ejemplo.txt';
try {

  if (!file_exists($archivo)) {
    throw new Exception("Archivo '$archivo' no fue localizado.");
  }
  if (file_exists("$archivo.new")) {
    throw new Exception("Archivo destino '$archivo.new' ya existe.");
  }
  if (!copy($archivo, "$archivo.new")) {
    throw new Exception("Archivo '$archivo' no pudo ser copiado.");
  }
  if (!unlink ($archivo)) {
    throw new Exception("Archivo '$archivo' no pudo ser borrado.");
  }

} catch (Exception $e) {
  echo '¡Perdón! Algo malo ha ocurrido en la línea ' . $e->getLine() . ': ' .
$e->getMessage ();
  exit();
}
echo 'ÉXITO: la operación con el archivo fue correcta.';
?>
```

www.compumusicmania.com

Aquí, dependiendo del resultado de varias operaciones, el script lanza una nueva excepción manualmente. Un controlador estándar de excepciones atrapa más adelante la excepción, y se extrae la información específica del error y se muestra al usuario.

Utilizar excepciones personalizadas

Un método más avanzado consiste en crear subclases a partir del objeto genérico Exception y crear objetos específicos para cada posible error. Este método es útil cuando necesitas tratar diferentes tipos de excepciones de manera única, porque te permite utilizar bloques `catch` separados (y separar el código controlador) para cada tipo de excepción. He aquí una versión modificada del ejemplo anterior, que ilustra este método:

```
<?php
// subclases de Exception
class ExcepArchivoPerdido extends Exception {}
class ExcepArchivoDuplicado extends Exception {}
class ExcepArchivoIO extends Exception {}

// establece el nombre del archivo
// intenta copiar y eliminar el archivo
$archivo = 'ejemplo.txt';
try {

  if (!file_exists($archivo)) {
    throw new ExcepArchivoPerdido($archivo);
  }
  if (file_exists("$archivo.new")) {
    throw new ExcepArchivoDuplicado ("$archivo.new");
  }
  if (!copy($archivo, "$archivo.new")) {
    throw new ExcepArchivoIO("$archivo.new");
  }
  if (!unlink ($archivo)) {
    throw new ExcepArchivoIO($archivo);
  }

} catch (ExcepArchivoPerdido $e) {
  echo 'ERROR: No se encontró el archivo \'' . $e->getMessage() . ' \";
  exit();
} catch (ExcepArchivoDuplicado $e) {
  echo 'ERROR: El archivo destino \'' . $e->getMessage() . ' \' ya
existe';
exit();
} catch (ExcepArchivoIO $e) {
  echo 'ERROR: No fue posible realizar operación de entrada/salida en el
archivo \'' . $e->getMessage() . '\'';
  exit();
} catch (Exception $e) {
```

www.compumusicmania.com

```
  echo '¡Oops! Algo malo ha ocurrido en la línea ' . $e->getLine() . ': '
. $e->getMessage();
  exit();
}
echo 'ÉXITO: la operación con el archivo fue correcta.';
?>
```

Este script extiende la clase base Exception para crear tres nuevos tipos de excepciones, cada uno de los cuales representa un posible error. Un bloque `catch` individual para cada excepción ahora posibilita la personalización del tratamiento en cada caso específico. El último bloque `catch` es un controlador genérico "atrapatodo": las excepciones que no son controladas por alguno de los bloques específicos superiores caerán en este controlador que lidiará con ellos.

Pregunta al experto

P: **He visto que las excepciones que no son atrapadas generan un error fatal y hacen que el script termine abruptamente. ¿Puedo cambiar esto?**

R: Sí y no. PHP ofrece la función `set_exception_handler()`, que te permite reemplazar el controlador de excepciones por defecto de PHP con tu propio código personalizado, de manera muy similar a lo que hace `set_error_handler()`. Sin embargo, aquí hay un elemento muy importante a considerar. Como has visto, el controlador de excepciones por defecto de PHP muestra una notificación y luego da por terminada la excepción del script. Utilizar un controlador de excepciones personalizado tiene un control limitado sobre este comportamiento: puedes cambiar el modo y la apariencia de la notificación que aparece, pero no puedes hacer que el script continúe ejecutándose más allá del punto donde se generó la excepción.

La moraleja de la historia es que las excepciones que no son atrapadas *siempre* darán como resultado la terminación abrupta del script. Por eso es una buena idea incluir siempre un bloque `catch` genérico en tu código de manejo de excepciones para atrapar *todas* las excepciones que genera el script, independientemente de su tipo.

Prueba esto 10-2 Validar datos de entrada en un formulario

Ahora que ya tienes un conocimiento básico sobre el funcionamiento de excepciones, además de lanzar y atrapar excepciones personalizadas, apliquemos estos conocimientos en un pequeño proyecto que pone en práctica estas herramientas. El siguiente ejemplo permite que el usuario haga una solicitud para adquirir obras de arte, seleccionando un artista, el medio y el rango de precio. Esta orden se valida, se forma en un mensaje de correo electrónico y se

(*continúa*)

www.compumusicmania.com

entrega al servidor de correo para su envío. Los errores en el proceso se manejan mediante controladores personalizados, produciendo un código que es fácil de leer y mantener.

He aquí el código (*arte.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 10-2: Validar datos de entrada de un formulario</title>
    <style type="text/css">
    div.correcto {
      width:200px;
      padding:5px;
      border:solid 1px black;
      color:green;
    }
    div.error {
      width:400px;
      padding:5px;
      border:solid 1px black;
      color:red;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 10-2: Validar datos de entrada de un formulario</h2>
    <h3 style="background-color: silver">Compra de Arte Fino</h3>
<?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    if (!isset($_POST['submit''])) {
?>
    <form method="post" action="arte.php">
      Artista: <br />
      <select name="artista">
        <option value="">--Seleccione uno--</option>
        <option value="Picasso">Picasso</option>
        <option value="Van Gogh">Van Gogh</option>
        <option value="Chagall">Chagall</option>
        <option value="Degas">Degas</option>
        <option value="Monet">Monet</option>
        <option value="Matisse">Matisse</option>
      </select>

      <p>

      Medio: <br />
      <select name="medio">
        <option value="">--Seleccione uno--</option>
        <option value="óleo">Óleo</option>
```

www.compumusicamania.com

```
        <option value="acuarela">Acuarela</option>
        <option value="tinta»"Tinta</option>
      </select>

      <p>

      Rango de precio: <br />
        <input type="text" size="4" name="min" /> y <input type="text"
size="5" name="max" />
      <p>

      Dirección de correo electrónico: <br />
      <input type="text" size="25" name="email" />

      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si la forma ya fue enviada
    // procesa los datos de entrada
    } else {
      error_reporting(E_ERROR);

      // define clases de excepción
      class ExcepDatosEntrada extends Exception {}
      class ExcepLogica extends Exception {}
      class ExcepCorreo extends Exception {}

      // obtiene los datos de entrada del formulario
      $artista = $_POST['artista'];
      $medio = $_POST['medio'];
      $min = $_POST['min'];
      $max = $_POST['max'];
      $email = $_POST['email'];

      try {

        // valida datos de entrada del formulario
        if (empty($artista)) {
          throw new ExcepDatosEntrada('Artista');
        }
        if (empty($medio)) {
          throw new ExcepDatosEntrada('Medio');
        }
        if (empty($email)) {
          throw new ExcepDatosEntrada('Dirección de Correo');
        }
       if (empty($min) || empty ($max) || (int)$min <= 0 || (int)$max <=0) {
          throw new ExcepDatosEntrada('Precio');
        }
        if($max < $min) {
```

(*continúa*)

```
          throw new ExcepLogica('El precio máximo no puede ser menor que
el precio mínimo');
        }
        // envía correo electrónico con elección
        $subject = 'Orden de compra';
        $to = $email;
        $from = $email;
        $body = "
          DETALLES DE LA ORDEN: \r\n\r\n
          Artista: $artista \r\n
          Medio: $medio \r\n
          Precio: entre $min y $max \r\n
        ";
        if (!mail($to, $subject, $body, "From:$from")) {
          throw new ExcepCorreo();
        }

        // presenta mensaje de éxito
       echo '<div class="correcto">ÉXITO: ¡La orden fue procesada!</div>';

      } catch (ExcepDatosEntrada $e) {
        echo '<div class="error">ERROR: Por favor proporcione datos
válidos para el campo marcado \'' . $e->getMessage() . '\'</div>';
        exit();
      } catch (ExcepLogica $e) {
        echo '<div class="error"> ERROR: '. $e->getMessage() . ' </div>';
        exit();
      } catch (ExcepCorreo $e) {
        echo '<div class="error"> ERROR: Imposible enviar correo
electrónico</div>';
        file_put_contents('error.log', '[' . date("d-M-Y h:m:s",
mktime()) . '] Error de envío de correo a: ' . $e->getMessage() . "\n",
FILE_APPEND);
        exit();
      } catch (Exception $e) {
        echo '<div class="error">' . $e->getMessage() . ' en línea ' .
$e->getLine() . '</div>';
        exit();
      }
    }
?>
  </body>
</html>
```

La primera parte del script simplemente genera un formulario Web para que el usuario haga su elección, como se muestra en la figura 10-5.

Una vez que el formulario es enviado, el script desactiva el reporte de errores para todos los errores fatales y define tres nuevos tipos de excepciones: ExcepDatosEntrada para los datos de entrada en el formulario, ExcepLogica para los errores lógicos en los datos de entrada

www.compumusicamania.com

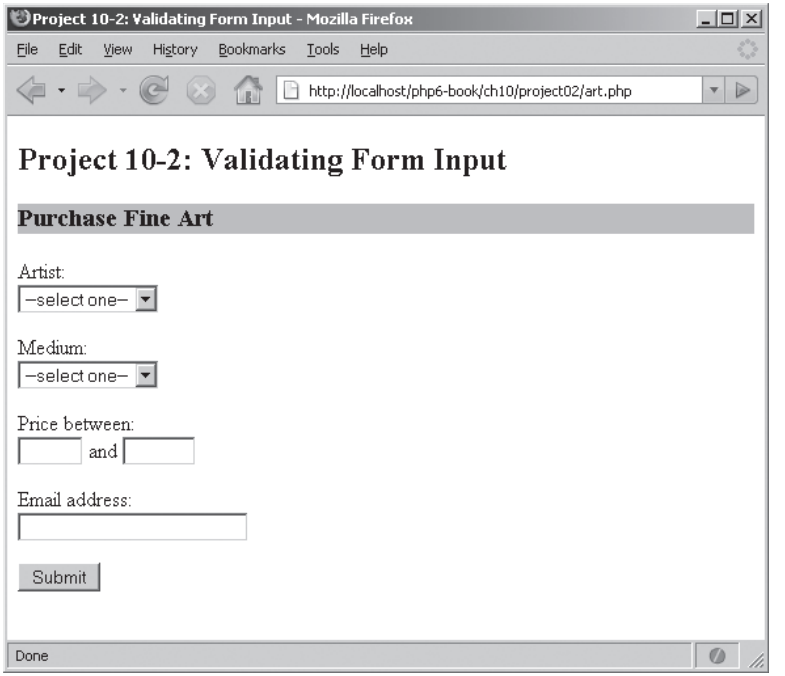


Figura 10-5 Formulario Web para que el usuario haga una orden de compra de arte fino

y ExcepCorreo para los errores en el envío del correo electrónico. Estas excepciones son simples extensiones del objeto genérico Exception, sin más adornos.

A continuación, se utiliza un bloque `try` para encapsular el proceso lógico del script. Primero, se verifica la consistencia de los campos del formulario; los errores en los campos de datos generan una excepción ExcepDatosEntrada o una ExcepLogica. Una vez que los datos han sido validados, se forman como un mensaje de correo electrónico, que es transmitido utilizando la función `mail()` de PHP. Si la función `mail()` regresa un valor falso, indica que el mensaje no fue enviado y surge una excepción ExcepCorreo; en caso contrario, aparece un mensaje de éxito.

Las excepciones generadas por el proceso lógico no son ignoradas; en cambio, cuatro bloques `catch` (uno para cada una de las excepciones definidas y uno genérico) atrapan estas excepciones, presentan al usuario la notificación apropiada del error y detienen el proceso del script. Cada una de estas excepciones es manejada de modo un poco diferente, para mostrar la manera en que se personaliza cada rutina de excepción: la excepción ExcepCorreo es enviada a un archivo con la fecha, hora y la dirección de correo electrónico del destinatario; por su parte, las excepciones ExcepDatosEntrada y ExcepLogica producen diferentes mensajes de error.

(*continúa*)

www.compumusicmania.com

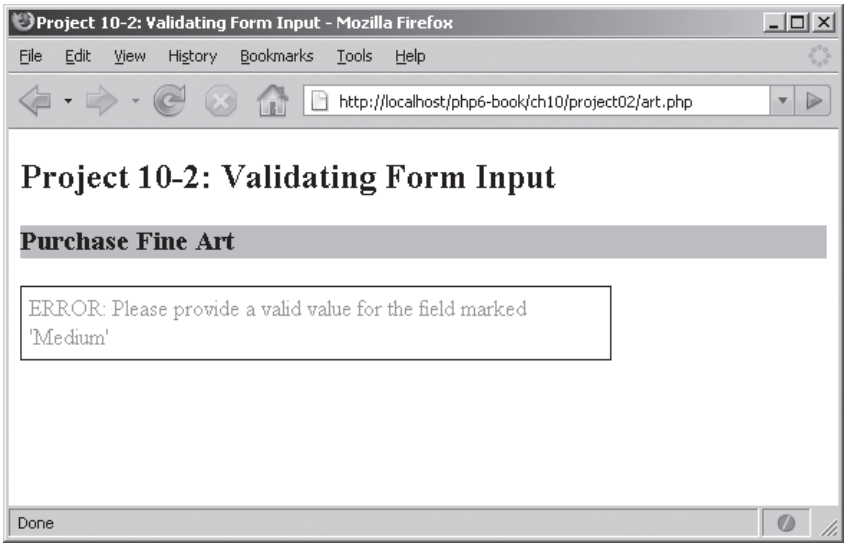


Figura 10-6 El resultado de atrapar una excepción ExcepDatosEntrada cuando se procesa el formulario Web

La figura 10-6 presenta el resultado que genera una excepción ExcepDatosEntrada.

La figura 10-7 presenta el resultado cuando falla la transmisión del correo electrónico y que genera una excepción ExcepCorreo.

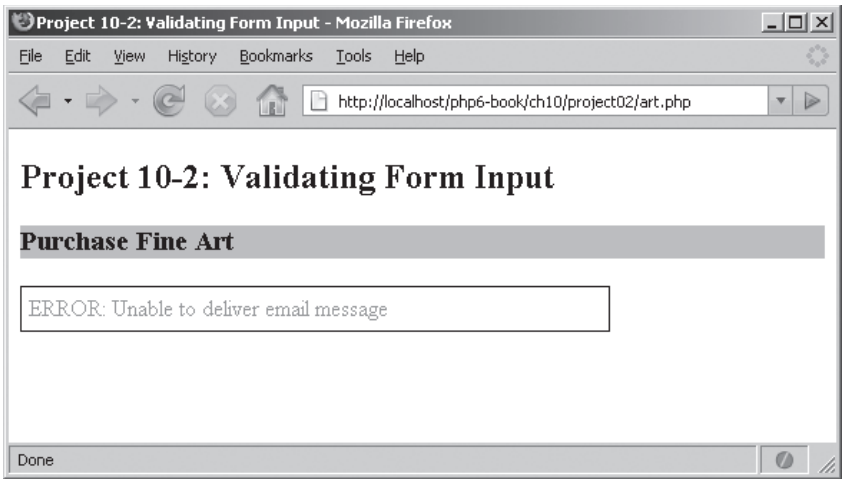


Figura 10-7 El resultado de atrapar una excepción ExcepCorreo cuando se procesa el formulario Web

www.compumusicmania.com

Registro de errores

Además de atrapar y manejar los errores en tiempo de ejecución, por lo regular también es una buena idea mantener un registro semipermanente de los errores que generó la aplicación, con el fin de inspeccionarlos y depurarlos. Ahí es donde entra en acción la función `error_log()` de PHP: te permite enviar los errores a un archivo de disco o a una dirección de correo electrónico en el momento en que ocurren.

La función `error_log()` necesita un mínimo de dos argumentos: el mensaje de error que debe enviarse y un valor numérico entero que indique el destino. Este valor entero puede ser 0 (el archivo de registros del sistema), 1 (una dirección de correo electrónico) o 3 (un archivo de disco). Para una dirección de correo electrónico o un archivo de disco se debe especificar la ruta de acceso correspondiente como tercer argumento para `error_log()`.

He aquí un ejemplo que muestra su funcionamiento:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if (mysqli_connect_error()) {
  error_log("ERROR: No se estableció la conexión. " . mysqli_connect_
error() . "\n", 3, 'debug.log');
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar query
// itera sobre colección de resultados
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysql ->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_array()) {
      echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";
    }
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  error_log("ERROR: No fue posible ejecutar $sql. " . $mysqli->error . "\
n", 3, 'debug.log');
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

www.compumusicmania.com

Suponiendo que se utilice un nombre de usuario o una contraseña incorrecta, el script enviará el siguiente mensaje al archivo *debug.log*:

```
ERROR: No se estableció la conexión. Acceso denegado para usuario
'usuario'@'localhost' (uso de contraseña: YES)
```

Es fácil combinar esta función de creación de reportes de error con el manejador de errores personalizado para asegurarse de que los errores del script queden almacenados en un archivo. He aquí un ejemplo que lo demuestra:

```
<?php
// controlador de errores personalizado
function miControlador($type, $msg, $file, $line, $context) {
    error_log("[" . date("d-M-Y h:m:s", mktime()) . "] $msg en la línea
$line de $file\n", 3, 'depuración.log');
    return false;
}

// reporta todos los errores
error_reporting(E_ALL);

// establece controlador personalizado
set_error_handler("miControlador");

// muestra E_NOTICE (variable indefinida)
echo $unaVariable;

// muestra E_WARNING (archivo perdido)
include("cualquier.php");
?>
```

En este script, todas las notificaciones y advertencias generadas por el código serán enviadas primero en automático al archivo *debug.log* con un sello cronológico. Después de ello, el controlador personalizado regresa un valor falso; esto se hace deliberadamente para que el control regrese al controlador de errores por defecto de PHP y lo maneje de la manera tradicional.

Depurar errores

Con aplicaciones más complejas se hace necesario encapsular las tareas más utilizadas en componentes independientes (funciones y clases) e importarlas conforme sea necesario a tus scripts PHP. Así como este enfoque mejora sin lugar a dudas el mantenimiento de tu código, también puede convertirse en una carga cuando las cosas no funcionan como es debido. Por ejemplo, es posible que un error en un componente afecte la función correcta de otros componentes, y rastrear ese error hasta su lugar de origen es una experiencia larga y frustrante.

www.compumusicmania.com

Existen varias herramientas para reducir la complejidad y laboriosidad de este proceso. La primera es proporcionada por PHP mismo, con la función `debug_print_backtrace()`. Esta función presenta una lista de todas las invocaciones de función que conducen a un error en particular, con lo que puedes identificar rápidamente la fuente del error. Para mostrarla en acción, revisa el siguiente script, que contiene errores deliberados:

```
<?php
// controlador de errores personalizado
// presenta un rastreo de invocaciones en caso de que ocurra un error
function miControlador($type, $msg) {
  debug_print_backtrace();
  exit();
}

class Hoja {
  public function_construct($num1, $num2) {
    $ex = new Ejemplo($num1, $num2);
  }
}

class Ejemplo {
  public function_construct($a, $b) {
    $this->a = $a+1;
    $this->b = $b-1;
    $this->run();
  }

  public function run() {
    return $this->a/$this->b;
  }
}

set_error_handler('miControlador');
$hoja = new Hoja(10,1);
echo 'Ejecución del código exitosa';
?>
```

Aquí, el script generará una advertencia de "división entre cero", no porque se trate de un error en la definición de la clase Hoja, sino porque el error es generado por el método `run()`, después de la invocación. La figura 10-8 muestra el mensaje de error que uno normalmente vería en este caso.

Depurar un error así en la práctica puede ser muy complicado, en especial cuando las definiciones de función y clase están contenidas en archivos separados. Sin embargo, como el controlador de errores en el script presenta un rastreo automático cuando ocurre el error, no es difícil identificar la función que lo causa. La figura 10-9 muestra el resultado modificado con rastreo automático.

www.compumusicmania.com

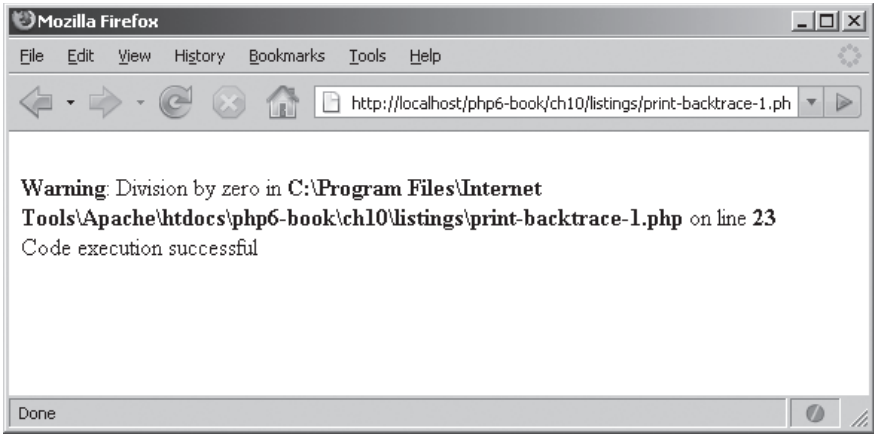


Figura 10-8 Página PHP de errores sin la información adicional de depuración

TIP

Para enriquecer el motor central de PHP con el fin de que añada automáticamente el rastreo de errores en todos los mensajes de error, intenta agregar la extensión gratuita Xdebug a tu configuración de PHP. Para más información e instrucciones de instalación visita www.xdebug.com/.

