USERS

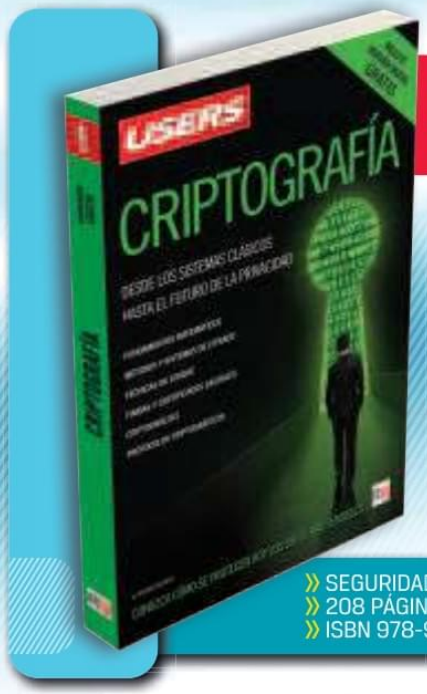
INCLUYE VERSIÓN DIGITAL GRATIS

PHP + MySQL desde cero

Teoría y práctica de la creación de bases de datos + Instalación y gestión de MySQL + Revisión de sentencias para consultas, inserción y actualización + Control de errores y administración de datos

RU

CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN



CONOZCA CÓMO SE PROTEGEN HOY LOS DATOS MÁS SENSIBLES

» SEGURIDAD
» 208 PÁGINAS
» ISBN 978-987-1949-35-9

EVITE FALLAS DE SEGURIDAD EN APLICACIONES WEB

» SEGURIDAD / INTERNET
» 320 PÁGINAS
» ISBN 978-987-1949-31-1

MEJORE EL RENDIMIENTO SIN REDISEÑAR LA ARQUITECTURA

» DESARROLLO / INTERNET
» 320 PÁGINAS
» ISBN 978-987-1949-20-5

TODAS LAS NOVEDADES DE LA SUITE DE OFICINA MÁS USADA

» MICROSOFT / OFFICE
» 320 PÁGINAS
» ISBN 978-987-1949-21-2

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

MÁS INFORMACIÓN / CONTÁCTENOS

usershop.redusers.com +54 (011) 4110-8700 usershop@redusers.com

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

PHP + MYSQL DESDE CERO

RedUSERS

TÍTULO: PHP + MySQL desde cero

AUTOR: Francisco Minera

COLECCIÓN: Desde Cero

FORMATO: 19 x 15 cm

PÁGINAS: 192

Copyright © MMXIV. Es una publicación de Fox Andina en coedición con DÁLAGA S.A. Hecho el depósito que marca la ley 11723. Todos los derechos reservados. Esta publicación no puede ser reproducida ni en todo ni en parte, por ningún medio actual o futuro sin el permiso previo y por escrito de Fox Andina S.A. Su infracción está penada por las leyes 11723 y 25446. La editorial no asume responsabilidad alguna por cualquier consecuencia derivada de la fabricación, funcionamiento y/o utilización de los servicios y productos que se describen y/o analizan. Todas las marcas mencionadas en este libro son propiedad exclusiva de sus respectivos dueños. Impreso en Argentina. Libro de edición argentina. Primera impresión realizada en Sevagraf, Costa Rica 5226, Grand Bourg, Malvinas Argentinas, Pcia. de Buenos Aires en VII, MMXIV.

ISBN 978-987-1949-66-3

Minera, Francisco
PHP + MySQL desde cero.
1a ed. - Ciudad Autónoma de Buenos Aires : Fox Andina; Buenos Aires: Dalaga, 2014.
192 p. ; 19x15 cm. - (Desde cero; 37)

ISBN 978-987-1949-66-3

1. Informática. I. Título
CDD 005.3

Prólogo al contenido

Tanto el lenguaje de programación como el gestor de bases de datos están posicionados en el mercado desde hace largo tiempo. Podemos analizar algunos hechos respecto a ellos.

En relación con las modificaciones, han sido positivas para los desarrolladores. Las nuevas versiones tanto de PHP como de MySQL han sido aceptadas de muy buena manera, y las mejoras incluidas han dado respuestas a los pedidos de la gran comunidad de usuarios.

Otro punto es el continuo crecimiento del número de programadores que optan por estas herramientas. Por un lado, están los que, sin experiencia previa, encuentran que pueden comenzar a desarrollar aplicaciones funcionales, incluso durante el proceso de aprendizaje. Por otro lado, quienes provienen de otros lenguajes o tecnologías de desarrollo ven que tanto PHP como MySQL les permiten elaborar soluciones rápidas, estables, de bajo costo y fácilmente actualizables.

Una de las virtudes del lenguaje (su facilidad de aprendizaje) deriva en algo que podría verse como negativo: hay muchas personas que programan en PHP. Si bien la competencia es algo que beneficia tanto a empleados como a empleadores, al haber tanta oferta, las condiciones de los empleadores tienen mayor preponderancia. Desde un lado positivo, la competencia genera que los programadores se esfuercen por ser mejores, y esto a su vez deriva en mejores condiciones laborales.