Un método aún más avanzado incluye el uso de la clase Debug de PHP, que se puede obtener gratuitamente en www.php-depuración.com/. Esta clase proporciona un conjunto de herramientas muy útil para rastrear la ejecución del script, observa y desecha variables, calcula tiempos de ejecución y realiza otras tareas de depuración comunes.

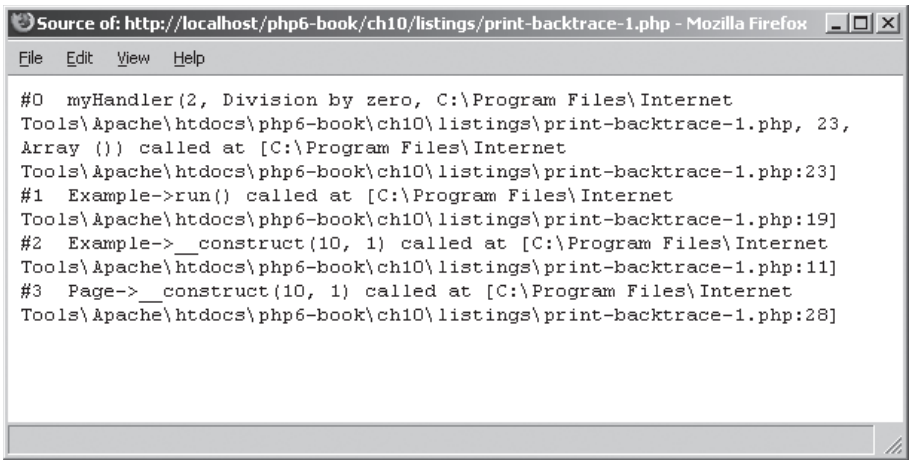


Figura 10-9 Página PHP de errores enriquecida con rastreo automático

www.compumusicmania.com

Para utilizar esta clase, descárgala y coloca sus archivos en el directorio de tu aplicación. Después, revisa el siguiente código para utilizarla, como sigue:

```
<?php
// controlador de errores personalizado
// presenta un rastreo de invocaciones en caso de que ocurra un error
function miControlador($type, $msg, $file, $line, $context) {
  global $debug;
  $debug->error($msg);
  echo '
  <!DOCTYPE html PUBLIC "-// W3C//DTD XHTML 1.0 Transitional//EN"
     "DTD/xhtml1-transitional.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <title>Página de Errores</title>
      <link rel="stylesheet" type="text/css" href="./PHP_Debug-1.0.0/css/
html_table.css" />
    </head>
    <body>
  ';
  $debug->display();
  echo '
    </body>
  </html>
  ';
  exit();
}

// añade ganchos de depuración a las clases y funciones
class Hoja {
  public function _construct($num1, $num2) {
    global $debug;
    $debug->add('Entering Page::_construct');
    $debug->dump($this);
    $debug->dump(func_get_args(), 'method args');
    $ex = new Ejemplo($num1, $num2);
  }
}

class Ejemplo {
  public function_construct($a, $b) {
    global $debug;
    $debug->add('Entering Hoja::_construct');
    $debug->dump($this);
    $debug->dump(func_get_args(), 'method args');
    $this->a = $a+1;
    $this->b = $b-1;
```

www.compumusicamania.com

```
    $this->run();
  }

  public function run() {
    global $debug;
    $debug->add('Entering Hoja::run');
    $debug->dump($this);
    $debug->dump(func_get_args(), 'method args');
    return $this->a/$this->b;
  }
}

// configura y lee archivos PHP_Debug
ini_set('incluir_ruta','./PHP_Debug-1.0.0/;');
include 'PHP_Debug-1.0.0/PHP/Debug.php;
include 'PHP_Debug-1.0.0/PHP/Debug/Renderer/HTML/TableConfig.php';
$options = array (
    'render_type'          => 'HTML',
    'render_mode'          => 'Table',
    'replace_errorhandler' => false,
);
$debug = new PHP_Debug($options);
set_error_handler('miControlador');

// comienza a ejecutar el código
$hoja = new Hoja(10,1);
echo 'Código ejecutado con éxito';
?>
```

La clase PHP_Debug proporciona un objeto con varias herramientas para ayudar a los desarrolladores a depurar sus scripts. Por ejemplo, el método `add()` del objeto puede utilizarse para rastrear la ejecución del script generando mensajes en ciertos puntos definidos por el usuario dentro del script, como pueden ser el comienzo de una función o de un bucle. De manera similar, el método `dump()` puede utilizarse para enviar el valor de una variable en determinado instante, mientras que el método `display()` puede usarse para enviar toda la depuración al dispositivo de salida. En la lista precedente, este método `display()` suele utilizarse como controlador de errores para que, en caso de que ocurra uno, genere de manera automática un rastreo nítido de todas las invocaciones que guiaron al error.

La figura 10-10 muestra los datos de salida del ejemplo anterior.

Como se aprecia en la figura 10-10, PHP_Debug puede proporcionar un medio más sofisticado para registrar todas las invocaciones de función que llevaron al error, en comparación con la función `depuración_print_backtrace()` de PHP.

www.compumusicmania.com

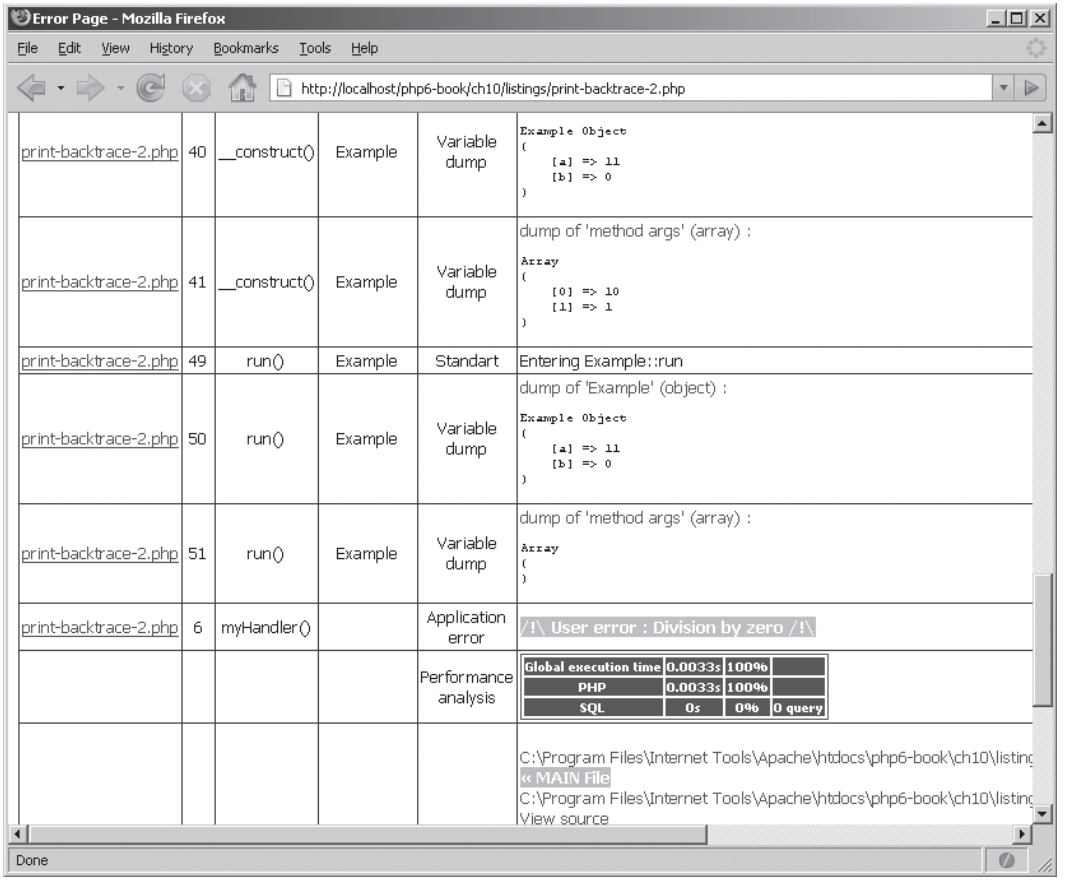


Figura 10-10 Rastreo generado por el paquete PHP_Debug

Resumen

Al finalizar este capítulo, debes tener una mejor idea de cómo funcionan las herramientas de depuración de PHP, y de la manera de personalizarlas para que se acoplen a las necesidades de una de tus aplicaciones en particular. Este capítulo abordó dos modelos: el antiguo, basado en errores, y el nuevo, basado en excepciones introducido en PHP 5. La facilidad de uso y la extensibilidad son puntos a favor del nuevo modelo en comparación con su antecesor, además de técnicas menos primitivas; sin embargo, como no cuenta con soporte completo en las bibliotecas principales de la aplicación, la mayoría de los programas desarrollados demandan una combinación juiciosa de ambos enfoques.

www.compumusicmania.com

Además de enseñarte todo lo relacionado con los errores y las excepciones, este capítulo también te presentó algunas funciones de utilidad para el almacenamiento de errores y la depuración de los mismos, y presentó una herramienta de terceros, el paquete PHP_Debug, para obtener un rastreo efectivo cuando ocurre un error. Finalmente, dos proyectos (una base de datos para almacenar mensajes de error y un validador para datos de entrada en un formulario Web) pusieron en práctica la teoría enseñada en el capítulo.

Para conocer más acerca de los temas abordados en este capítulo, visita las siguientes ligas:

- Excepciones, en **www.php.net/exceptions**
- Funciones para el manejo de errores, en **www.php.net/errorfunc**
- La página oficial de PHP_Debug, en **www.php-debug.com**

Autoexamen Capítulo 10

1. Menciona dos beneficios de utilizar el modelo basado en excepciones.

2. ¿Qué tipos de errores de script no pueden atraparse con un controlador personalizado de errores?

3. ¿Qué es un espacio de memoria reservado? ¿Por qué es útil? Proporciona un ejemplo práctico que demuestre su utilidad.

4. ¿Cuál controlador de errores se activará después de la última línea de cada uno de los siguientes bloques de código al ejecutarse?

```
<?php
set_error_handler('controladorA');
restore_error_handler();
?>

<?php
set_error_handler('handlerA');
set_error_handler('handlerB');
set_error_handler('handlerA');
restore_error_handler();
restore_error_handler();
?>
```

5. Utilizando un controlador de errores personalizado, muestra cómo convertir notificaciones y advertencias PHP en excepciones.

6. Escribe un programa que automáticamente envíe al correo electrónico del administrador las excepciones no capturadas en un script.

www.compumusicmania.com

Capítulo 11

Seguridad con PHP

www.compumusicmania.com

Habilidades y conceptos clave

- Familiarizarse con aspectos de seguridad en las aplicaciones PHP
- Prevenir tipos comunes de ataques limpiando los datos de entrada y de salida
- Incrementar la seguridad de tus archivos de aplicación, bases de datos y sesiones
- Aprender a validar cadenas de texto, números y fechas
- Comenzar a utilizar expresiones regulares para la validación de datos de entrada
- Aprender sobre la configuración de seguridad de PHP

En los capítulos anteriores aprendiste mucho sobre traer y utilizar datos de fuentes externas en tus aplicaciones PHP. Has procesado con éxito datos enviados mediante un formulario Web, has conectado tus páginas Web a bases de datos para generar páginas dinámicas y has recuperado información codificada en XML.

Ahora bien, la interacción con todas estas fuentes de datos externas puede hacer que tus aplicaciones PHP sean más relevantes y útiles, pero hay algunos riesgos que debes controlar. Las aplicaciones PHP inseguras son vulnerables a los ataques de usuarios maliciosos, y los datos de entrada mal validados pueden causar cálculos erróneos y reportes equivocados. Este tipo de vulnerabilidades pueden ser causa de corrupciones significativas y pérdida de datos, lo mismo que provocar vergüenza a los desarrolladores orgullosos.

Ahí es donde entra este capítulo. En las siguientes páginas se te presentará una introducción a varias técnicas que pueden utilizarse para validar los datos de entrada para tu aplicación, haciéndola más robusta y segura. Además, se ofrecerán ejemplos prácticos de higiene en los datos de entrada y salida, se abordarán algunas de las más importantes variables de seguridad PHP en el archivo de configuración y se te sugerirán varios recursos en línea donde puedes aprender más sobre la seguridad para aplicaciones PHP.

Higiene en los datos de entrada y salida

Como desarrollador de aplicaciones Web, tendrás que aprender a vivir con un desafortunado hecho: siempre habrá allá afuera gente que se divierte encontrando agujeros en tu código y explotándolos para propósitos malignos.

Casi siempre los ataques consisten en enviar a tu aplicación datos de entrada inteligentemente disfrazados que la "engañan" para que haga algo que realmente no debe hacer. Un

www.compumusicmania.com

ejemplo común de este tipo son los "ataques de inyección SQL", donde el atacante manipula desde afuera tu base de datos con una consulta SQL incrustada dentro de los datos de un formulario. Por ello, una de las tareas más importantes que debe realizar un desarrollador, antes de utilizar cualquier dato de salida enviado por un usuario, es higienizar los datos eliminando cualquier carácter especial o símbolo que pueda tener.

De manera similar, si tu aplicación va a utilizar datos provenientes de fuentes externas, siempre es necesario "limpiar" estos datos antes de mostrarlos al usuario. Una falla en este aspecto puede provocar que los atacantes incrusten contenido malicioso en tus páginas Web sin que lo notes. Un ejemplo común de este tipo es el "ataque de script de sitios cruzados", donde el atacante puede obtener acceso a datos sensibles de los usuarios al implantar de manera maliciosa código JavaScript o HTML como formulario en tus páginas Web. Con esto en mente, es siempre esencial higienizar por rutina los datos de salida antes de ponerlos a disposición de tus usuarios.

Por fortuna, PHP incluye varias funciones para ayudar a los desarrolladores en las tareas de saneamiento de datos de entrada y salida. He aquí una breve lista:

- La función `addslashes()` elimina caracteres especiales (como comillas dobles y diagonales invertidas) de los datos de entrada, de manera que sea seguro insertarlos en la base de datos. Como opción, utiliza el método `real_escape_string()` de MySQLi para sanear los datos de entrada antes de insertarlos en una base de datos MySQL, o bien la función `sqlite_escape_string()` para hacer lo mismo en bases de datos SQLite.
- La función `strip_tags()` permite a los desarrolladores eliminar todas las etiquetas HTML y PHP de una cadena de texto, regresando así sólo contenido ASCII. Esto puede ser útil para eliminar código potencialmente malicioso tanto de los datos de entrada de usuarios como de fuentes remotas.
- La función `htmlentities()` se encarga de reemplazar caracteres especiales como `"`, `&`, `<` y `>` por su correspondiente valor HTML. Al convertir estos caracteres especiales y evitar que sean interpretados como código HTML por el cliente, esta función es muy útil para "desarmar" los datos e incapacitarlos para afectar la apariencia y funcionalidad de la página Web.

He aquí un ejemplo donde se utiliza el método `real_scape_string()` y la función `strip_tags()` para incapacitar los datos de entrada antes de guardarlos en una base de datos MySQL:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user" "pass", "música");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}
```

www.compumusicmania.com

```
// limpia valores de entrada
if (isset($_POST['artista']) && !empty($_POST['artista'])) {
  $artista = $mysqli->real_scape_string(htmlspecialchars($_
POST['artista']));
}

if (isset($_POST['país']) && !empty($_POST['país'])) {
  $país = $mysqli->real_scape_string(htmlspecialchars($_POST['país']));
}

// intenta ejecutar query
// añade un nuevo registro
// datos de salida: " Nuevo artista con id: 7 ha sido añadido."
$sql = "INSERT INTO artistas (artista_nombre, artista_país) VALUES
('$artista', '$país')";
if ($mysqli->query($sql)=== true) {
  echo 'Nuevo artista con id: ' . $mysqli->insert_id . 'ha sido
añadido.';
} else {
  echo "ERROR: No fue posible ejecutar query: $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Y a continuación un ejemplo para higienizar los datos de entrada de un formulario con la función `htmlentities()`:

```
<?php
// define una matriz para la limpieza de datos
$limpia = array ();

// strip tags from POST input
if (isset($_POST['nombre']) && !empty($_POST['nombre'])) {
  $limpia['nombre'] = htmlentities($_POST['nombre']);
}

// código a procesar//
?>
```

Por último, un ejemplo para limpiar los datos de salida potencialmente peligrosos antes de presentarlos al usuario:

```
<?php
// define una matriz para la limpieza de datos
$limpia = array();

// obtiene datos remotos
```

www.compumusicmania.com

```
$out = 'Je<script>document.location.href="http://un.sitio.malo.com/";
</script>la';

// convierte todos los caracteres especiales en entidades
// antes de utilizarlas
$limpia['out'] = htmlentities($out);

echo $out; // inseguro
echo $limpia['out']; // seguro
?>
```

Asegurar los datos

Además de limpiar los datos de entrada, también es importante que tu aplicación no permita que los usuarios vean o manipulen los archivos o bases de datos privadas sin que se lo permitas. Las siguientes secciones abordan algunas técnicas que puedes utilizar para proteger el acceso a los archivos de configuración y otras fuentes de datos.

Asegurar los archivos de configuración

A menos que configures explícitamente tu servidor Web para que impida el acceso a ciertos tipos de archivo, un usuario remoto puede acceder a cualquier documento localizado bajo el archivo raíz del servidor . Esto hace que las aplicaciones Web sean muy vulnerables a los accesos remotos sin autorización, puesto que por lo regular guardan sus archivos de configuración con el resto del código de aplicación bajo el archivo raíz del servidor.

Una manera sencilla de rellenar este hueco de seguridad es almacenar cualquier dato de configuración sensible fuera del archivo raíz del servidor Web, y leerlo en tu aplicación conforme sea necesario mediante invocaciones `require()` o `include()`. Dado que ambas funciones aceptan rutas de acceso del sistema de archivos del sistema (en vez de rutas HTTP), pueden importar archivos de directorios que no forman parte del archivo raíz del servidor Web; por lo mismo se hace más difícil que los atacantes tengan acceso a los datos de configuración.

He aquí un ejemplo de lo que se puede hacer:

```
<?php
// la raíz de documentos del servidor es: /var/www/html/
// el archivo de configuración está guardado en: /var/www/apps/conf/

// tu script puede leer esta ruta de acceso
include_once'/var/www/apps/conf/myapp.conf'; // sí funcionará

// un atacante que sigue la misma ruta de acceso con HTTP no lo
conseguirá
file_get_contents ('http://localhost/../apps/conf/myapp.conf '); // no
funcionará
?>
```

www.compumusicamania.com

NOTA

Para que esto funcione, el directorio que contiene el archivo de configuración debe tener sus bits de permisos configurados de tal manera que el usuario propietario de los procesos del servidor Web pueda leer y modificar sus archivos. Esto es muy importante en los sistemas de desarrollo *NIX.

Asegurar el acceso a la base de datos

Una razón común para no aplicar seguridad al acceso de las bases de datos es porque se trata de una tarea "difícil" y "complicada". Esto no es completamente cierto. En casi todos los casos son sólo unos sencillos pasos que debes seguir para hacer más difícil que los atacantes tengan acceso a tu base de datos, con lo que se reduce drásticamente el riesgo de que le suceda algo malo a tu información. Dado que las bases de datos más utilizadas con PHP son MySQL, las siguientes tres sugerencias están directamente relacionadas con este motor de base de datos; sin embargo, también pueden aplicarse a cualquier otro RDBMS.

- **Dar a los usuarios sólo el nivel de acceso que necesitan** La mayor parte de las bases de datos, incluida MySQL, permiten mantener un control preciso sobre los niveles de acceso garantizados para usuarios individuales que utilizan la base. Hay una buena razón para usar este sistema de privilegios y permitir que los usuarios sólo tengan el acceso que necesitan: si se les concede permiso abierto a toda la base de datos, una sola cuenta en riesgo puede implicar pérdida o robo de importantes datos. MySQL ofrece los comandos `GRANT` y `REVOKE` para controlar los niveles de acceso para usuarios; diferentes comandos realizan la misma acción en otras bases de datos.

  El método recomendado aquí es definir un conjunto de usuarios individual para cada aplicación PHP, y permitirles el acceso únicamente a la base de datos que utiliza dicha aplicación. Esto significa que aunque la aplicación corra riesgo y un usuario malintencionado robe los permisos, la extensión del daño que puede provocar será limitada y no afectará a otras bases de datos en el mismo servidor.

- **Utilizar claves de acceso seguras** Cuando se instala por primera vez MySQL, el acceso al servidor de base de datos queda restringido únicamente al administrador (`'root'`). Por defecto, esta cuenta se inicializa con una clave de acceso en blanco, permitiendo acceso a cualquiera. Resulta sorprendente la frecuencia con la que los desarrolladores novatos suelen omitir este agujero en la seguridad; ellos permanecen ignorantes del peligro que implica seguir trabajando con la configuración por defecto. En MySQL, la contraseña se cambia con la herramienta *mysqladmin*; el proceso puede ser diferente para otras bases de datos.

  Para mejorar la seguridad, entonces, es una buena idea resetear la contraseña del administrador del servidor durante la instalación y luego distribuirla sólo entre un pequeño grupo

www.comprumusicmania.com

de usuarios que realmente necesiten conocerla. Una contraseña de administrador que es conocida por muchos usuarios es demasiado insegura; el viejo adagio "el pez por su propia boca muere" sigue siendo cierta desde los tiempos de nuestros abuelos.

- **Deshabilitar el acceso remoto** La configuración más común para una plataforma de desarrollo LAMP tiene el servidor de base de datos y el Web hospedados en la misma computadora física. En este caso, es posible reducir el riesgo de un ataque externo de manera importante al permitir sólo acceso "local" al servidor de base de datos y bloquear todas las conexiones externas. Con MySQL, esto se realiza utilizando la opción `–skip_networking` al iniciar el servidor; el proceso es diferente en otras bases de datos.

Asegurar las sesiones

En el capítulo 9 viste cómo las herramientas de manejo de sesión integradas a PHP pueden utilizarse para proteger páginas Web, y restringir el acceso a los usuarios que firmaron su entrada correctamente al sistema. Sin embargo, este sistema no es totalmente seguro: siempre resulta posible que un usuario malintencionado "robe" una sesión obteniendo acceso al identificador único de sesión y que lo utilice para recrear la misma.

La guía de seguridad de PHP, escrita por Chris Shiflett y otros, ubicada en www.phpsec.org/projects/guide/, recomienda varios métodos para impedir que los usuarios con malas intenciones roben las sesiones. Una de estas recomendaciones sugiere el uso de contraseñas adicionales para cada cliente con el objetivo de verificar su identidad. En esencia, esta técnica consiste en registrar varios atributos del cliente la primera vez que ingresa a una página manejada por sesión; por ejemplo, puede registrar el encabezado `'User_Agent'` para identificar el nombre del explorador del cliente, para luego verificar estos mismos atributos la próxima vez que visite la página. Si no hay coincidencia, significaría que la sesión del cliente ha sido robada y el acceso a la misma será denegado.

He aquí un ejemplo de cómo podría utilizarse esta técnica. El primer paso implica registrar el contenido del encabezado `'User_Agent'` del cliente como una variable de sesión, cuando la sesión se inicializa por primera vez:

```
<?php
// inicia sesión
// registra el nombre del cliente remoto como variable de sesión
session_start();
$_SESSION['verif_remote_agent'] = $_SERVER['HTTP_USER_AGENT'];
echo 'Sesión iniciada. Su ID de sesión es: ' . session_id();

// Código a procesarse //
?>
```

Ahora, en todos los accesos posteriores, además de verificar la presencia de una sesión válida, el script debe verificar también el encabezado `'User_Agent'` enviado por el cliente.

www.compumusicmania.com

```
<?php
// verifica sesión
// también verifica el nombre del cliente remoto
session_start();
if (!isset($_SESSION['verif_remote_agent']) || $_SESSION['verif_remote_
agent'] != $_SERVER['HTTP_USER_AGENT']) {
   die('La verificación de sesión falló.');
} else {
  echo 'Sesión verificada.';
}
// Código a procesarse //
?>
```

Este método, aunque sencillo y muy efectivo, no resulta completamente seguro: es posible que un atacante dedicado le dé la vuelta creando un encabezado `'User_Agent'`. Sin embargo, la verificación adicional incorpora un poco más de seguridad que antes no se tenía, porque el atacante tendría que adivinar primero el encabezado correcto. Así, se cuenta con un poco más de seguridad que antes.

Los detalles de las diferentes técnicas sugeridas en la guía de seguridad de PHP para hacer las sesiones más seguras rebasan los alcances de este capítulo; sin embargo, encontrarás muchos ejemplos bien documentados en www.phpsec.org/projects/guide/.

Validar los datos de entrada del usuario

Los formularios no son el único medio por el que un script PHP puede recibir datos de entrada; sin embargo, ése es el *medio más común*. Antes de utilizar estos datos, es necesario verificar su validez, para cortar los datos incorrectos o incompletos. Esta sección aborda varias técnicas que puedes utilizar para validar los datos de entrada del usuario, recibidos en un formulario Web o de cualquier otra manera.

Trabajar con campos obligatorios

Uno de los errores más comunes del programador novato consiste en olvidar la verificación de los valores en los campos requeridos dentro de un formulario Web. Esto puede dar como resultado una base de datos con numerosos registros vacíos, y uno de ellos puede, a su vez, afectar la exactitud requerida para los cálculos.

Una manera sencilla para verificar si todos los campos requeridos en un formulario han sido llenados consiste en revisar cada clave correspondiente en `$_POST` o `$_GET` con la función PHP `empty()`. Ésta verifica si la variable no está vacía y si contiene un valor diferente a cero. He aquí un ejemplo:

```
<?php
// define una matriz de datos válidos
```

www.compumusicmania.com

```
$valid = array();
// verifica el nombre de usuario
if (!empty($_POST['nombredeusuario'])) {
  $valid['nombredeusuario'] = trim($_POST['nombredeusuario']);
} else {
  die ('ERROR: Nombre de usuario ausente.');
}

// verifica contraseña
if (!empty($_POST['clavedeacceso'])) {
  $valid['clavedeacceso'] = trim($_POST['clavedeacceso']);
} else {
  die ('ERROR: Contraseña ausente.');
}

// Código a procesarse //
?>
```

En este listado, el script verifica primero si se enviaron los datos correspondientes al nombre del usuario y la contraseña. Si ambos resultan verdaderos, entonces el script continúa su ejecución; si cualquiera de ellos regresa un valor falso, el script termina de inmediato, sin intentar siquiera realizar el procesamiento de los datos del formulario.

Advierte también el uso de la matriz `$valid` en el ejemplo, que se utiliza para almacenar datos que han pasado la validación. Ésta es una buena práctica de programación para asegurarte de que no estés utilizando información que no haya sido validada con anticipación. Al asegurar que tu código sólo utilice datos provenientes de `$valid` se reduce el riesgo de insertar datos no válidos para tu base o tus cálculos.

Debemos hacer notar que la función `empty()` sólo se puede utilizar con variables de formulario en que el cero se considera un valor no válido. Esto se debe a que `empty()` regresa un valor falso si la variable que transmites contiene un valor `NULL`, una cadena de texto vacía (`' '`) o un cero como valor. Si el cero es considerado como valor válido en tu variable de formulario, reemplaza `empty()` por una combinación de funciones `isset()` y `trim()`, como en el siguiente ejemplo:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica incremento de oferta
if (isset($_POST['incremento']) && trim($_POST['incremento']) != '') {
  $valid['incremento'] = trim($_POST['incremento']);
} else {
  die ('ERROR: Valor de incremento ausente.');
}

// Código a procesarse //
?>
```

www.compumusicmania.com

La verificación de errores aquí es simple y lógica: la función `trim()` se utiliza para recortar espacios vacíos de la variable del campo, que es comparada después con una cadena vacía. Si la comparación resulta verdadera, significa que el campo fue enviado sin información y el script deja de ejecutarse y envía un mensaje de error.

A partir de estos ejemplos, debe quedar claro que una simple verificación condicional es todo lo que se requiere para asegurar que los campos requeridos de tu formulario nunca estén vacíos. No aplicar esta validación significa que los datos de entrada enviados por el usuario serán procesados sin comprobar que todos los valores requeridos estén presentes. Esto puede guiar a situaciones de riesgo: por ejemplo, el usuario podría registrarse con una contraseña en blanco, situación que puede traer consecuencias serias para la seguridad general de la aplicación.

Pregunta al experto

P: Ya valido los datos de entrada de mis formularios con JavaScript. ¿Por qué necesitaría validarlos también con PHP?

R: Es una práctica común utilizar lenguajes de script del lado del cliente, como JavaScript y VBScript, para validar los datos de entrada del lado del cliente. Sin embargo, este tipo de validación no es completamente segura. Dos casos sencillos lo remarcan:

- El código fuente de las páginas Web puede verse en casi todos los exploradores. Es posible que el usuario guarde el formulario Web, desactive la validación del lado del cliente editando el código fuente del formulario y la vuelva a enviar al servidor con valores ilegales.
- Si el usuario desactiva las funciones de JavaScript en su explorador, no se ejecutará el código del lado del cliente. Las rutinas de validación para el formulario serán omitidas por completo, con lo que de nuevo se abren las puertas para el ingreso de valores ilegales en el sistema.

La validación de entrada basada en PHP resuelve ambos aspectos de seguridad, porque el código PHP se ejecuta en el servidor y, por tanto, no puede modificarse ni deshabilitarse del sistema del cliente.

Trabajar con números

Como has visto, cuando defines una nueva tabla en una base de datos, también es necesario definir el tipo de datos que se ingresarán en cada campo. Sin embargo, diferentes sistemas de bases de datos difieren en el rigor aplicable en cada tipo de dato. Por ejemplo, SQLite permite el ingreso de cadenas de texto en los campos marcados como numéricos (`NUMERIC`), mientras que MySQL "corrige" automáticamente estos valores convirtiéndolos en 0 antes de insertarlos en un campo `INT` o `FLOAT`. Dadas estas diferencias, por lo general es una buena

www.compumusicmania.com

idea forzar la verificación del tipo de dato desde la aplicación PHP, para evitar que a tu base de datos lleguen valores incorrectos o capturados de manera equívoca.

Una manera fácil de verificar si una variable contiene un dato numérico es mediante la función de PHP `is_numeric()`, que regresa un valor verdadero cuando se invoca con números como argumento. El siguiente ejemplo muestra su funcionamiento:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si el campo edad es un número
if (is_numeric(trim($_POST['edad']))) {
  $valid['edad'] = trim($_POST['edad']);
} else {
  die ('ERROR: Edad no es un número.');
}

// Código a procesarse //
?>
```

Sin embargo, la función `is_numeric()` no distingue entre valores enteros y de punto flotante. Si necesitas este nivel de validación, una opción consiste en convertir primero la variable en un entero o valor de punto flotante y luego probarla con la función `strval()`. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si la edad es un valor entero
if (strval($_POST['edad']) == strval((int)$_POST['edad'])) {
  $valid['edad'] = trim($_POST['edad']);
} else {
  die ('ERROR: Edad no es un valor entero.');
}

// verifica si el precio es un valor de punto flotante
if (strval($_POST['precio']) == strval((float)$_POST['precio'])) {
  $valid['precio'] = trim($_POST['precio']);
} else {
  die ('ERROR: Precio no es un valor decimal.');
}

// Código a procesar //
?>
```

Aquí la lógica es muy simple: si la variable contiene un entero, el valor de la cadena regresado por `strval()` después de convertirlo en entero será idéntico al valor original de la variable.

www.compumusicmania.com

Por otra parte, si la variable contiene un valor que no sea numérico, el valor original de la variable no coincidirá con el valor obtenido después de la conversión.

Para una prueba aún más estrecha, utiliza la función `ctype_digit()` de PHP. Esta función verifica cada carácter del valor que le es transmitido, y regresa un valor verdadero sólo si cada carácter es un valor entre 0 y 9. He aquí un ejemplo:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si el campo edad es un número
if (ctype_digit($_POST['edad'])) {
  $valid['edad'] = trim($_POST['edad']);
} else {
  die ('ERROR: Edad no es un número.');
}

// Código a procesarse //
?>
```

PRECAUCIÓN

Advierte que el rigor de `ctype_digit()` puede ser contraproducente en algunos casos: la función regresará un valor falso cuando se transmitan valores decimales, porque el punto decimal no es un dígito entre 0 y 9.

En algunos casos tal vez desees reforzar un rango de valores numéricos que acepte tu aplicación. Verificar esto es tan sencillo como utilizar operadores de comparación PHP, como se muestra a continuación:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si el valor es un entero entre 1 y 31
if ((strval($_POST['day']) == strval((int)$_POST['day'])) &&
($_POST['day'] >= 1 && $_POST['day'] <= 31)) {
  $valid['day'] = trim($_POST['day']);
} else {
  die ('ERROR: Edad no es un número.');
}

// Código a procesarse //
?>
```

www.compumusicmania.com

Trabajar con cadenas de texto

Muchas bases de datos (incluida MySQL) truncarán de manera automática los valores de cadena de caracteres, si éstos exceden la longitud específica correspondiente al campo. Esto es inquietante porque significa que los datos de entrada proporcionados por el usuario pueden ser fácil y silenciosamente corrompidos sin que aparezca ninguna notificación al respecto. Por tanto, es una buena idea realizar una validación a nivel de aplicación para las cadenas de texto, para alertar a los usuarios en caso de que el texto rebase la longitud permitida y facultarlos para modificar la cadena.

Un buen lugar para comenzar con este tipo de validación es la función `strlen()`, que regresa la longitud de la cadena. Esto resulta útil para asegurar que los datos del formulario no excedan una longitud específica. Examina el siguiente ejemplo, que lo ilustra:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el nombre de usuario
if (!empty($_POST['nombredeusuario'])) {
  $nombredeusuario = trim($_POST['nombredeusuario']);
} else {
  die ('ERROR: Nombre de usuario ausente.');
}

// verifica la longitud del nombre de usuario
if (strlen($nombredeusuario) <= 25) {
  $valid['nombredeusuario'] = $nombredeusuario;
} else {
  die ('ERROR: Nombre de usuario demasiado largo.');
}

// Código a procesarse //
?>
```

Si necesitas estar completamente seguro de que los datos de un campo son sólo caracteres alfabéticos (por ejemplo, en los casos de nombres y apellidos), PHP ofrece la función `ctype_alpha()`. Al igual que la función `ctype_digit()` abordada en la sección anterior, esta función regresa un valor verdadero sólo si cada carácter dentro de la cadena es alfabético. He aquí un ejemplo:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el nombre de pila
if (isset($_POST['nombre']) && ctype_alpha($_POST['nombre'])) {
```

www.compumusicmania.com

```
  $valid['nombre'] = trim($_POST['nombre']);
} else {
  die ('ERROR: Nombre ausente o no válido.');
}

// Código a procesarse //
?>
```

TIP

PHP también ofrece la función `ctype_alnum()` para caracteres alfanuméricos, la función `ctype_space()` para espacios en blanco y la función `ctype_print()` para caracteres imprimibles. Para obtener la lista completa visita www.php.net/ctype.

Comparar patrones

Para una validación de cadenas de texto más compleja, PHP da soporte a *expresiones regulares*, una herramienta muy poderosa para validar patrones de cadenas de texto. Comúnmente asociada con la plataforma *NIX, una expresión regular te permite definir patrones utilizando un conjunto especial de *metacaracteres*. Estos patrones pueden compararse con el texto existente en un archivo, con los datos insertados en una aplicación o con los valores enviados en un formulario Web. Dependiendo de la coincidencia entre los patrones, los datos pueden ser considerados válidos o no.

Siguiendo los estándares de Perl, una expresión regular se encierra entre diagonales y por lo regular tiene esta apariencia:

```
/fo+/
```

El símbolo de adición (+) en esta expresión es un metacarácter: significa "coincide con una o más existencias del carácter anterior". En el contexto del ejemplo anterior, la expresión regular se traduce como "un patrón que contenga el carácter *f* seguido por una o más apariciones del carácter *o*". Por tanto, coincidirá con las palabras "fotografía", "formidable" y "fogata", pero no coincidirá con "frío" ni con "fatal".

Similares a + son los metacaracteres `*` y `?`. Son utilizados para coincidir con cero o más existencias de los caracteres anteriores y cero o una existencia de los caracteres precedentes, respectivamente. De esta manera, la expresión regular `/ma*/` coincidirá con "mamá", "manía" y "menor", mientras que la expresión `/com?/` coincidirá con "coincidencia", "consciente", "colateral", pero no coincidirá con "ciencia" ni con "cama".

También puedes especificar un rango con el número de coincidencias. Por ejemplo, la expresión regular `/jim{2,6}/` coincidiría con "jimmy" y con "jimmmmmmy", pero no con "jim". Los números encerrados entre llaves representan los valores menor y mayor del rango de la coincidencia; puedes dejar fuera el límite superior para un rango abierto.

www.compumusicmania.com

Metacarácter	Lo que significa
^	Inicio de una cadena de texto
$	Final de una cadena de texto
.	Cualquier carácter, excepto uno de nueva línea
\s	Un solo espacio en blanco
\S	Un solo carácter que no sea un espacio en blanco
\d	Un dígito entre 0 y 9
\w	Un carácter alfabético o numérico, o subrayado
[A-Z]	Un carácter alfabético en mayúsculas
[a-z]	Un carácter alfabético en minúsculas
[0-9]	Un dígito entre 0 y 9
|	Operador lógico OR (O)
(?=	Prueba condicional positiva
(?!	Prueba condicional negativa

Tabla 11-1 Metacaracteres de expresiones regulares

La tabla 11-1 muestra una breve lista de metacaracteres útiles.

Para ver el funcionamiento de éstos, imagina un formulario Web que requiere que el usuario introduzca su nombre de usuario, contraseña y número de seguro social. Al validar estos datos de entrada, el desarrollador necesita reforzar las siguientes restricciones:

- El nombre de usuario debe tener entre tres y ocho caracteres de longitud y contener sólo caracteres alfabéticos.
- La contraseña debe tener entre cinco y ocho caracteres de longitud y contener al menos un número.
- El número de seguro social debe contener nueve dígitos, y un guión después del tercer y quinto dígito.

Funciones como `strlen()` y `ctype_alnum()` son demasiado sencillas para este tipo de validación. En lugar de ellas, un mejor método consiste en definir patrones para cada valor de entrada, y comparar el dato de entrada contra el patrón para decidir si es válido o no. He aquí un ejemplo de la manera de hacerlo con expresiones regulares:

```
<?php
// define una matriz de datos válidos
```

www.compumusicmania.com

```
$valid = array();
// verifica nombre de usuario
if (isset($_POST['nombredeusuario']) && preg_match('/^([a-zA-Z]){3,8}$/',
$_POST['nombredeusuario'])) {
  $valid['nombredeusuario'] = trim($_POST['nombredeusuario']);
} else {
  die ('ERROR: Nombre de usuario ausente o no válido.');
}

// verifica contraseña
if (isset($_POST['clavedeacceso']) && preg_match('/^(?=.*\d).{5,8}$/',
$_POST['clavedeacceso'])) {
  $valid['clavedeacceso'] = trim($_POST['clavedeacceso']);
} else {
  die ('ERROR: Contraseña ausente o no válida.');
}

// verifica número de seguro social
if (isset($_POST['nss']) && preg_match('/^([0-9]){3}-([0-9]){2}-
[0-9]){4}$/', $_POST['nss'])) {
  $valid['nss'] = trim($_POST['nss']);
} else {
  die ('ERROR: NSS ausente o no válido.');
}

// Código a procesarse //
?>
```

Este código utiliza la función PHP `preg_match()` para probar si los datos de entrada se acoplan al patrón definido por la misma. Esta función `preg_match()` requiere dos argumentos obligatorios: un patrón y los valores que se compararán contra este patrón. Regresa un valor verdadero si se encuentra una coincidencia y falso en caso contrario.

Regresando a las expresiones regulares, no debe ser muy difícil decodificarlos después de revisar el material de la tabla 11-1. Especifican el rango de caracteres permitido para cada valor insertado y las longitudes mínimas y máximas para cada subconjunto de caracteres. Se les considerará válidos sólo si los datos de entrada coinciden con el patrón especificado.

He aquí otro ejemplo; éste valida números telefónicos internacionales:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el número telefónico
if (isset($_POST['tel']) && preg_match("/^(\+|00[1-9]{8,14}$/",
$_POST['tel'])) {
  $valid['tel'] = trim($_POST['tel']);
} else {
  die ('ERROR: Número telefónico ausente o no válido.');
}
?>
```

www.compumusicamania.com

Si juegas un poco con este código, verás que acepta los números *+441865123456* y *0091112345678*, aunque cada uno tiene un formato diferente. Esto se debe a que la expresión regular utiliza el metacarácter |, que funciona como el operador lógico OR (O) y hace posible crear un patrón que da soporte a opciones internamente. Por supuesto, puedes hacer el patrón más riguroso o más flexible, dependiendo de los requerimientos específicos de tu aplicación.

De esta manera, las expresiones regulares proporcionan una herramienta poderosa y flexible para validar los datos de entrada de acuerdo con una serie de reglas personalizadas... aunque cueste un poco acostumbrarse a la sintaxis.

Validar direcciones de correo electrónico y URL

Una tarea común cuando se trabaja con datos de entrada proporcionados por el usuario incluye verificar las direcciones de correo electrónico y los URL, para asegurar que tengan el formato correcto. Existen varias maneras de realizar esto; tal vez el método más común consista en utilizar expresiones regulares, como en el siguiente ejemplo:

```
<?php
// función para validar
// una dirección de correo electrónico
function validaemail($str) {
    return preg_match("/^([a-z0-9_-])+([\.a-z0-9_-])*@([a-z0-9-])+(\
.[a-z0-9-]+)*\.([a-z]{2,6})$/", strtolower($str));
}

// verifica dirección de correo electrónico
// datos de salida: 'válido'
echo validaemail("joe@dominio.com") ? "válido" : "no válido";

// verifica dirección de correo electrónico
// datos de salida: 'no válido'
echo validaemail("joe@dominio.") ? "válido" : "no válido";
?>
```

No es difícil encontrar una expresión regular para validar direcciones de correo electrónico; se puede encontrar una gran cantidad de opciones en línea, desde muy estrictas hasta más relajadas. El código anterior utiliza uno de los patrones más estrictos, que restringe el rango de los caracteres tanto para el nombre de usuario como para el dominio y requiere una longitud de caracteres para el nivel superior de dominio entre dos y seis caracteres.