Un último punto está relacionado con la visión de algunas empresas para con el lenguaje, que, ya sea por publicidad, falta de conocimiento o soporte técnico, prefieren apostar a alternativas como .Net o Java. Esperamos que, luego de leer este manual, puedan poner en práctica sus conocimientos y sacar sus propias conclusiones.

El libro de un vistazo

Conocer a fondo todas las características y posibilidades que un lenguaje de programación como PHP nos brinda es una tarea que requiere paciencia y aplicación por parte del desarrollador. A lo largo de los siguientes capítulos intentaremos ir asimilando, poco a poco, las técnicas y los métodos necesarios para sacar provecho de la potencia del lenguaje.

*01 PHP

En este capítulo analizaremos información relevante relacionada con el lenguaje de programación PHP: cómo debemos adquirirlo, las versiones y sus principales características.

*02 BASES DE DATOS

En este capítulo aprenderemos lo que es una base de datos, conoceremos su estructura principal para almacenar información y haremos una introducción a lo que necesitamos para utilizar MySQL.

*03 MYSQL

Para poner en práctica los conceptos vistos en el capítulo anterior, elegimos una base de datos muy popular en este momento y con mucho futuro: MySQL. Analizaremos las razones de esta afirmación y por qué MySQL es el complemento ideal de PHP.

*04 PHP Y MYSQL

En este capítulo brindaremos información relevante sobre las distintas alternativas que nos ofrece el lenguaje de programación PHP para acceder y definir estructuras de bases de datos utilizando el motor MySQL.

*05 UTILIZAR SQL EN PHP

Una vez creada la base de datos y establecida una conexión, comenzaremos a interactuar con los datos para realizar consultas de selección y de modificación. En este capítulo encontraremos la información relacionada con este proceso.

*06

MANEJO DE ERRORES

Tanto PHP como MySQL nos brindan funciones y opciones de configuración que nos permitirán tener un acceso controlado y un entendimiento de los errores que puedan surgir en nuestras aplicaciones. En este capítulo aplicaremos estas herramientas.

*Ap ON WEB

ADMINISTRACIÓN DE BASES DE DATOS

Mientras desarrollamos aplicaciones, cumplimos múltiples roles. Pero, cuando trabajamos en un sistema real, las tareas se vuelven específicas. En este capítulo, entenderemos la naturaleza y el rol de los administradores de bases de datos.

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, podrá encontrar una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para que pueda distinguirlos en forma más sencilla, cada recuadro está identificado con diferentes iconos:

CURIOSIDADES E IDEAS

ATENCIÓN

DATOS ÚTILES Y NOVEDADES

SITIOS WEB

Contenido del libro

Prólogo **3**
El libro de un vistazo **4**
Información complementaria **5**
Introducción **10**

*01

PHP

Características de PHP **12**
PHP 5 13
Qué versión de PHP utilizar 16
Extensiones en PHP **16**
Ventaja de trabajar con extensiones/ bibliotecas 17
Extensiones incorporadas 18
Bases de datos 19
Portabilidad **21**
Resumen **23**
Actividades **24**

*02

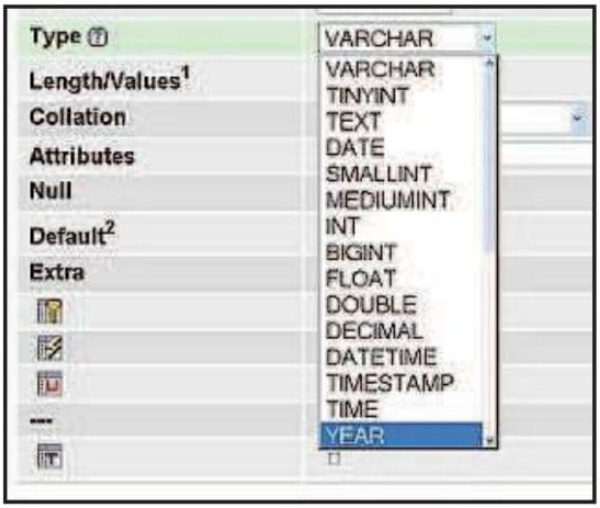
Bases de datos

Qué es una base de datos **26**
Arquitectura de bases de datos 26
Modelo de datos relacional **27**
Base de datos relacional 27
Tipos de relaciones entre tablas 34
Gestor de base de datos: MySQL **40**
Por qué MySQL 40
Obtener MySQL 40
Licencia de uso 41
Resumen **41**
Actividades **42**

*03

MySQL

Tipos de tablas **44**
ISAM 44
MyISAM 44
MERGE 46
HEAP 47
InnoDB 48
BerkeleyDB 49
En qué casos usar cada tabla 50
Tipos de datos **50**
Cadenas de caracteres 50
Numéricos 53
Fecha y hora 56
Referencia de funciones **59**