También es posible escribir una función similar para validar URL. He aquí un ejemplo:

```
<?php
// función para validar URL
function validaUrl($str) {
    return preg_match("/^(http|https|ftp): \/\/([a-z0-9]([a-z0-9_-]*[a-
z0-9])?\.)+[a-z]{2,6}\/?([a-z0-9\?\._-~&#=+%]*)?/", strtolower($str));
}
```

www.compumusicmania.com

```
// verifica URL
// datos de salida: "válido"
echo validaUrl("http://www.ejemplo.com/html/index.php") ? "válido" : "no
válido";

// datos de salida: "no válido"
echo validaUrl("http://ejemplo.com") ? "válido" : "no válido";
?>
```

Los URL tienen diferentes formas y tamaños, como las direcciones de correo electrónico, y puedes seleccionar el grado de rigidez con el que los vas a validar. La expresión regular utilizada en el ejemplo anterior restringe los protocolos a HTTP, HTTPS y FTP; requiere que el nivel superior del dominio tenga una longitud entre dos y seis caracteres, y da soporte a rutas de acceso que contengan nombres de archivo y anclas.

Como opción, quizás una manera más sencilla de completar la misma tarea consiste en utilizar la función PHP `filter_var()`, que proporciona reglas de validación integradas para tipos comunes de datos de entrada, incluidas direcciones de correo electrónico y URL. He aquí una versión modificada del ejemplo anterior para validar una dirección de correo electrónico:

```
<?php
// función para validar
// una dirección de correo electrónico
function validaemail($str){
    return filter_var($str, FILTER_VALIDATE_EMAIL);
}

// verifica la dirección de correo electrónico
// datos de salida: 'válido'
echo validaemail("joe@dominio.com") ? "válido" : "no válido";

// verifica la dirección de correo electrónico
// datos de salida: 'no válido'
echo validaemail("joe@dominio.") ? "válido" : "no válido";
?>
```

Aquí, la función `filter_var()` prueba la variable enviada para ver si se trata de una dirección de correo electrónico válida, y regresa un valor verdadero o falso según el caso. La constante `FILTER_VALIDATE_EMAIL` le indica a `filter_var()` que verifique la variable contra el patrón que se espera para una dirección de correo electrónico. De manera similar, existe la constante `FILTER_VALIDATE_URL`, que puede utilizarse para probar la validez de los URL (aunque utiliza una prueba menos rigurosa que las expresiones regulares mostradas antes):

```
<?php
// función para validar una URL
function validaUrl($str) {
    return filter_var($str, FILTER_VALIDATE_URL);
```

www.compumusicamania.com

```
}
// verifica URL
// datos de salida: 'válido'
echo validaUrl("http://www.ejemplo.com/html/index.php") ? "válido" : "no
válido";

// datos de salida: 'no válido'
echo validaUrl("http:/ejemplo.com") ? "válido" : "no válido";
?>
```

Trabajar con fechas

Validar fechas es otro aspecto importante para el procesamiento de los datos de entrada. Sin la validación apropiada es muy fácil que el usuario introduzca una fecha no válida como 29 de febrero de 2009 o 31 de junio de 2008. Por ello, es importante asegurar que los valores de fecha proporcionados por el usuario sean genuinos antes de utilizarlos en los cálculos del programa.

En PHP, esta tarea es muy sencilla en comparación con otros programas, porque tiene la función `checkdate()`, que acepta tres argumentos: mes, día y año, y regresa un valor booleano que indica si la fecha es válida o no. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el día
if (!empty($_POST['día']) && ctype_digit($_POST['día'])) {
  $valid['día'] = trim($_POST['día']);
} else {
  die ('ERROR: Día ausente.');
}

// verifica el mes
if (!empty($_POST['mes']) && ctype_digit($_POST['mes'])) {
  $valid['mes'] = trim($_POST['mes']);
} else {
  die ('ERROR: Mes ausente.');
}

// verifica el año
if (!empty($_POST['año']) && ctype_digit($_POST['año'])) {
  $valid['año'] = trim($_POST['año']);
} else {
  die ('ERROR: Año ausente.');
}
// verifica la validez de la fecha
if (!checkdate($valid['mes'], $valid['día'], $valid['año'])){
  die ('ERROR: Fecha no válida.');
```

www.compumusicamania.com

```
}

// Código a procesarse//
?>
```

PRECAUCIÓN

Si estás almacenando fechas en los campos `DATE`, `DATETIME` o `TIMESTAMP` de MySQL, ten presente que esta base de datos *no* realiza ninguna verificación rigurosa de fecha por sí misma. Por lo tanto, la responsabilidad de verificar las fechas antes de guardarlas en una tabla de MySQL reside por completo en el desarrollador de la aplicación. Lo más que hará MySQL, en caso de encontrar una fecha *no* válida, será reemplazarla con una serie de ceros... que no es la mejor solución. Lee más sobre el manejo de fechas y horas por parte de MySQL en http://dev.mysql.com/doc/mysql/en/datetime.html.

Prueba esto 11-1 Validar datos de entrada de un formulario

Ahora que ya conoces los aspectos básicos sobre el saneamiento de los datos de entrada y la validación de los mismos, apliquemos estos conocimientos en un proyecto práctico. El siguiente ejemplo presenta un formulario Web que solicita al usuario el ingreso de varios detalles de un libro: título, autor, número ISBN y precio. Después valida estos datos utilizando una combinación de las técnicas abordadas en las secciones anteriores; una vez validados los datos, los guarda en una base de datos de SQLite.

Para comenzar, crea una base de datos de SQLite y una tabla para almacenar los registros ingresados por el usuario:

```
shell> sqlite libros.db
SQLite version 3.3.17
Enter ".help" for instructions
sqlite> CREATE TABLE libros (
...>  id INTEGER PRIMARY KEY,
...>  título TEXT,
...>  autor TEXT,
...>  isbn INTEGER,
...>  precio REAL
...>);
```

A continuación escribe el código PHP para validar los datos ingresados por el usuario mediante un formulario Web, y guárdalos en la base de datos. He aquí el script (*libros.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 11-1: Validar datos de entrada de un formulario</title>
     <style type="text/css">
```

www.compumusicamania.com

```
    div.error {
      color:red;
      font-weight: bolder;
    }
    div.correcto {
      color:green;
      font-weight: bolder;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 11-1: Validar datos de entrada de un formulario</h2>
    <h3 style="background-color: silver">Escriba los detalles del libro</h3>
<?php
    // muestra mensaje de error en la validación de los datos de entrada
    function getInputError($key, $errArray) {
      if (in_array($key, $errArray)) {
        return "<div class=\"error\">ERROR: Datos no válidos para el
campo '$key'</div>";
      } else {
        return false;
      }
    }

    $inputErrors = array();
    $submitted = false;

    // si el formulario ya fue enviado
    // valida los datos de entrada
    if (isset($_POST['submit'])) {
      $submitted = true;
      $valid = array();

      // valida título
      if (!empty($_POST['título'])) {
        $valid['título'] = htmlentities(trim($_POST['título']));
      } else {
        $inputErrors[] = 'título';
      }

      // valida nombre del autor
      if (!empty($_POST['autor']) && preg_match("/^[a-zA-Z\s.\-]+$/",
$_POST['autor'])) {
        $valid['autor'] = htmlentities(trim($_POST['autor']));
      } else {
        $inputErrors[] = 'autor';
      }

      // valida ISBN
```

(*continúa*)

www.compumusicamania.com

```
      if (!empty($_POST['isbn']) && preg_match('/^(97(8|9))?\d{9}
(\d|X)$/', $_POST['isbn'])) {
        $valid['isbn'] = htmlentities(trim($_POST['isbn']));
      } else {
        $inputErrors[] = 'isbn';
      }

      // valida precio
      if (!empty($_POST['precio']) && is_numeric($_POST['precio']) &&
$_POST['precio'] > 0)
{
        $valid['precio'] = htmlentities(trim($_POST['precio']));
      } else {
        $inputErrors[] = 'precio';
      }
    }

    // si el formulario no ha sido enviado
    // o si existe un error de validación
    // vuelve a mostrar el formulario
    if (($submitted == true && count($inputErrors) > 0) || $submitted ==
false) {
?>
    <form method="post" action="libros.php">
      Título: <br />
      <input type="text" size="25" name="título"
         value="<?php echo isset($_POST['título']) ? $_POST['título'] :
'';?>" /><br />
      <?php echo getInputError('título', $inputErrors); ?>
      <p>
      Autor: <br />
      <input type="text" size="25" name="autor"
         value="<?php echo isset($_POST['autor']) ? $_POST['autor'] :
'';?>" /><br />
      <?php echo getInputError('autor', $inputErrors); ?>
      <p>
      ISBN: <br />
      <input type="text" size="25" name="isbn"
         value="<?php echo isset($_POST['isbn']) ? $_POST['isbn'] :
'';?>" /><br />
      <?php echo getInputError('isbn', $inputErrors); ?>
      <p>
      Precio: <br />
      <input type="text" size="6" name="precio"
         value="<?php echo isset($_POST['precio']) ? $_POST['precio'] :
'';?>" /><br />
      <?php echo getInputError('precio', $inputErrors); ?>
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
```

```
<?php
    // si el formulario fue enviado sin errores
    // inserta el contenido en la base de datos
    } else {
      // abre la base de datos SQLite
      try {
        $pdo = new PDO('sqlite:libros.db');
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
      } catch (PDOException $e) {
        die("ERROR: No se estableció la conexión: " . $e->getMessage());
      }

      // crea un query de inserción
      try {
        $título = $pdo->quote($valid['título']);
        $autor = $pdo->quote($valid['autor']);
        $isbn = $pdo->quote($valid['isbn']);
        $precio = $pdo->quote($valid['precio']);
        $sql = "INSERT INTO libros (título, autor, isbn, precio) VALUES
($título, $autor, $isbn, $precio)";
        $ret = $pdo->exec($sql);
        echo '<div class="correcto">ÉXITO: Registro guardado;</div>';
      } catch (Exception $e) {
        echo '<div class="error">ERROR: ' . $e->getMessage() . '</div>';
      }

      // cierra conexión
      unset($pdo);
    }
?>
  </body>
</html>
```

La figura 11-1 muestra la apariencia del formulario Web.

Cuando se envía este formulario, el script comienza a trabajar validando los datos proporcionados por el usuario. Además de verificar que todos los campos tengan su respectivo valor, el script también utiliza expresiones regulares para probar el nombre del autor, el número de ISBN, y la función `is_numeric()` asegura que el valor insertado en el campo precio sea un número. Los campos que no tienen validación son marcados, añadiéndolos a la matriz `$inputErrors`. Los campos válidos son limpiados pasándolos por la función `htmlentities()` y luego añadiéndolos a la matriz `$valid`.

Una vez que se han completado las fases de validación y limpia, el script comienza a verificar si ha ocurrido algún error de validación. Suponiendo que no haya errores, se abre la conexión PDO hacia la base de datos SQLite, y los valores de la matriz `$valid` se integran en una consulta `INSERT` y se guardan en la base de datos. Sin embargo, si ocurre uno o más errores, el formulario se vuelve a desplegar y muestra un mensaje de error para que el usuario corrija los valores equivocados. Advierte la función definida por el usuario `getInput`

www.compumusicmania.com

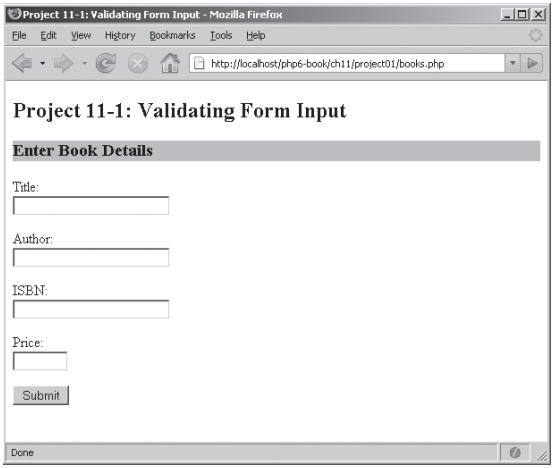


Figura 11-1 Formulario Web para insertar detalles de libros

`Error()`, que verifica y muestra el estatus de error de cada valor de entrada, proporcionando así una manera conveniente de notificar al usuario sobre todos los errores cometidos al mismo tiempo en lugar de mostrarlos uno tras otro después de cada corrección.

La figura 11-2 muestra la salida cuando existe error en alguno de los campos del formulario. La figura 11-3 muestra la salida cuando los datos de entrada son validados y guardados con éxito.

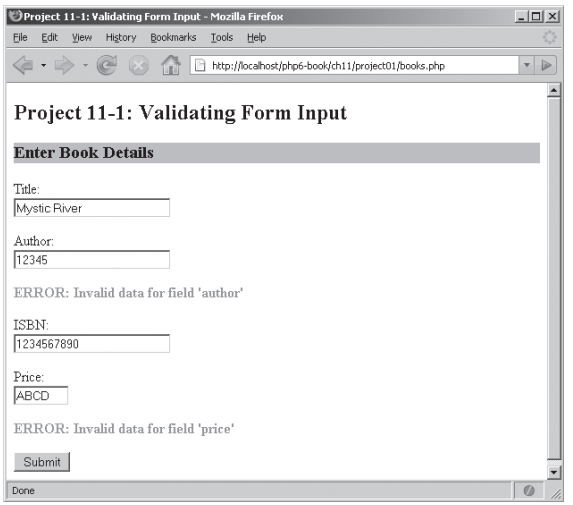


Figura 11-2 La salida cuando los datos del formulario tienen errores

www.comp.umusicmania.com

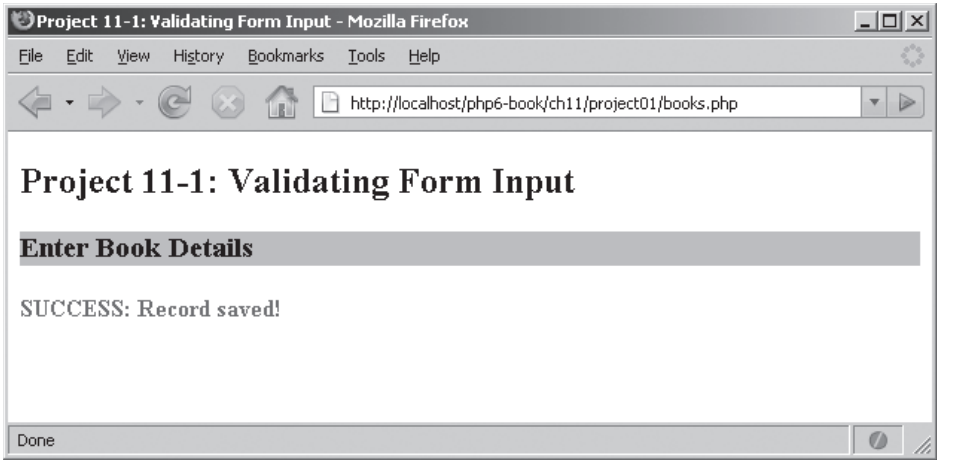


Figura 11-3 La salida cuando los datos del formulario son validados con éxito

Configurar la seguridad con PHP

Además de las técnicas presentadas en secciones anteriores, también existen varias directivas de configuración de PHP que puedes establecer con el fin de reducir el riesgo de que los atacantes obtengan acceso no autorizado a tu aplicación. Todas estas directivas pueden establecerse en el archivo de configuración de PHP, *php.ini*, o durante el periodo de ejecución con la función `ini_set()` de PHP. A continuación una breve lista:

- **`'disable_functions'`** La directiva `'disable_functions'` permite que los desarrolladores desactiven ciertas funciones integradas de PHP por razones de seguridad. Ejemplos de estas funciones son las que permiten la ejecución remota de comandos como `exec()` y `passthru()`, o bien funciones que muestran información interna de PHP o el servidor Web como `phpinfo()`.
- **`'disable_classes'`** Al igual que `'disable_functions'`, la directiva `'disable_classes'` permite que los desarrolladores desactiven ciertas clases PHP que impliquen cierto riesgo.
- **`'allow_url_fopen'`** La directiva `'allow_url_fopen'` determina si un script PHP puede leer datos de un URL remoto como si fueran archivos, con las funciones de archivo como `file_get_contents()`, `include()` o `fopen()`. A menos que tu aplicación necesite obtener datos de un URL remoto a través de HTTP o FTP, esta función debe quedar deshabilitada.

www.comunidadmania.com

- **`'open_basedir'`** La directiva `'open_basedir'` permite a los desarrolladores restringir todas las operaciones de archivo que tienen lugar dentro del script PHP a un directorio en particular y sus directorios secundarios. Cuando esta directiva está activa, un script PHP no podrá "ver" fuera del directorio de nivel superior mencionado en la directiva. Es útil para restringir la aplicación a un árbol de directorios definido, y reducir así la posibilidad de acceder a archivos de sistema sensibles, o manipularlos.
- **`'error_reporting'`** Ya viste la función `error_reporting()` en el capítulo anterior. Esta directiva `'error_reporting'` realiza la misma tarea: permite que el desarrollador controle el nivel de reporte de errores. La guía de seguridad de PHP recomienda que en la mayor parte de los casos sea configurada como `E_ALL`, de manera que todos los errores (notificaciones y advertencias) sean reportados al desarrollador.
- **`'display_errors'`** La directiva `'display_errors'` controla la posibilidad de que los mensajes de error generados por el script sean presentados o no en pantalla. El manual de PHP recomienda activar esta directiva en ambientes de desarrollo, pero deshabilitarla en ambientes de producción, porque los atacantes pueden utilizar la información del diagnóstico que muestra un mensaje de error para localizar y explotar vulnerabilidades en el código PHP.
- **`'log_errors'`** El hecho de que no se muestren los errores, no significa que deban ser completamente ignorados. La directiva `'log_errors'` especifica que los errores que ocurren en un script deben escribirse en un archivo de ingreso para su posterior análisis. Casi siempre, esta directiva debe estar activada, en especial si `'display_errors'` está deshabilitada, de manera que exista un registro de los errores generados por la aplicación.
- **`'expose_php'`** La directiva `'expose_php'` determina si PHP añade información sobre sí mismo al contexto del servidor Web. El manual PHP recomienda que se deshabilite esta directiva, para evitar que los atacantes obtengan información adicional sobre las capacidades del servidor.
- **`'max_input_time'`** La directiva `'max_input_time'` determina la cantidad máxima de tiempo que tiene un script PHP para recibir o interpretar datos de entrada, incluida la información transmitida por GET y POST. Un límite de este tiempo reduce el tiempo del que dispone un atacante para intentar la construcción y transmisión interactiva de una requisición POST o GET.
- **`'session.name'`** La directiva `'session.name'` controla el nombre de la cookie de sesión utilizado por PHP para rastrear la sesión del usuario. Por defecto, esta cookie recibe el nombre de `PHPSESSID`. Es buena idea cambiar este nombre, de nuevo con el fin de hacer más difícil que los atacantes identifiquen y vean su contenido.

La tabla 11-2 muestra una lista de dónde pueden establecerse estas directivas.

www.compumusicmania.com

Directiva	Puede ser establecida en *php.ini*	Puede ser establecida en tiempo de ejecución ini_set( )
'disable_functions'	Sí	No
'disable_classes'	Sí	No
'allow_url_fopen'	Sí	Sí
'open_basedir'	Sí	Sí
'error_reporting'	Sí	Sí
'display_errors'	Sí	Sí
'log_errors'	Sí	Sí
'expose_php'	Sí	No
'max_input_time'	Sí	No
'session.name'	Sí	Sí

Tabla 11-2 Directivas de seguridad de PHP

NOTA

Es necesario reiniciar el servidor Web para activar los cambios hechos en el archivo de configuración PHP *php.ini*.

Resumen

Este capítulo se concentró específicamente en la seguridad, presentando varias técnicas que puedes usar para reducir el riesgo de daño (ya sea intencional o accidental) a tu aplicación PHP. Te presentó las bases del saneamiento de datos de entrada y salida, explicó la manera de limpiar los datos de entrada para las bases de datos y de salida para terceros; te ofreció consejos para tomar medidas de seguridad en tus archivos de aplicación, bases de datos y sesiones. También te ofreció un curso relámpago sobre la validación de datos de entrada, mostrándote la manera de utilizar expresiones regulares, funciones de tipo de datos y pruebas condicionales con el fin de validar la información proporcionada por el usuario antes de utilizarla para realizar cálculos o insertarla en tu base de datos.

Este capítulo cubrió mucho terreno, pero es sólo la punta del *iceberg*: la seguridad en PHP es un tema muy amplio y construir una aplicación robusta, resistente a ataques, requiere tanto conocimiento como experiencia. Por fortuna, no es difícil adquirir experiencia en este tema; existen numerosos recursos disponibles en línea para ayudarte a aprender más sobre potenciales ataques y a cerrar huecos de seguridad en el código de tus aplicaciones. Te invitamos a visitar las siguientes direcciones para aprender más sobre los temas tratados en este capítulo:

- Una panorámica sobre temas de seguridad en PHP, en **www.php.net/security**
- La Guía de Seguridad PHP, en **www.phpsec.org/projects/guide/**

www.computermusicmania.com

- Artículos y discusiones sobre la seguridad en PHP por PHP Security Consortium, en **www.phpsec.org/**
- Funciones de expresiones regulares, en **www.php.net/pcre**
- Funciones de tipos de caracteres, en **www.php.net/ctype**
- Discusiones sobre ataques de sitios cruzados, en **http://en.wikipedia.org/wiki/Cross-site_scripting**
- Una discusión sobre ataques de inyección SQL, en **http://en.wikipedia.org/wiki/SQL_injection**
- Ejemplos de ataque de sitios cruzados, en **http://ha.ckers.org/xss.html**
- Ejemplos de expresiones regulares, en **www.regexlib.com**
- Tutoriales sobre expresiones regulares, en **www.melonfire.com/community/columns/trog/article.php?id=2** y **www.regular-expressions.info/tutorial.html**
- Tutoriales sobre validación de datos del lado del cliente en **www.sitepoint.com/article/client-side-form-validation** y **http://home.cogeco.ca/~ve3ll/jstutor5.htm**

Autoexamen Capítulo 11

1. Menciona y explica dos ataques abordados en este capítulo. También explica cómo defender tu aplicación contra los mismos.
2. Menciona y ejemplifica dos funciones para limpiar los datos de salida de una aplicación.
3. Demuestra el uso de una expresión regular para validar códigos postales de Estados Unidos, con el formato *ddddd-dddd*.
4. ¿Por qué debe deshabilitarse el despliegue de errores en los ambientes de producción?
5. Explica lo que hacen las siguientes funciones:
   - `ctype_alnum()`
   - `addslashes()`
   - `filter_var()`
   - `htmlentities()`
   - `sqlite_escape_string()`
   - `preg_match()`
   - `strval()`

www.compumusicmania.com

Capítulo 12

Extender PHP

www.sempumusicmania.com

Habilidades y conceptos clave

- Aprender sobre los depósitos de código PEAR y PECL
- Entender cómo instalar un paquete PEAR y uno PECL
- Comunicarse con un servidor POP3 utilizando el paquete PEAR
- Crear dinámicamente un archivo ZIP utilizando el paquete PECL

Como un lenguaje de código libre, PHP tiene el soporte de miles de desarrolladores de todo el mundo. El soporte de esta comunidad, junto con la facilidad de uso del lenguaje, ha producido cientos de aplicaciones auxiliares y extensiones que pueden ser utilizadas para añadir nuevas capacidades al motor original de PHP. Estos auxiliares y extensiones conforman una base robusta y estable para el código que es invaluable para los desarrolladores, porque les permite crear rápidamente aplicaciones Web de alta calidad y eficiencia sin necesidad de "escribir mucho código".

Dos de los más extensos depósitos en línea para estos complementos son el depósito de extensiones y aplicaciones PHP (PEAR, PHP Extension and Application Repository), y la biblioteca comunitaria de extensiones PHP (PHP Extension Community Library). Este capítulo presenta una introducción a ambos; utiliza ejemplos prácticos para demostrar cómo pueden hacer más simple y efectivo el desarrollo de aplicaciones PHP.

Utilizar PEAR

PEAR es el depósito de extensiones y aplicaciones PHP, disponible en la dirección Web http://www.pear.php.net/. Su propósito es proporcionar a los desarrolladores una biblioteca de clases PHP reutilizables (también llamadas *paquetes*), que pueden integrarse fácilmente en cualquier aplicación PHP y que siguen el estilo estándar de codificación, así como la estructura de archivos. Los paquetes de PEAR son distribuidos como archivos TAR comprimidos y pueden instalarse en cualquier sistema de desarrollo PHP utilizando el instalador PEAR (incluido con todas las distribuciones de PHP).

Los paquetes PEAR abarcan una diversa amalgama de categorías. Algunas de ellas son:

- Autentificación del usuario (Auth, Auth_HTTP).
- Integración de bases de datos (MDB, DB_DataObject, DB_QueryTool, Query2XML y Structures_DataGrid).
- Procesamiento de formularios (HTML_QuickForm, Validate y Text_CAPTCHA).
- Protocolos de red (Net_SMTP, Net_POP3, NET_LDAP y Net_FTP).

www.compumusicmania.com

- Formatos de archivo (File_PDF, Archive_TAR y Spreadsheet_Excel_Writer).
- Localización de aplicaciones (I18N).
- Comparaciones, ingreso y prueba de unidades (Estudio comparativo, Log, PHPUnit).

Instalar paquetes PEAR

PHP incluye un instalador automático para los paquetes PEAR. Este instalador tiene la capacidad de conectarse automáticamente con el servidor central PEAR, descargar los paquetes requeridos e instalarlos en tu ambiente de desarrollo PHP.

NOTA

Los usuarios de Windows primero deben configurar manualmente el instalador PEAR, ejecutando el archivo por lotes *go-pear.bat*, localizado en el directorio de instalación de PHP. Este archivo configurará el instalador de PEAR, generará los registros necesarios y colocará los archivos de instalación en su ubicación correcta dentro del sistema. Para mayores detalles, revisa las notas de instalación para Windows en el manual de PEAR, en http://pear.php.net/manual/en/installation.getting.php.

Para instalar un paquete de PEAR utilizando el instalador, simplemente escribe el siguiente comando en la consola:

```
shell> pear install nombre-del-paquete
```

El instalador de PEAR se conectará con el servidor de paquetes PEAR, descargará el paquete solicitado y lo instalará en la ubicación adecuada dentro del sistema. La figura 12-1 muestra un ejemplo de la instalación del paquete Net_POP3.

Figura 12-1 Instalación de un paquete de PEAR en UNIX

www.compumusicmania.com

TIP

Si no puedes hacer que el instalador PEAR funcione correctamente, intenta añadir la opción –v a la línea de comando para la instalación, para que muestre información adicional de depuración.

Prueba esto 12-1 Acceder a buzones electrónicos POP3 con PEAR

Para demostrar la utilidad de PEAR en la vida real, hagamos una aplicación sencilla: un lector de buzones electrónicos, que acepte la identificación del usuario y lo conecte con su servidor de correo electrónico para recuperar los datos de su buzón. Para hacerlo en PHP, un desarrollador necesitaría recompilar PHP con soporte para las extensiones IMAP o programar una conexión que se comunicara directamente con el servidor de correo electrónico, enviar manualmente una solicitud e interpretar las respuestas. La primera opción consume demasiado tiempo (y por lo regular no es práctica en servidores compartidos o de producción), mientras que la segunda ocupa mucho tiempo y recursos del sistema.

Sin embargo, PEAR ofrece una tercera opción más sencilla: el paquete Net_POP3 en http://pear.php.net/package/Net_POP3, que proporciona una API ya hecha para conectarse a un servidor de correo que cumpla con POP3 e interactuar con él. El uso de este paquete ofrece dos importantes beneficios: en primer lugar, puede aplicarse de inmediato a un script PHP, sin necesidad de hacer cambios en el servidor; y en segundo lugar, proporciona una API de alto nivel que ya tiene integrada toda la funcionalidad requerida, incluido el manejo de errores, con lo que se reduce de manera importante el total del tiempo de desarrollo requerido para el proyecto.

He aquí un ejemplo del código (*pop3.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 12-1: Accesar buzones electrónicos POP3 con PEAR</title>
  </head>
  <body>
    <h2>Proyecto 12-1: Accesar buzones electrónicos POP3 con PEAR</h2>
    <?php
    // si el formulario ya ha sido enviado
    if (isset($_POST['submit'])) {

      // crea excepciones
      class InputException extends Exception {}
      class ConnException extends Exception {}

      // obtiene los datos de entrada del formulario
      $host = $_POST['host'];
```

www.compumusicmania.com

```
$puerto = $_POST['puerto'];
$usuario = $_POST['usuario'];
$clave = $_POST['clave'];

try {
// valida los datos de entrada del formulario
if (empty($host)) {
  throw new InputException('Nombre del host perdido');
}
if (empty($puerto)) {
  throw new InputException('Puerto perdido');
}
if (empty($usuario)) {
  throw new InputException('Nombre de usuario perdido');
}
if (empty($clave)) {
  throw new InputException('Contraseña perdida');
}

// crea objeto
require_once 'Net/POP3.php';
$pop3 =& new Net_POP3();

// establece conexión con el host
if (PEAR::isError($ret = $pop3->connect($host, $puerto))) {
  throw new ConnException($ret->getMessage());
}

// inicio de sesión
if (PEAR::isError($ret = $pop3->login($usuario, $clave, 'USER'))) {
  throw new ConnException($ret->getMessage());
}

// obtiene la cantidad de mensajes y el tamaño del buzón electrónico
echo $pop3->numMsg() . ' mensajes en el buzón, ' . $pop3->getSize()
. 'bytes <p/>';

// obtiene encabezados de los mensajes más recientes
if ($pop3->numMsg() > 0) {
  $msgData = $pop3->getParsedHeaders($pop3->numMsg());
  echo 'Mensajes más recientes de ' . htmlentities($msgData['From']
) . ',subject\'' , htmlentities($msgData['Subject']) . '\'';
}

// desconecta
$pop3->disconnect();

} catch (InputException $e) {
  die ('Error en la validación de datos de entrada: ' . $e->getMessage());
} catch (ConnException $e) {
  die ('Error en la conexión: El servidor dice' . $e->getMessage());
} catch (Exception $e) {
  die ('ERROR: ' . $e->getMessage());
```

(*continúa*)

```
    }
  } else {
  ?>
  <form method="post" action="pop3.php">
    Nombre del servidor: <br />
    <input type="text" size="20" name="host" />
    <p>
    Puerto del servidor: <br />
    <input type="text" size="4" name="puerto" value="110" />
    <p>
    Nombre de usuario: <br />
    <input type="text" size="20" name="usuario" />
    <p>
    Contraseña: <br />
    <input type="password" size="10" name="clave" />
    <p>
    <input type="submit" name="submit" value="Enviar" />
  </form>
  <?php
  }
  ?>
 </body>
</html>
```

Este script comienza generando un sencillo formulario Web para ingresar la información sobre el servidor de correo electrónico del usuario (figura 12-2). Una vez que el formulario

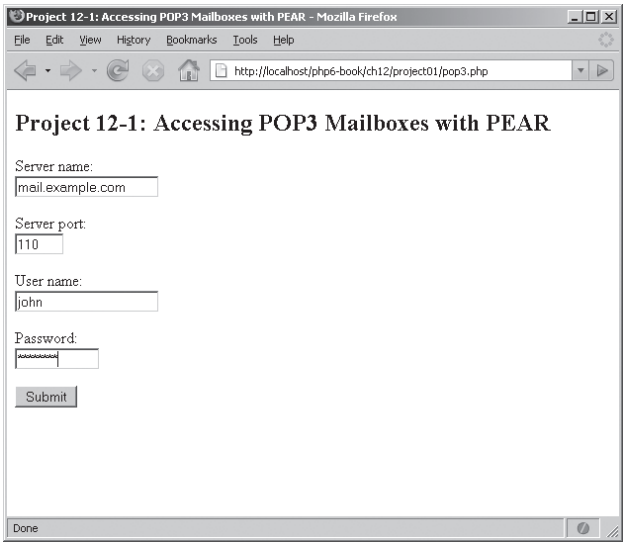


Figura 12-2 Formulario Web para ingresar los datos del servidor de correo electrónico

www.compumusicmania.com

es enviado, los datos ingresados en él son validados y en caso de que cualquiera de ellos no sea válido, se lanzan las respectivas excepciones. A continuación se carga la clase Net_POP3 y se inicializa una instancia de la misma. La clase expone el método `connect()`; cuando pasa el nombre del servidor POP3 y el puerto como argumentos, este método intenta abrir una conexión al servidor POP3 especificado. Cuando se establece la conexión, el método `login()` de la misma clase es utilizado para ingresar al servidor y acceder al buzón electrónico del usuario, utilizando los datos de nombre de usuario y contraseña proporcionados por el formulario. Después de ingresar con éxito, se dispone de cierta cantidad de métodos útiles para interactuar con el buzón, por ejemplo: el método `numMsg()` regresa la cantidad de mensajes existentes en el buzón, mientras que el método `getSize()` regresa el tamaño del buzón en bytes. Cuando es posible, también se recuperan el tema y el remitente del mensaje más reciente utilizando el método `getParsedHeaders()`, que regresa una matriz con los encabezados del mensaje. Una vez que esta información ha sido recolectada y presentada, el método `disconnect()` se utiliza para dar por terminada la conexión con el servidor.

La figura 12-3 muestra el resultado de una conexión correcta.

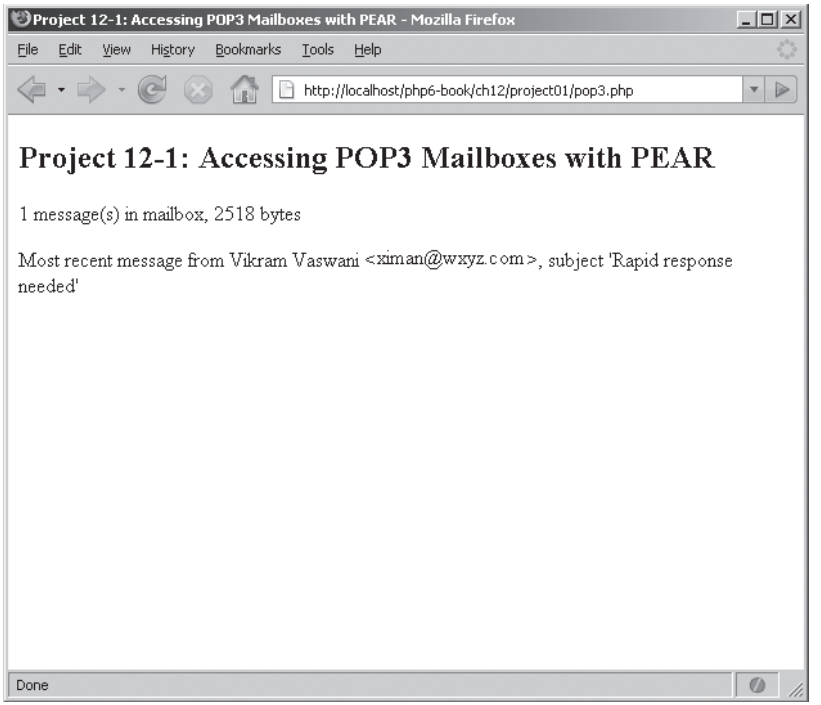


Figura 12-3 Información recuperada de un servidor POP3 con Net_POP3

www.compumusicamania.com

Utilizar PECL

PECL es la biblioteca comunitaria de extensiones PHP, disponible en el sitio Web http://pecl.php.net/. PECL busca expandir las capacidades de PHP a través de módulos de lenguaje de bajo nivel, escritos en lenguaje de programación C; casi todos ellos deben integrarse directamente en el motor PHP (por lo general compilándolos). Las extensiones PECL se distribuyen como archivos comprimidos TAR y pueden instalarse en el sistema de desarrollo ya sea a través de un proceso manual de compilación e instalación o con el instalador PECL (incluido con cada distribución de PHP).

Instalar extensiones PECL

El procedimiento para instalar las extensiones PECL es diferente en Windows y UNIX. Para bajar, compilar e instalar extensiones en UNIX, todo en uno, se ejecuta el siguiente comando en la consola:

```
shell> pecl install nombre-de-la-extensión
```

El instalador PECL se conectará ahora con el servidor PECL, bajará el código fuente, lo compilará y lo instalará en la ubicación apropiada de tu sistema. La figura 12-4 muestra un ejemplo de instalación de las extensiones Zip, disponibles en http://pecl.php.net/package/zip.

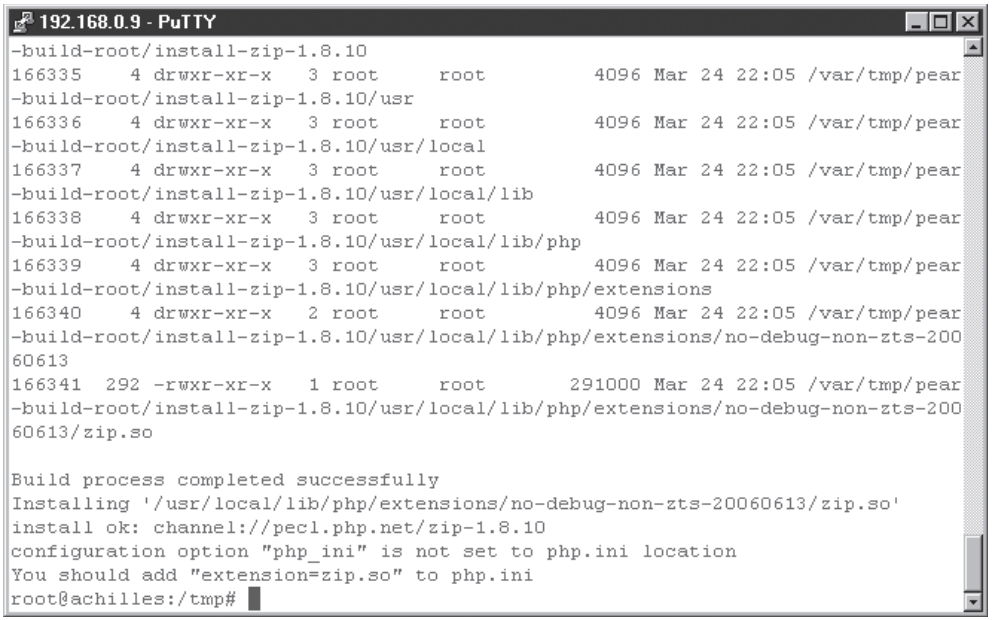


Figura 12-4 Instalación de las extensiones PECL en UNIX

www.compumusicmania.com

Como opción, es posible bajar el código fuente y compilarlo manualmente en un módulo PHP:

```
shell# cd zip-1.8.10
shell# phpize
shell# ./configure
shell# make
shell# make install
```

Este proceso debe generar un módulo PHP cargable llamado *zip.so* y copiarlo en el directorio de extensiones PHP. Ahora debe habilitarse en el archivo de configuración *php.ini*.

Para los usuarios de Windows es más sencillo: sólo necesitan descargar una extensión PECL precompilada, copiarla al directorio de extensiones PHP y luego activar la extensión en el archivo de configuración *php.ini*. Las extensiones PECL precompiladas para Windows están disponibles en http://pecl4win.php.net/.

Una vez que las extensiones han sido correctamente instaladas y activadas, reinicia el servidor Web y verifica los datos de salida del comando `phpinfo()`. Si las extensiones han sido instaladas correctamente, `phpinfo()` las mostrará como se aprecia en la figura 12-5.

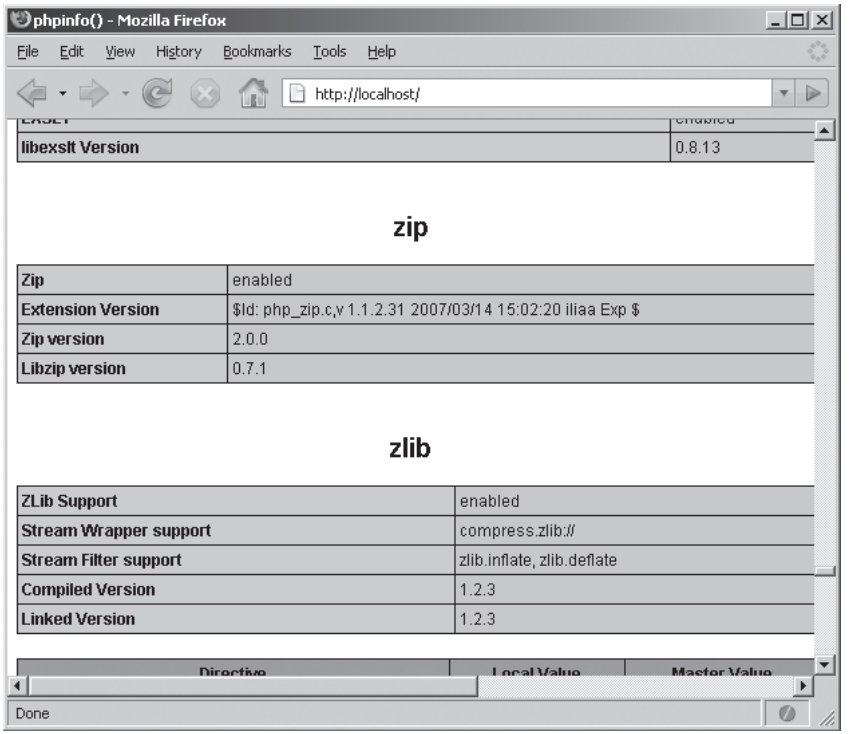


Figura 12-5 Los datos de salida del comando phpinfo(), mostrando las extensiones PECL

www.compumusicmania.com

TIP

Puedes encontrar más información sobre la instalación de las extensiones PECL en el manual de PHP, en www.php.net/manual/install.pecl.php

Prueba esto 12-2 Crear archivos Zip con PECL

Pongamos otro ejemplo: crear un archivo comprimido Zip. Esto no es algo que normalmente puedas hacer con PHP, porque la construcción original de PHP no incluye soporte para archivos Zip. PECL viene al rescate con su extensión Zip, que proporciona una API completa para leer y escribir archivos Zip.

He aquí el ejemplo en acción (*zip.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 12-2: Crear archivos Zip con PECL</title>
  </head>
  <body>
    <h2>Proyecto 12-2: Crear archivos Zip con PECL</h2>
    <?php
      // incrementa el tiempo de ejecución del script
      ini_set('max_execution_time', 300);

      // crea un objeto
      $zip = new ZipArchive();

      // abre archivo
      if ($zip->open('mi-archivo.zip', ZIPARCHIVE::CREATE) !== TRUE) {
          die ("ERROR: No fue posible abrir archivo.");
      }

      // inicializa un reiterador
      // pasa el directorio que debe ser procesado
      $iterator = new RecursiveIteratorIterator(new RecursiveDirectory
Iterator("app/"));

      // hace reiteraciones sobre el directorio
      // añade cada archivo encontrado a archive
      foreach($iterator as $key->$value) {
        $zip->addFile(realpath($key), $key) or die ("ERROR: No fue
posible añadir archivo: $key");
        echo "Añadiendo archivo $key…<br />";
      }

      // cierra y guarda archivo
```

www.compumusicmania.com

```
      $zip->close();
      echo "Archivo creado con éxito.";
    ?>
  </body>
</html>
```

Este código muestra cómo se puede utilizar la extensión Zip de PECL para añadir soporte Zip a PHP. El script comienza creando una instancia del objeto ZipArchive; este objeto sirve de punto de entrada para todas las funciones de la extensión Zip. A continuación, el método `open()` del mismo objeto se utiliza para crear un nuevo archivo, y el método `addFile()` se usa después, en combinación con un RecursiveDirectoryIterator, para iterar sobre el directorio *app/* y sus directorios secundarios, añadiendo todos los que encuentra al archivo Zip. Una vez que todos los archivos se han añadido, el método `close()` del mismo objeto se encarga de realizar la compresión y escribir el archivo final en disco.

La figura 12-6 muestra el resultado.

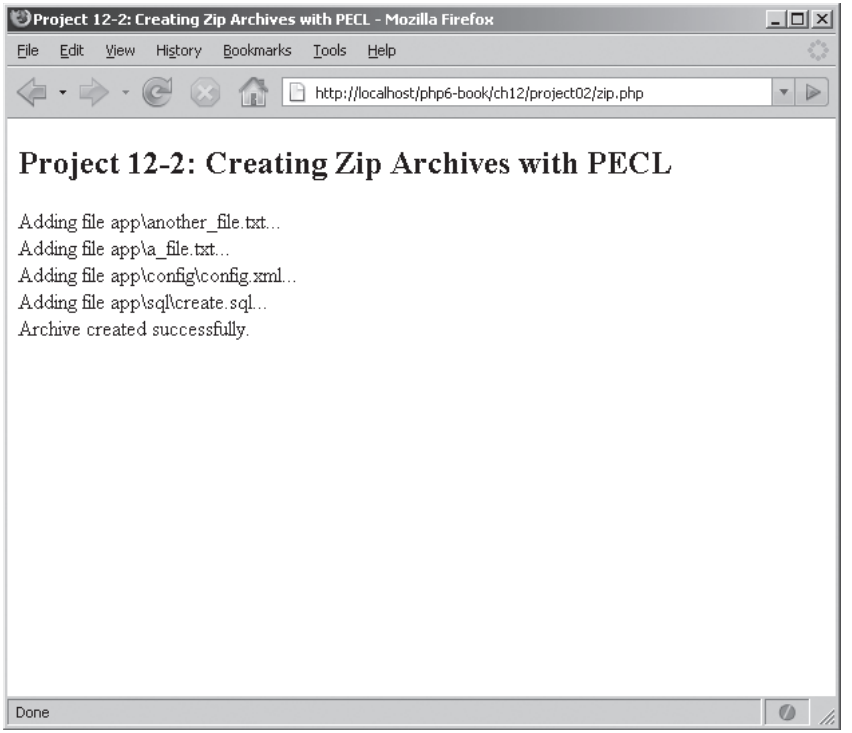


Figura 12-6 Creación dinámica de un archivo Zip con PHP

www.compumusicmania.com

Resumen

Aunque ya has llegado al capítulo final de este libro, debe quedarte claro que tu viaje por PHP está muy lejos de haber concluido..., esto es sólo el principio. Tanto PEAR como PECL son depósitos muy populares que albergan programas auxiliares de alta calidad y estables para PHP y su contenido crece constantemente. Este capítulo mostró sólo dos de los cientos de programas auxiliares disponibles: una biblioteca cliente POP3 y un módulo para archivos comprimidos en formato Zip.

Si alguna vez te encuentras con algún problema de desarrollo difícil, ocupa unos momentos buscando en estos depósitos; existe una gran probabilidad de que encuentres la solución ya hecha en alguno de ellos, lo que te ahorrará muchas horas de trabajo. Mientras tanto, los siguientes vínculos te ayudarán a aprender más sobre los temas tratados en este capítulo:

- El sitio Web PEAR, en **http://pear.php.net**
- El sitio Web PECL, en **http://pecl.php.net**
- Notas de instalación de PEAR, en **http://pear.php.net/manual/en/installation.php**
- Notas de instalación de PECL, en **http://php.net/manual/install.pecl.php**
- Documentación para el paquete Net_POP3, en **http://pear.php.net/manual/en/package.networking.net-pop3.php**
- Documentación para las extensiones PECL Zip, en **www.php.net/zip**

Has llegado al final del libro. Espero que lo hayas disfrutado y encontrado útil, y que ahora tengas las bases necesarias para salir y crear tus propias aplicaciones PHP de alta calidad. ¡Buena suerte y feliz codificación!

Autoexamen Capítulo 12

1. ¿Cuál es la diferencia entre PEAR y PECL?
2. Describe los pasos para instalar una extensión PECL en el ambiente de desarrollo Windows.
3. Revisa la documentación utilizada para el paquete Net_POP3 utilizado en este capítulo y reescribe el código para que muestre el contenido completo del mensaje más reciente.
4. Descarga e instala la extensión ID3 de PECL para trabajar con archivos MP3, y luego escribe una aplicación PHP para procesar un directorio de archivos MP3, imprimiendo el título de la canción y el artista que la interpreta. (Pista: el paquete PECL está disponible en http:// pecl.php.net/package/id3, y la documentación está disponible en www.php.net/id3.)

www.compumusicmania.com

Parte IV

Apéndices

www.compumusicmania.com

Apéndice A

Instalar y configurar los programas requeridos

www.compumusicmania.com

Habilidades y conceptos clave

- Aprender a obtener e instalar el software MySQL, SQLite, PHP y Apache de Internet
- Realizar pruebas básicas para asegurar que la aplicación funcione correctamente
- Descubrir la manera de activar automáticamente todos los componentes requeridos cuando arranque el sistema
- Dar los pasos básicos para salvaguardar la seguridad de tu instalación de MySQL

En este libro has aprendido sobre el lenguaje de programación PHP y cómo puede ser utilizado para construir aplicaciones Web sofisticadas. Algunos de los ejemplos de este libro también incluyen el uso de PHP con componentes de terceros, como XML, MySQL y SQLite. Este apéndice te enseña a instalar y configurar estos componentes en tu estación de trabajo, además de la manera de crear un ambiente de desarrollo que puede ser utilizado para ejecutar el código presentado en este libro.

PRECAUCIÓN

La intención de este apéndice es proporcionar un panorama general sobre el proceso de instalación y configuración de MySQL, SQLite, PHP y Apache bajo UNIX y Windows. *No intenta ser* un sustituto para la documentación de instalación que acompaña cada uno de los paquetes mencionados. Si encuentras dificultades para instalar los paquetes que aquí se mencionan, visita su respectivo sitio Web o busca información en Web sobre el manejo de errores y su posible corrección (algunos sitios se mencionan al final de este apéndice).

Obtener el software

El primer paso consiste en asegurar que tienes todos los programas necesarios. He aquí la lista:

- **PHP** PHP proporciona un conjunto de herramientas para desarrollo de aplicaciones para Web y de consola. Puede ser descargado de www.php.net/. En el mismo sitio encontrarás la versión en código fuente como la binaria para las plataformas Windows, UNIX y Mac OS X. Los usuarios de UNIX deben descargar la versión en código fuente más reciente, mientras que los de Windows deben descargar la última versión en formato binario. En el momento en que este libro se imprimió, la versión más reciente de PHP era la 5.3.0 alpha 1.

www.compumusicmania.com

- **Apache** Apache es un servidor Web rico en características que trabaja bien con PHP. Puede ser descargado gratuitamente en http://httpd.apache.org/ como código fuente y en formato binario, para diferentes plataformas. Los usuarios de UNIX deben descargar el código fuente más reciente, mientras que los de Windows deben descargar el instalador binario adecuado para la versión de Windows que manejen. En el momento en que este libro se imprimió, la versión más reciente del servidor Apache era la 2.2.9.
- **MySQL** El servidor de base de datos MySQL es un sistema para almacenamiento y recuperación de datos robusto y escalable. Está disponible en código fuente y versión binaria en www.mysql.com/. Las versiones binarias están disponibles para Linux, Solaris, FreeBSD, Mac OS X, Windows, HP-UX, IBM AIX, SCO OpenUNIX y SGI Irix, mientras que el código está disponible para las plataformas Windows y UNIX. La versión binaria es la más recomendable por dos razones: es más fácil de instalar y el equipo de desarrolladores de MySQL la ha optimizado para las diferentes plataformas. En el momento en que este libro se imprimió, la versión más reciente de la base de datos MySQL era la 5.0.67.
- **SQLite** SQLite es una base de datos independiente significativamente más pequeña que MySQL. Está disponible tanto en código fuente como en formato binario en www.sqlite.org/, para las plataformas Windows, UNIX y Mac OS X. Los usuarios de Windows y UNIX deben descargar la versión binaria (una liga hacia la descarga está disponible en la página Web compañera de este libro). En el momento en que este libro se imprimió, la versión más reciente de la base de datos SQLite era la 3.6.1. Sin embargo, como SQLite 3.x soporta PHP 5.3, todavía es una versión experimental en el momento de la publicación de este libro, por lo que en los ejemplos se utilizó SQLite 2.x.

Además de estos cuatro componentes básicos, los usuarios de UNIX pueden requerir algunas bibliotecas de soporte. He aquí la lista:

- La biblioteca `libxml2`, disponible en www.xmlsoft.org/
- La biblioteca `zlib`, disponible en www.gzip.org/zlib/

Por último, los usuarios de ambas plataformas necesitarán una herramienta de descompresión capaz de manejar archivos TAR (archivos Tape) y GZ (GNU Zip). En UNIX, las utilidades *tar* y *gzip* son apropiadas y suelen incluirse en el sistema operativo. En Windows, una buena herramienta para descomprimir es WinZip, disponible en www.winzip.com/.

NOTA

Los ejemplos de este libro han sido desarrollados y probados en SQLite 2.8.17, MySQL 5.0.67, con Apache 2.2.9 y PHP 5.2.5 y 5.3.0 alpha1.

www.compumusicmania.com

Instalar y configurar los programas

Una vez que se han obtenido los programas necesarios, el paso siguiente consiste en instalar las diferentes piezas y hacer que se comuniquen entre sí. La siguiente sección delinea los pasos a seguir para las plataformas Windows y UNIX.

NOTA

Si estás utilizando una computadora Apple, encontrarás las instrucciones para instalar PHP en Mac OS X en el manual PHP: www.php.net/manual/en/install.macosx.php

Instalar en UNIX

El proceso de instalación en UNIX incluye diferentes pasos: instalar MySQL a partir del formato binario; compilar e instalar PHP del código fuente, y compilar y configurar Apache para que maneje correctamente las solicitudes de páginas Web PHP. Estos pasos son descritos con gran detalle en las siguientes subsecciones.

Instalar MySQL

Para instalar MySQL a partir del formato binario, sigue estos pasos:

1. Asegúrate de haber ingresado al sistema como usuario "root".

```
[user@host]# su - root
```

2. Extrae el contenido del archivo binario de MySQL en el directorio apropiado de tu sistema, por ejemplo: */usr/local/*.

```
[root@host]# cd/usr/local
[root@host]# tar -xzvf /tmp/mysql-5.0.67-linuxi686.tar.gz
```

   Los archivos de MySQL deben extraerse en un directorio cuyo nombre esté de acuerdo con el formato *mysql-versión-so-arquitectura*, por ejemplo: *mysql-5.0.67-linux-i686*.

3. Para facilitar el uso, establece un nombre corto para el directorio creado en el paso anterior, creando un vínculo suave llamado *mysql* que apunte a este directorio en la misma ubicación.

```
[root@host]# ln -s mysql-5.0.67-linux-i686 mysql
```

4. Por razones de seguridad, nunca debe ejecutarse el servidor de base de datos MySQL como superusuario. Por lo tanto, es necesario crear un usuario especial "mysql" y un grupo para este propósito. Realiza esta tarea con los comandos `groupadd` y `useradd`, y luego cambia la propiedad del directorio de instalación de MySQL a los usuarios y grupos recién creados:

```
[root@host]# groupadd mysql
[root@host]# useradd -g mysql mysql
[root@host]# chown -R mysql /usr/local/mysql
[root@host]# chgprp -R mysql /usr/local/mysql
```

www.compumusicmania.com

5. Inicializa las tablas de MySQL con el script de inicialización *mysql_install_db* que se incluye con el paquete de distribución.

```
[root@host]# /usr/local/mysql/scripts/mysql_install_db --user=mysql
```

La figura A-1 muestra lo que verás cuando aplicas el comando.

Como lo sugieren los datos de salida anteriores, este script de inicialización prepara e instala las diferentes tablas de la base de datos MySQL, y también establece los permisos por defecto para el acceso a la base.

6. Altera la propiedad de los binarios de MySQL para que el propietario sea "root":

```
[root@host]# chown -R root /usr/local/mysql
```

y asegúrate de que el usuario "mysql" creado en el paso 4 tenga permisos de lectura y escritura al directorio de datos MySQL.

```
[root@host]# chown -R mysql /usr/local/mysql/data
```

7. Arranca el servidor MySQL ejecutando manualmente el script *mysqld_safe*.

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

Ahora MySQL debe iniciar de manera normal.

Una vez que la instalación se ha completado correctamente y el servidor ha arrancado, pasa a la sección "Probar MySQL" para verificar que funciona como debe.

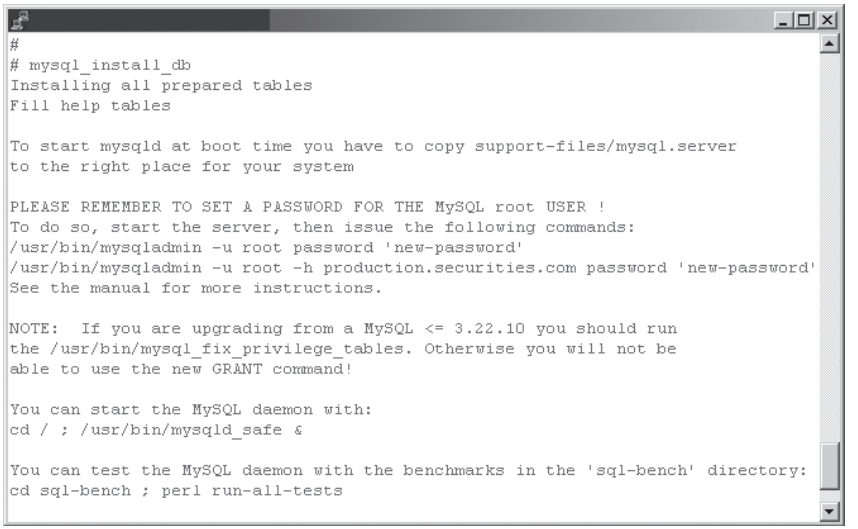


Figura A-1 Los datos de salida del script *mysql_install_db*

www.compumusicmania.com

Instalar Apache y PHP

PHP puede integrarse con el servidor Web Apache de dos maneras: como un módulo dinámico cargado dentro del servidor Web en tiempo de ejecución, o como un módulo estático que está integrado al código fuente de Apache en tiempo de construcción. Cada opción tiene ventajas y desventajas:

- Instalar PHP como un módulo dinámico facilita la actualización del motor PHP más adelante, porque sólo necesita volver a compilar el módulo PHP y no el resto del servidor Web Apache. Por otra parte, con un módulo cargado dinámicamente, el rendimiento tiende a ser más lento en comparación con un módulo estático, que está más integrado al servidor.
- Instalar PHP como un módulo estático mejora el rendimiento, porque dicho módulo está compilado directamente en el código fuente del servidor Web. Sin embargo, esta integración cercana tiene una importante desventaja: si decides actualizar el motor PHP, necesitarás reintegrar el nuevo módulo PHP al código fuente de Apache y volver a compilar el servidor Web.

Esta sección muestra cómo compilar PHP como módulo dinámico que se carga en el servidor Apache en tiempo de ejecución.

1. Asegúrate de haber ingresado al sistema como usuario "root".

```
[user@host]# su - root
```

2. Extrae el contenido del archivo fuente de Apache en el directorio temporal de tu sistema.

```
[root@host]# cd/tmp
[root@host]# tar -xzvf /tmp/httpd-2.2.9.tar.gz
```

3. Para permitir que PHP se cargue dinámicamente, el servidor Apache debe ser compilado con soporte para compartir objetos dinámicamente (DSO). Este soporte se activa con la opción `--enable-so`, transmitida al script de configuración *configure* del servidor Apache, como se muestra a continuación:

```
[root@host]# cd /tmp/httpd-2.2.9
[root@host]# ./configure --prefix=/usr/local/apache --enable-so
```

Aparecerán algunas pantallas con datos de salida (la figura A-2 muestra un ejemplo), mientras que el script *configure* establece las variables necesarias para el proceso de compilación.

4. Ahora, compila el servidor utilizando `make`, e instálalo en tu sistema utilizando `make install`.

```
[root@host]# make
[root@host]# make install
```

La figura A-3 muestra lo que probablemente verás durante el proceso de compilación. Ahora, Apache ya debe estar instalado en */usr/local/apache/*.

www.compumusicmania.com

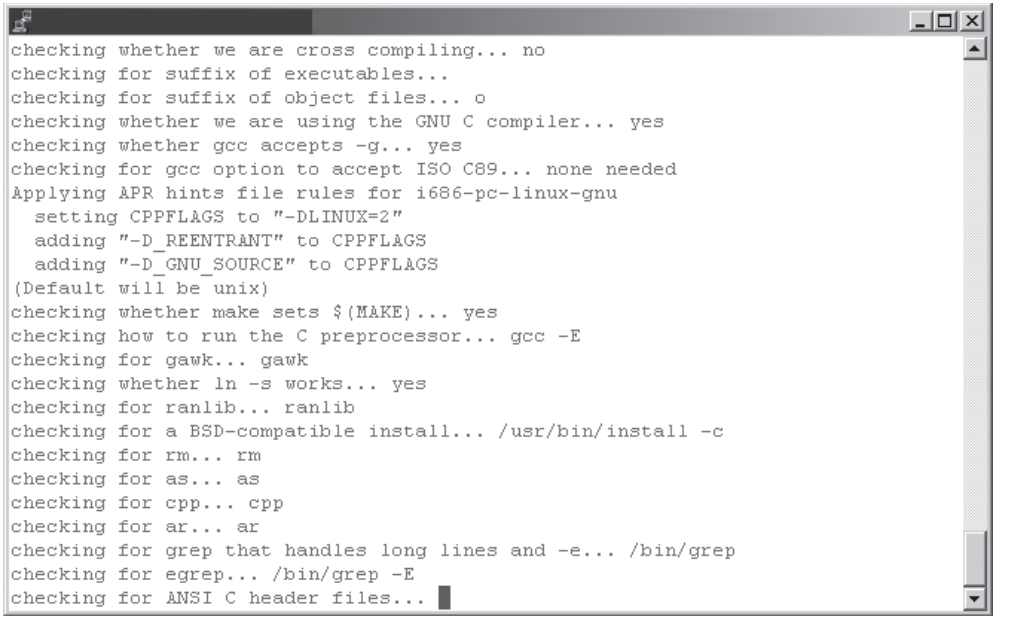


Figura A-2 Configuración del árbol fuente Apache

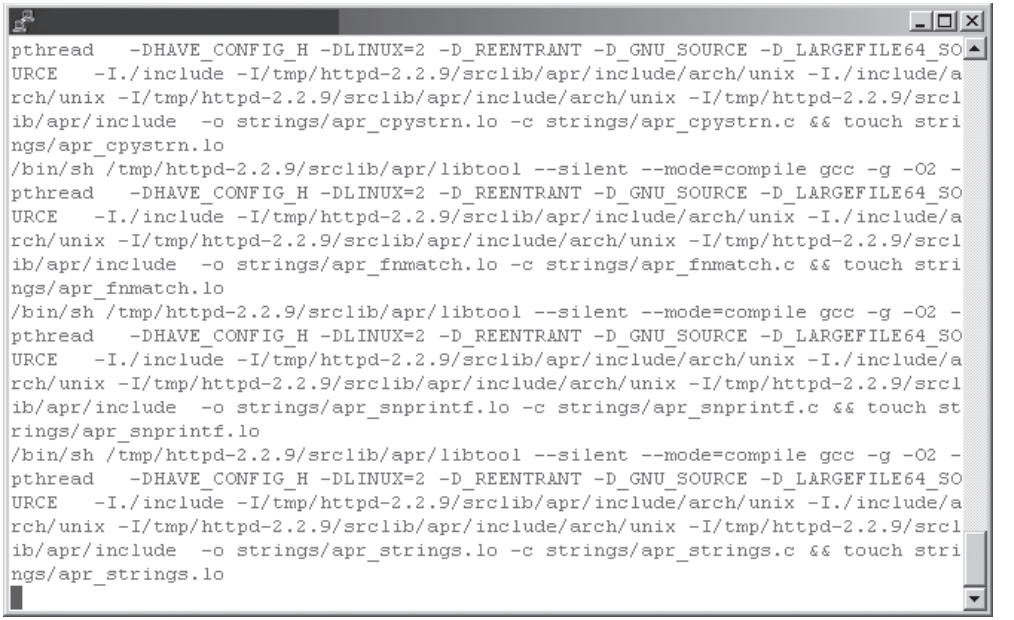


Figura A-3 Compilación de Apache

www.computeciencia.com

5. Ahora compila e instala PHP. Comienza por extraer el contenido del archivo fuente de PHP en el directorio temporal de tu sistema.

```
[root@host]# cd /tmp
[root@host]# tar -xzvf /tmp/php-5.3.0.tar.gz
```

6. Este paso es el más importante en el proceso de instalación de PHP. Implica enviar argumentos al script *configure* para configurar el módulo PHP. Estos parámetros en líneas de comando especifican las extensiones PHP que serán activadas, y también le indican a PHP dónde encontrar las bibliotecas de soporte necesarias para esas extensiones.

```
[root@host]# cd /tmp/php-5.3.0
[root@host]# ./configure -prefix=/usr/local/php --with-apx2=/usr/
local/apache/bin/apxs --with-zlib --with-mysqli=mysqlnd --with-pdo-
mysql=mysqlnd
```

He aquí una breve explicación de lo que hace cada uno de estos argumentos:

- El argumento `--with-apxs2` le indica a PHP dónde encontrar el script APXS (APache eXtenSion) de Apache. Este script simplifica la tarea de construir e instalar los módulos descargables de Apache.
- El argumento `--with-zlib` le indica a PHP que active las características de compresión (Zip), que son utilizadas por diferentes servicios PHP.
- El argumento `--with-mysqli` activa la extensión PHP MySQLi y le indica a PHP que utilice el Controlador Nativo MySQL (mysqlnd).
- El argumento `--with-pdo-mysql` activa el controlador MySQL PDO y le indica a PHP que utilice el Controlador Nativo MySQL (mysqlnd).

La figura A-4 muestra lo que verás durante el proceso de configuración.

TIP

El proceso de configuración de PHP es muy complejo, te permite controlar muchos aspectos del comportamiento de PHP. Para ver una lista completa de las opciones disponibles, usa el comando `configure --help`, y visita la página www.php.net/manual/en/configure.php para explicaciones detalladas de lo que hace cada una de estas opciones.

7. A continuación, compila e instala PHP utilizando `make` y `make install`:

```
[root@host]# make
[root@host]# make install
```

La figura A-5 muestra lo que probablemente verás durante el proceso de instalación. Ahora, PHP debe estar instalado en */usr/local/php/*.

www.compumusicmania.com

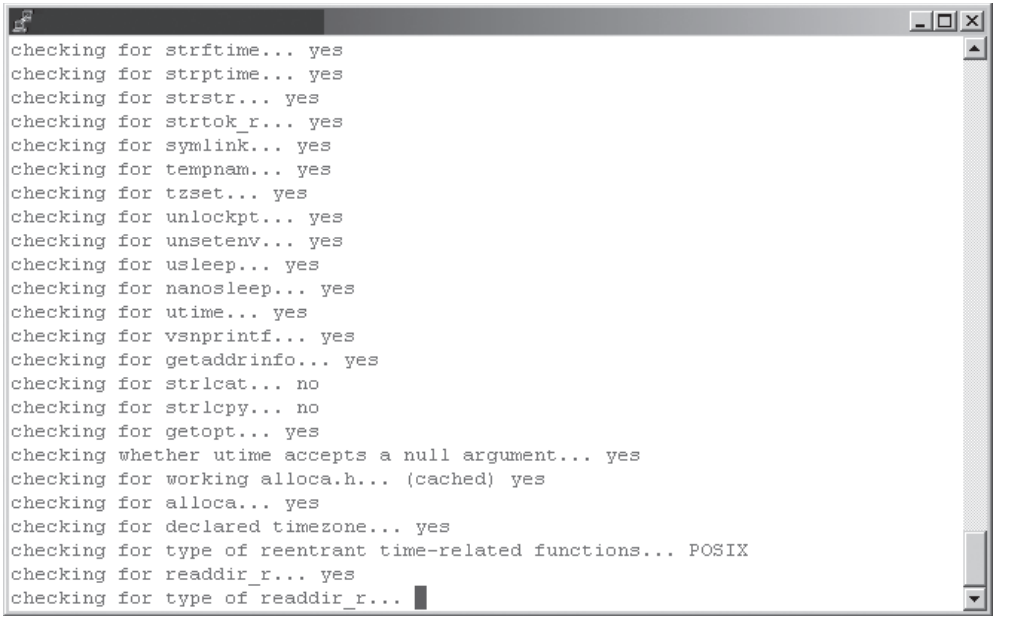


Figura A-4 Configuración del árbol fuente de PHP

Figura A-5 Compilación de PHP

www.compumusicmania.com

8. El paso final en el proceso de instalación consiste en configurar Apache para que reconozca correctamente las solicitudes de páginas PHP. Esto se logra abriendo el archivo de configuración de Apache, *httpd.conf* (puede localizarse en el subdirectorio *conf/* en el directorio de instalación de Apache), con un editor de textos y añadiendo la siguiente línea:

   ```
   AddType application/x-httpd-php .php
   ```

   Guarda los cambios en el archivo. Además, verifica que la siguiente línea aparezca en algún lugar del archivo:

   ```
   LoadModule php5_module libexec/libphp5.so
   ```

9. Arranca el servidor Apache ejecutando manualmente el script *apachectl*.

   ```
   [root@host]# /usr/local/apache/bin/apachectl start
   ```

   Apache debe ejecutarse de manera normal.

Una vez que la instalación se ha completado correctamente y que el servidor ha iniciado, ve a la sección titulada "Probar PHP" para comprobar que todo esté funcionando como es debido.

Pregunta al experto

P: ¿Por qué debo activar manualmente el controlador PDO MySQL, pero no el PDO SQLite en la configuración de PHP?

R: En PHP 5, tanto la extensión SQLite como el controlador PDO SQLite se activan por defecto. Por lo tanto, no hay necesidad de activar manualmente el controlador PDO SQLite durante el proceso de configuración. Sin embargo, todos los demás controladores PDO, incluidos aquellos para MySQL, PostgreSQL y ODBC, requieren activación manual.

Instalar SQLite

Para instalar SQLite a partir de la versión binaria, sigue estos pasos:

1. Asegúrate de haber ingresado al sistema como usuario "root".

   ```
   [user@host]# su - root
   ```

2. Desempaqueta los archivos SQLite binarios en el directorio apropiado de tu sistema (por ejemplo, */usr/local/bin*) y haz el binario ejecutable (figura A-6).

   ```
   [user@host]# cd /usr/local/bin
   [user@host]# gunzip sqlite2-2.8.17.bin.gz
   [user@host]# chmod +x sqlite2-2.8.17.bin
   [user@host]# ln -s sqlite2-2.8.17.bin sqlite
   ```

www.comprumusicamania.com

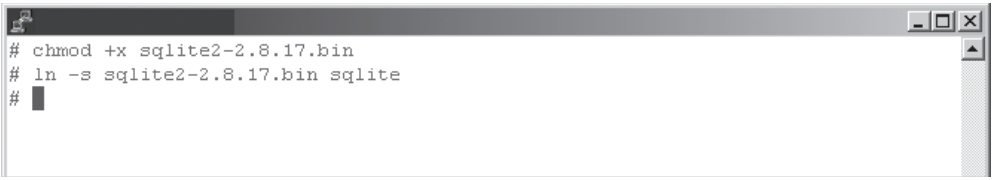


Figura A-6 Instalación de SQLite

SQLite está instalado y listo para utilizarse. Para comprobarlo, ve a la sección titulada "Probar SQLite".

Instalar en Windows

Compilar aplicaciones en Windows es un proceso desafiante, sobre todo para desarrolladores novatos. Con esto en mente, es recomendable que los usuarios de Windows se concentren en instalar y configurar las versiones binarias precompiladas de MySQL, SQLite, PHP y Apache, en vez de intentar compilarlas desde el código fuente. Estas versiones se descargan desde los sitios Web mencionados en la sección anterior y deben instalarse en el orden que se presenta en las siguientes subsecciones.

Instalar MySQL

La versión binaria de MySQL para Windows incluye un instalador automático, que te permite tener funcionando la base de datos en tu sistema en pocos minutos.

1. Ingresa como administrador (si estás utilizando Windows NT/2000/XP/Vista) y desempaqueta el archivo binario en un directorio temporal de tu sistema.
2. Haz doble clic en el archivo *setup.exe* para iniciar el proceso de instalación. Debes ver una pantalla de bienvenida (figura A-7).
3. Selecciona el tipo de instalación requerido (figura A-8).

   Lo más frecuente es que sea una instalación Typical; sin embargo, si no te agradan las opciones por omisión, o si tienes poco espacio en disco, selecciona la opción Custom y decide qué componentes del paquete deben instalarse.
4. MySQL debe empezar a instalarse en tu sistema (figura A-9).
5. Una vez que se complete la instalación, debes ver una notificación. En este punto tendrás la opción de lanzar el asistente de configuración para el servidor MySQL (MySQL Server Instance Config Wizard), para completar la configuración del programa. Selecciona esta opción y tendrás que ver la pantalla de bienvenida correspondiente (figura A-10).

www.compumusicmania.com

Figura A-7 Inicio de la instalación de MySQL en Windows

Figura A-8 Selección del tipo de instalación de MySQL

www.compumusicmania.com

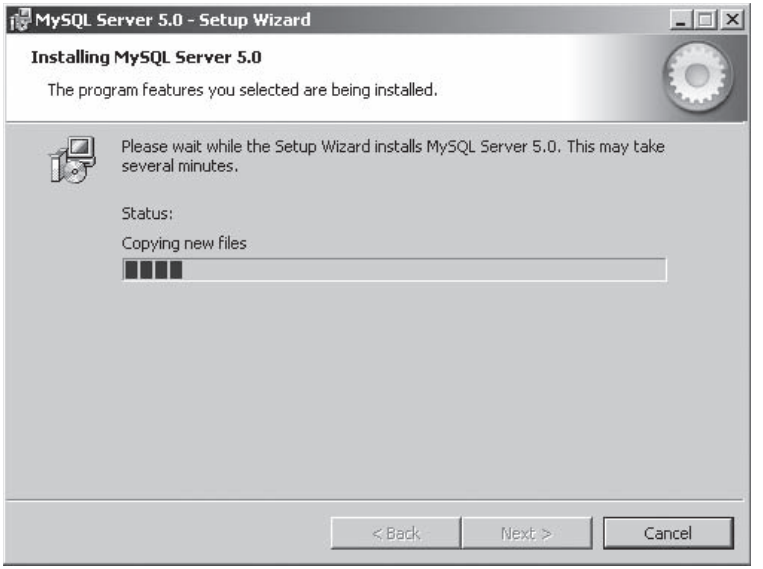


Figura A-9 Progreso de la instalación de MySQL

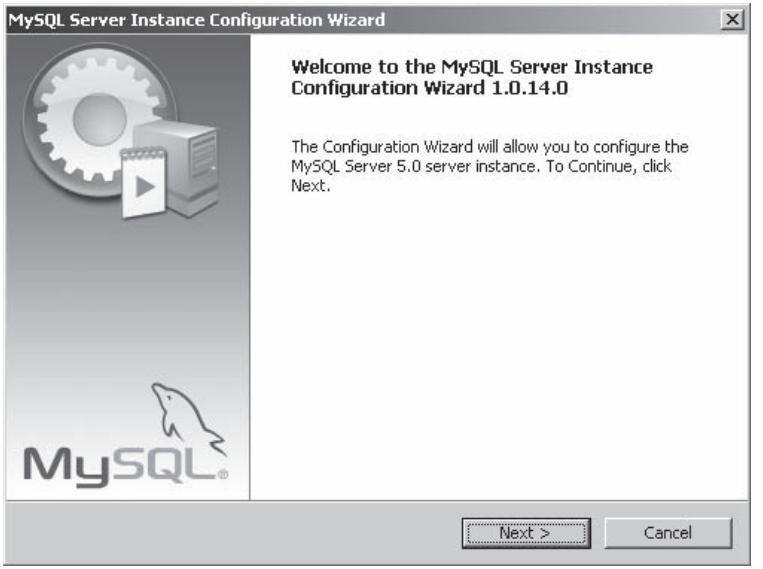


Figura A-10 Inicia la configuración de MySQL en Windows

www.compumusicamania.com

6. Selecciona el tipo de configuración (figura A-11). En casi todos los casos, la configuración estándar (Standard Configuration) será suficiente.
7. Instala MySQL como un servicio de Windows; de esta manera iniciará y se detendrá automáticamente junto con Windows (figura A-12).
8. Ingresa la clave de acceso para la cuenta del administrador de MySQL ("root") (figura A-13).
9. El servidor ya estará configurado con las opciones especificadas e iniciará automáticamente. Se te presentará una notificación de la instalación correcta una vez que se hayan completado todas las tareas requeridas (figura A-14).

Ahora puedes proceder a probar el servidor como se describe en la sección "Probando MySQL", para asegurar que todo trabaja como debe.

Instalar Apache

Una vez que MySQL está instalado, el siguiente paso es instalar el servidor Web Apache. En Windows, se trata de un proceso de colocar el cursor y hacer clic, similar al utilizado en la instalación de MySQL.

1. Haz doble clic en el instalador Apache para comenzar el proceso de instalación. Debes ver una pantalla de bienvenida (figura A-15).

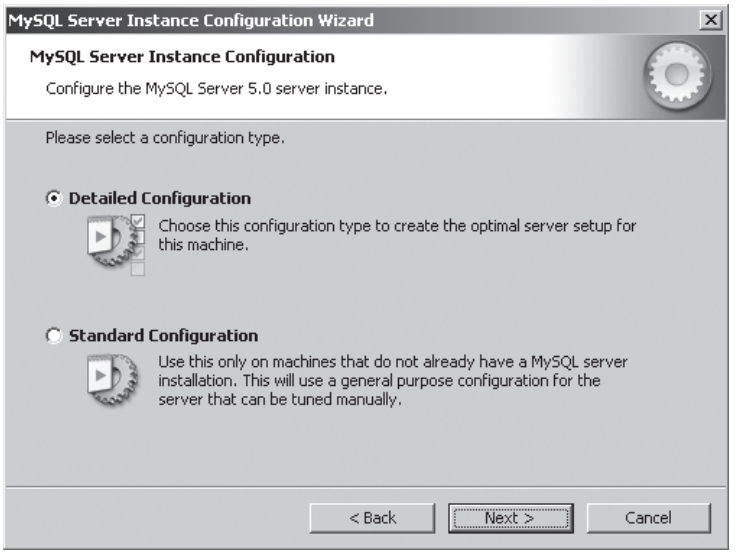


Figura A-11 Seleccionar el tipo de configuración

www.compumusicmania.com

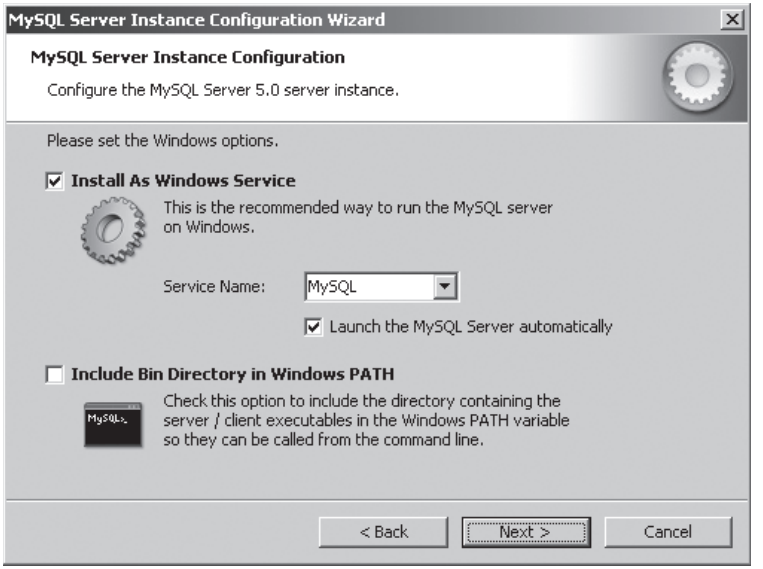


Figura A-12 Establecer el servicio MySQL

Figura A-13 Establecer la contraseña del administrador

www.comunidadmania.com

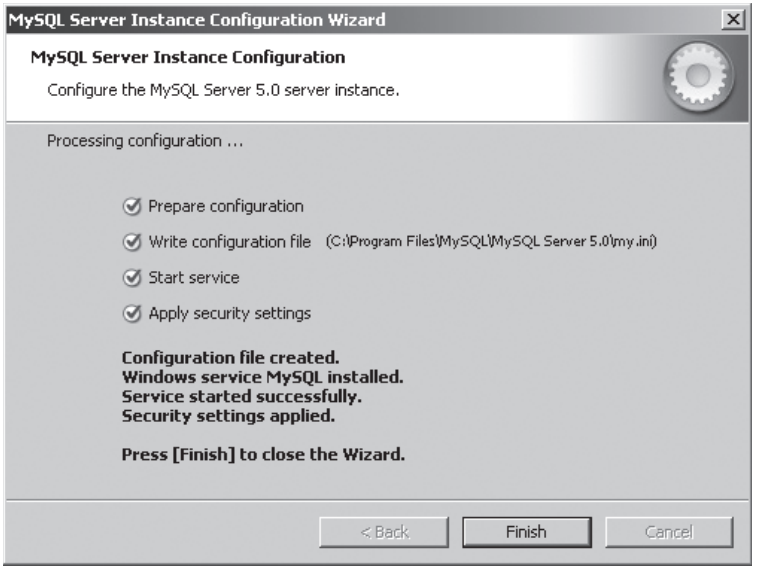


Figura A-14 La configuración de MySQL completada correctamente

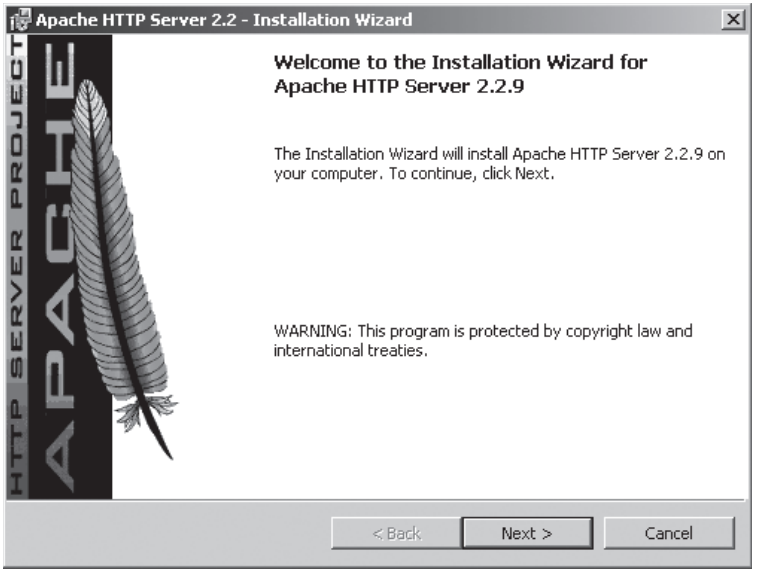


Figura A-15 Principia la instalación de Apache para Windows

www.comunidadmania.com

2. Lee el acuerdo de licencia y acepta los términos para seguir adelante.
3. Lee la información descriptiva y procede a ingresar la información básica sobre el servidor y la dirección de correo electrónico que será mostrada en las páginas de error (figura A-16).
4. Selecciona el tipo de instalación requerida (figura A-17).

   Puedes seleccionar la opción Custom para decidir qué componentes del paquete deben instalarse.
5. Selecciona la ubicación donde debe instalarse Apache, por ejemplo: *c:\archivos de programa\grupo apache* (figura A-18).
6. Apache debe quedar instalado en la ubicación especificada (figura A-19). El proceso de instalación toma algunos minutos para completarse, así que es buena idea que vayas por una taza de café.
7. Una vez que la instalación esté completada, el instalador Apache mostrará una notificación de éxito e iniciará el servidor Web.

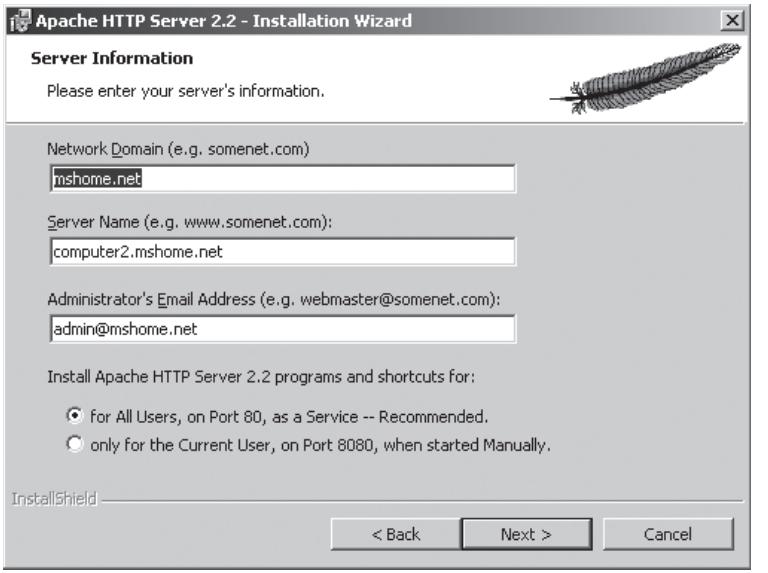


Figura A-16 Ingreso de la información del servidor Apache

www.compumusicmania.com

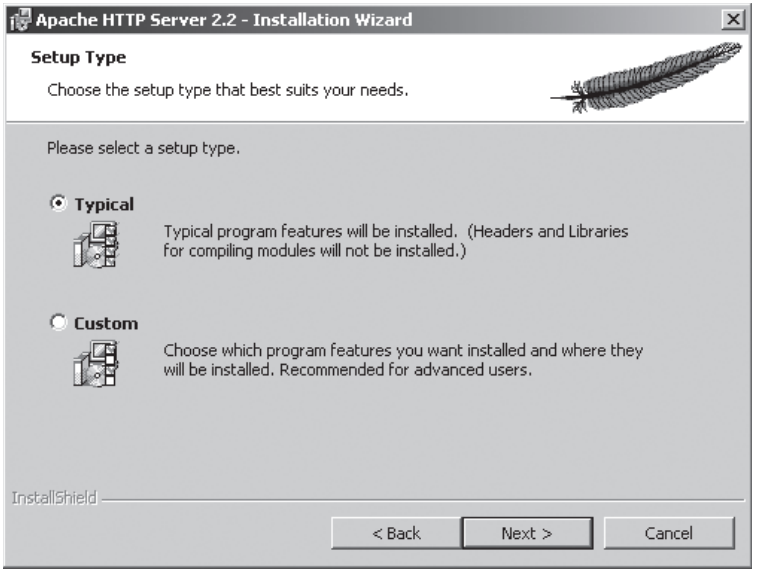


Figura A-17 Selección del tipo de instalación de Apache

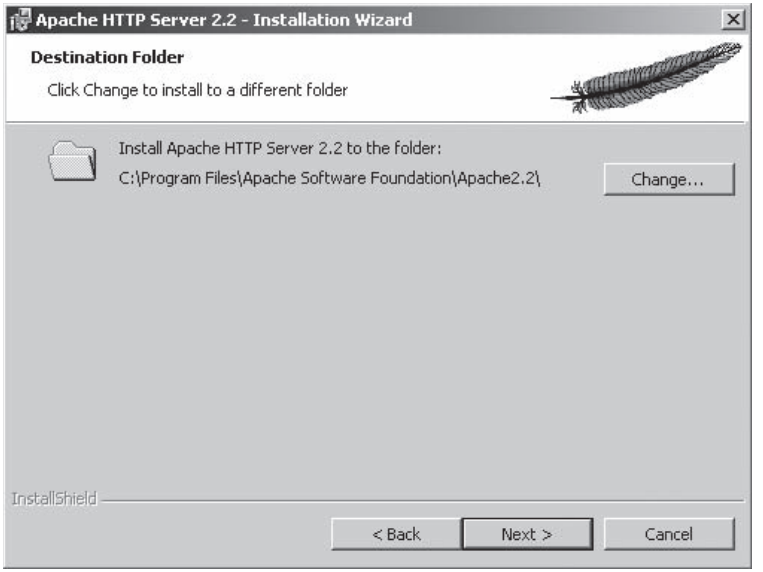


Figura A-18 Selección de la ubicación para instalar Apache

www.compumusicmania.com

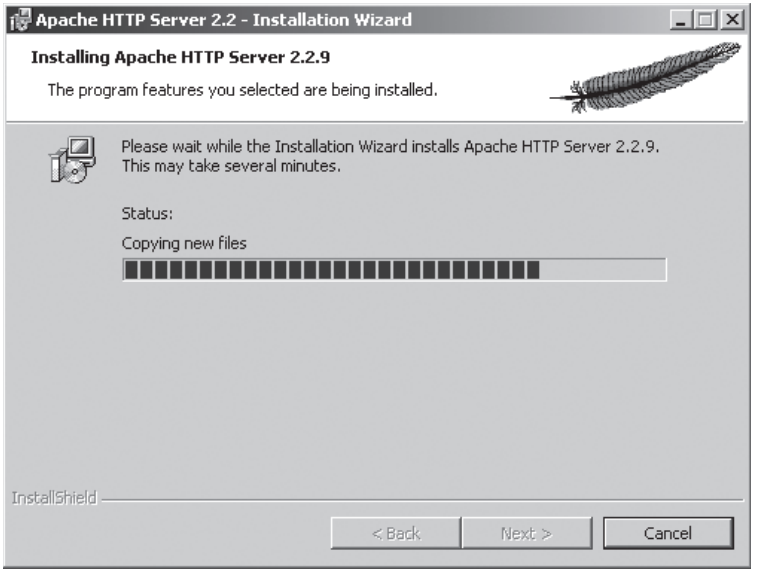


Figura A-19 Instalación de Apache en progreso

Instalar PHP

Existen dos versiones del archivo binario para instalar PHP en Windows: un archivo Zip que contiene todas las extensiones PHP y requiere instalación manual y la versión automática del instalador para Windows que sólo contiene el archivo binario PHP sin extensiones extra. Esta sección describe el proceso de instalación para el archivo Zip PHP.

1. Ingresa como administrador (si estás utilizando Windows NT/2000/XP/Vista) y desempaqueta el archivo binario en un directorio de tu sistema, por ejemplo: *c:\php*. Después de la extracción, este directorio debe parecerse al que se presenta en la figura A-20.
2. A continuación, renombra el archivo *php.ini-recommended* en el directorio de instalación PHP y nómbralo *php.ini*. Este archivo contiene los valores de configuración para PHP, que pueden ser utilizados para modificar la manera en que funciona. Lee los comentarios dentro del archivo para conocer más sobre las configuraciones disponibles.
3. Dentro del archivo *php.ini* localiza la línea

   ```
   extension_dir = "./"
   ```

   y modifícala para que se lea así:

   ```
   extension_dir = "c:\php\ext\"
   ```

www.compumusicmania.com

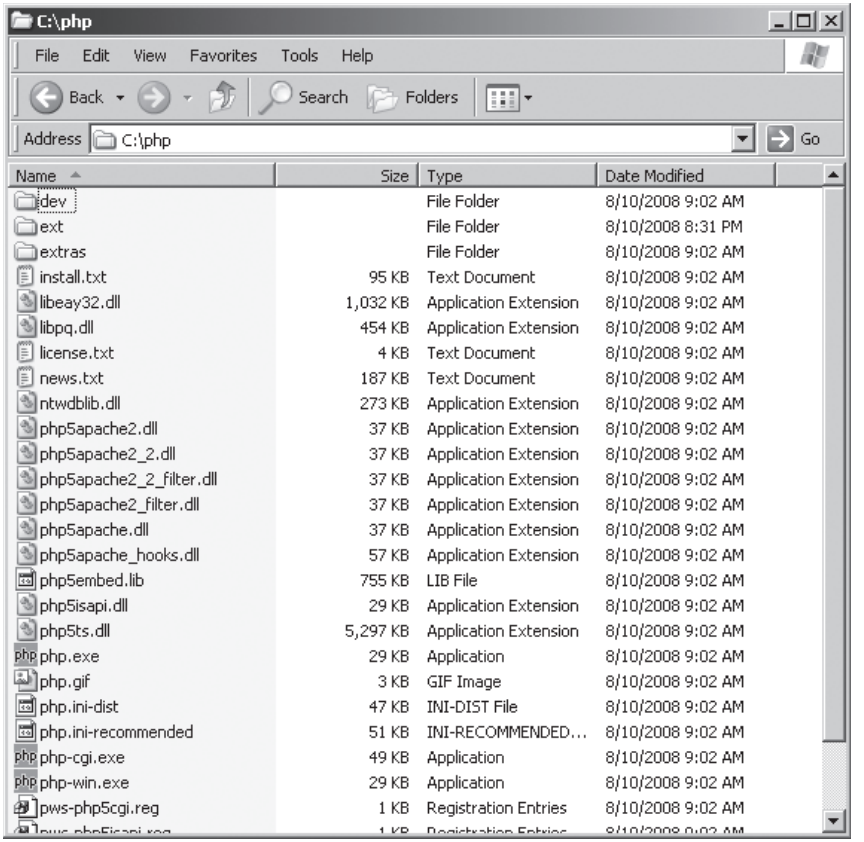


Figura A-20 La estructura de directorio creada al desempaquetar la versión binaria de instalación para Windows

Esto le indica a PHP dónde se localizan las extensiones incluidas en el paquete. Recuerda reemplazar la ruta de acceso "`c:\php\`" con la ubicación de tu instalación PHP.

4. A continuación busca las siguientes líneas y elimina el punto y coma al principio de cada una de ellas (si está presente) de manera que se lean así:

```
extension=php_pdo.dll
extension=php_sqlite.dll
extension=php_mysqli.dll
extension=php_pdo_sqlite.dll
extension=php_pdo_mysqli.dll
```

Esto se encarga de activar las extensiones de PHP para MySQLi, SQLite y PDO.

www.comunidadmania.com

5. Abre el archivo de configuración de Apache, *httpd.conf* (que puede localizarse en el subdirectorio *conf/* del directorio de instalación de Apache) en un editor de textos y añádele las siguientes líneas:

```
AddType application/x-httpd-php .php
LoadModule php5_module "c:\php\php5apache2_2.dll"
SetEnv PHPRC C:\php\
```

Estas líneas le indican a Apache cómo manejar los scripts PHP y dónde encontrar el archivo de configuración *php.ini*. Recuerda reemplazar la ruta de acceso *c:\php* con la ubicación de tu instalación PHP.

6. Cuando el servidor Apache está instalado, se añade de manera automática al menú Inicio. Utiliza este grupo del menú para detener y reiniciar el servidor, como se muestra en la figura A-21.

Ahora PHP está instalado y configurado para funcionar con Apache. Para comprobarlo, ve a la sección "Probar PHP".

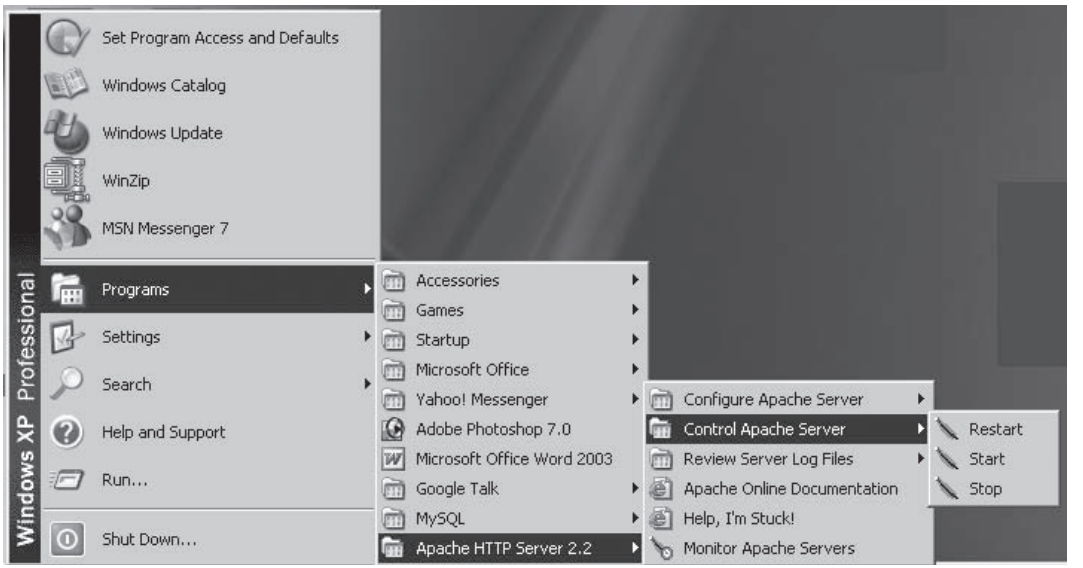


Figura A-21 Controles del servidor Apache en Windows

www.compumusicmania.com

Instalar SQLite

Dado que SQLite es un archivo ejecutable independiente, la instalación en Windows se realiza en un abrir y cerrar de ojos: simplemente ingresa como administrador (si estás utilizando Windows NT/2000/XP/Vista) y desempaqueta el archivo binario en un directorio temporal de tu sistema. Para mayor facilidad renombra el archivo binario *sqlite2.exe* por *sqlite.exe*.

La versión binaria de SQLite ya está instalada y lista para funcionar. Para probarla, ve a la sección "Probar SQLite".

Probar el software

Una vez que el software ha sido instalado con éxito y que los diferentes servidores están en funcionamiento, puedes verificar que todo funcione como es debido mediante unas cuantas pruebas sencillas.

Probar MySQL

Primero, arranca el programa cliente de línea de comandos de MySQL, cambiando al subdirectorio *bin/* de tu directorio de instalación MySQL y escribiendo el siguiente comando:

```
prompt# mysql –u root
```

Se te debe recompensar con un mensaje como el que aparece a continuación:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26288
Server version: 5.0.51a-community MySQL Community Edition (GPL)
Type 'help' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

En este punto, ya estás conectado al servidor MySQL y puedes comenzar a ejecutar comandos SQL o consultas para verificar que el servidor está funcionando apropiadamente. A continuación presentamos algunos ejemplos con sus respectivos datos de salida:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.13 sec)
mysql> USE mysql;
```

www.compumusicmania.com

```
Database changed
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| proc                      |
| procs_priv                |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
17 rows in set (0.01 sec)
mysql> SELECT COUNT(*) FROM user;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

Si ves datos de entrada similares a éstos, tu instalación de MySQL está funcionando correctamente. Sal del cliente de líneas de comando escribiendo el siguiente comando, y regresarás a la línea del principio:

```
mysql> exit
```

Si no ves datos de salida semejantes a los que se presentan arriba, o si tu SQL lanza mensajes de advertencia o errores, revisa el procedimiento de instalación en la sección anterior, al igual que los documentos que acompañan a la versión de MySQL que instalaste para saber qué anda mal.

Probar PHP

Una vez que instalaste con éxito PHP como módulo de Apache, debes probarlo para asegurarte de que el servidor Web reconoce los scripts PHP y los maneja correctamente.

www.compumusicmania.com

Para realizar esta prueba crea un script PHP en cualquier editor de texto que contenga las siguientes líneas:

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

Guarda este archivo como *prueba.php* en la raíz de los documentos de tu servidor Web (el subdirectorio *htdocs/* de tu directorio de instalación de Apache), y dirige tu explorador Web a http://localhost/prueba.php. Debes ver una página con la información de la construcción de PHP, como se muestra en la figura A-22.

Pon atención a la lista de extensiones para asegurarte de que estén activas las correspondientes a SimpleXML, MySQLi y PDO. Si no están activas, revisa el procedimiento de instalación, así como la documentación que acompaña al programa, para saber qué salió mal.

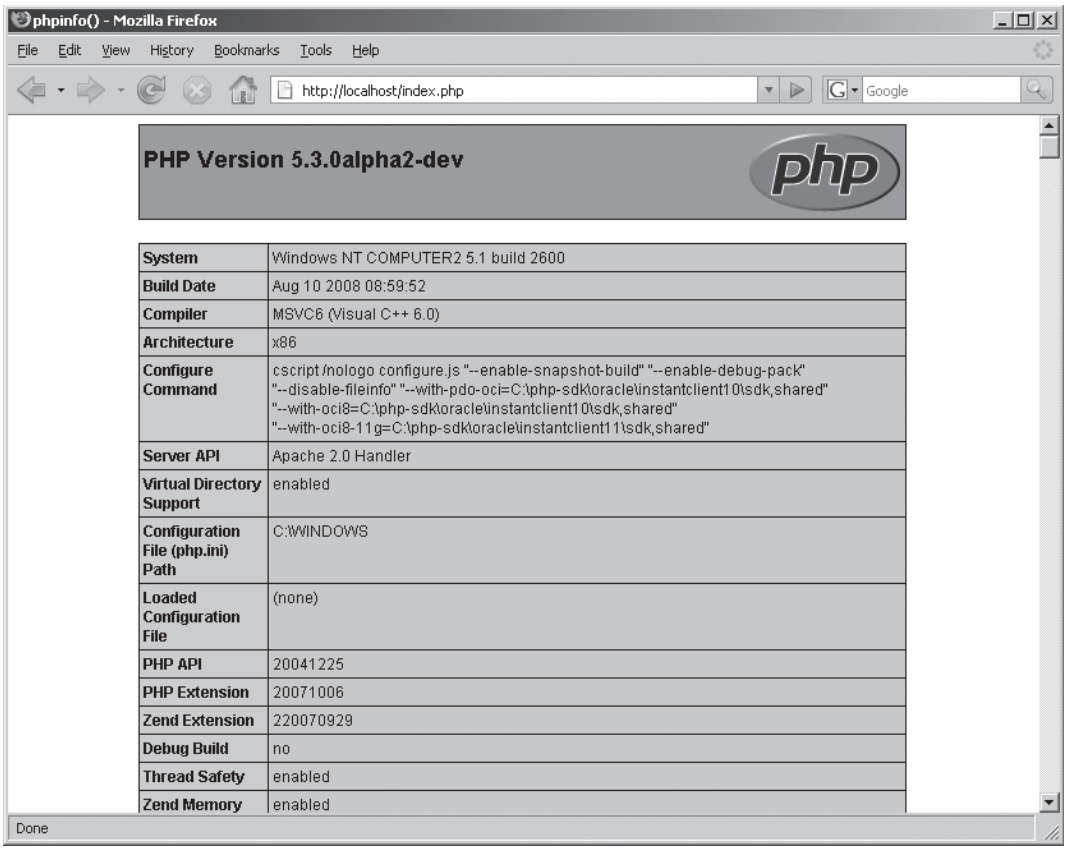


Figura A-22 Datos de salida del comando `phpinfo()`

www.compumusicamania.com

Probar SQLite

Una vez que has instalado SQLite, puedes probarlo al cambiar al directorio donde fue instalado y ejecutando el binario SQLite en la línea de comando, como se muestra aquí:

```
prompt# sqlite
```

Debes recibir una recompensa con el mensaje que se muestra a continuación:

```
SQLite versión 2.8.17
Enter ".help" for instructions
sqlite>
```

En este punto, puedes ejecutar comandos SQL o comandos internos SQLite para probar si las cosas están funcionando como es debido. He aquí algunos ejemplos:

```
sqlite> .show
     echo: off
  explain: off
  headers: off
     mode: list
nullvalue: ""
   output: stdout
separator: "|"
    width:
sqlite> CREATE TABLE prueba (
   ...> fld1 INTEGER PRIMARY KEY,
   ...> fld2 TEXT
   ...> );
sqlite> INSERT INTO prueba (fld2) VALUES ('Hola');
sqlite> SELECT * FROM prueba;
1|Hola
```

Si ves datos de salida parecidos a los anteriores, tu instalación de SQLite está funcionando correctamente. Sal del cliente de línea de comandos escribiendo el siguiente comando, y regresarás a la primera instancia.

```
sqlite> .quit
```

Realizar pasos posteriores a la instalación

Una vez que las pruebas fueron completadas, es posible que quieras realizar las siguientes dos tareas.

www.compumusicmania.com

Establecer la contraseña de superusuario en MySQL

En UNIX, la primera vez que se instala MySQL, el acceso al servidor de base de datos está restringido al administrador de la misma, también conocido como "root". Por defecto, este usuario se inicializa con una contraseña en blanco, lo que se considera en general como una mala práctica. Por lo tanto, debes remediarlo lo antes posible estableciendo una contraseña para este usuario a través de la utilidad *mysqladmin* incluida en el paquete, con la siguiente sintaxis en UNIX:

```
[root@host]# /usr/local/mysql/bin/mysqladmin –u root password 'nueva-
contraseña'
```

En Windows, puedes utilizar el MySQL Server Instance Config Wizard, que te permite establecer o restablecer la contraseña para el administrador de la base de datos (consulta la sección "Instalar en Windows" para conocer más detalles).

El cambio de la contraseña se aplica de inmediato, sin necesidad de reiniciar el servidor.

Pregunta al experto

P: ¿Cambiar la contraseña de "root" de MySQL en UNIX afecta a la misma cuenta en todo el sistema operativo?

R: No. El usuario "root" de MySQL no es el mismo que el superusuario del sistema operativo UNIX. Así que modificar uno no afecta al otro.

Configurar MySQL y Apache para comenzar automáticamente

En UNIX, los servidores MySQL y Apache tienen scripts para arrancar y terminar su ejecución. Estos scripts se localizan dentro de la jerarquía de instalación de cada programa. He aquí un ejemplo para utilizar el script de control del servidor MySQL:

```
[root@host]# /usr/local/mysql/support-files/mysql.server start
[root@host]# /usr/local/mysql/support-files/mysql.server stop
```

Y aquí un ejemplo de cómo utilizar el script de control de Apache:

```
[root@host]# /usr/local/apache/bin/apachectl start
[root@host]# /usr/local/apache/bin/apachectl stop
```

www.compumusicamania.com

- Para que MySQL y Apache arranquen automáticamente en el momento en que inicia UNIX, simplemente invoca su respectivo script de control con los parámetros apropiados desde tus scripts de arranque y detención en la jerarquía `/etc/rc.d/*`.
- Para arrancar MySQL y Apache automáticamente en Windows, añade un vínculo de los archivos binarios *mysqld.exe* y *apache.exe* al grupo Inicio. También puedes iniciar automáticamente MySQL instalándolo como servicio de Windows (ver la sección titulada "Instalar en Windows" para instrucciones).

Resumen

Como aplicaciones de código libre, MyQSL, SQLite, Apache y PHP están disponibles para una amplia gama de plataformas y arquitecturas, tanto en formato binario como en código fuente. Este capítulo muestra el proceso de instalación y configuración de los componentes de software necesarios para crear un ambiente de desarrollo en las dos plataformas más comunes, UNIX y Windows. También te enseñó a configurar tu computadora para lanzar estos componentes automáticamente cada vez que se arranca el sistema, y te ofreció algunos consejos para la seguridad básica de MySQL.

Para saber más sobre el proceso de instalación descrito en este capítulo, o para conocer detalles sobre manejo y asesoría en la detección y solución de problemas, puedes visitar las siguientes páginas:

- Notas de instalación de MySQL, en **http://dev.mysql.com/doc/refman/5.0/en/installing-binary.html**
- Lineamientos generales para compilar Apache en UNIX, en **http://httpd.apache.org/docs/2.2/install.html**
- Notas específicas para Windows para la instalación del archivo binario de Apache, en **http://httpd.apache.org/docs/2.2/platform/windows.html**
- Instrucciones de instalación de PHP para Windows, en **www.php.net/manual/en/install.windows.php**
- Instrucciones de instalación de PHP para UNIX, en **www.php.net/manual/en/install.unix.php**
- Instrucciones de instalación de PHP para Mac OS X, en **www.php.net/manual/en/install.macosx.php**
- Preguntas más frecuentes de SQLite, en **www.sqlite.org/faq.html**

www.compumusicmania.com