Funciones para trabajar
con cadenas de caracteres 60
Funciones para trabajar
con campos numéricos 73
Funciones para trabajar
con fecha y hora 81
Funciones de conversión 93
Funciones agregadas o estadísticas 94
Operadores de comparación 95
Operadores lógicos 96
El uso del monitor de MySQL 98
Resumen 101
Actividades 102

*04

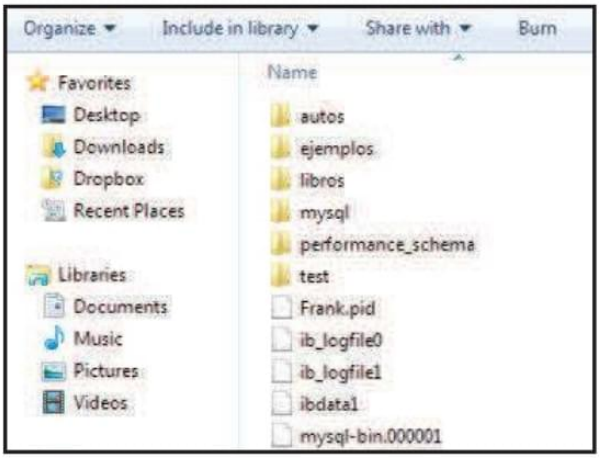
PHP Y MYSQL

Conectar PHP con MySQL 104
mysql_connect 104
mysql_pconnect 107
Ejecución de sentencias 108
Creación de base de datos
a través de PHP 110
Selección de una base de datos 113
Creación de tablas a través de PHP 114
Obtener listados de bases de datos
y tablas a través de PHP 116
Uso de múltiples bases
de datos en una aplicación 119
Cerrar conexión con la base de datos 121
Ejemplo práctico: autos.php 123
Resumen 125
Actividades 126

*05

Utilizar SQL en PHP

Consultas de selección 128
Selección simple 128
Consultas multitabla 128
Subconsultas 129
Recorrer las filas devueltas
de una consulta 129
mysql_fetch_array 129
mysql_fetch_row 134
mysql_fetch_object 135
mysql_fetch_assoc 137
mysql_fetch_field 140
Moverse entre registros 142
mysql_data_seek 142
Número de registros y campos devueltos ... 144
mysql_num_rows 144
mysql_affected_rows 145
mysql_num_fields 147
Insert 148
Campos autoincrementables 148
mysql_insert_id 149
Delete 150



Update 152
Ejemplo práctico: libros.php 152
Resumen 167
Actividades 168

*06

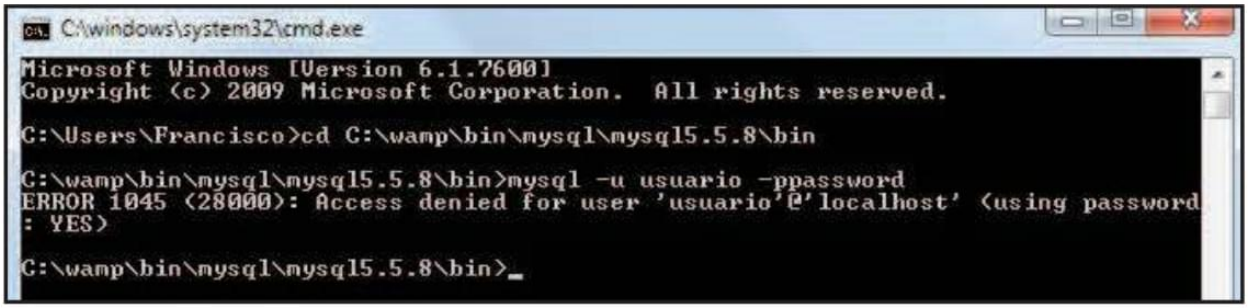
Manejo de errores

Reportes de error en PHP 170
error_reporting 170
display_errors 173
display_startup_errors 174
log_errors 175
error_log 176
log_errors_max_len 177
ignore_repeated_errors 177
ignore_repeated_source 180
track_errors 180
html_errors 181
error_prepend_string 183
error_append_string 184
warn_plus_overloading 184
mysql_error 184
mysql_errno 186
register globals 186
perror 191
Manejo de excepciones 191
Resumen 191
Actividades 192

*Ap ON WEB

Administración de bases de datos

El rol del administrador
El rol del usuario
Sistema de privilegios
Tabla user
Tabla host
Tabla db
Tablas tables_priv y columns_priv
Jerarquías de acceso
Comandos grant y revoke
Gestión de privilegios con INSERT
Gestión de privilegios con SELECT
Gestión de privilegios con UPDATE
Gestión de privilegios con DELETE
Gestión de usuarios
Baja de usuarios
mysql_change_user
Listar todos los usuarios mediante una consulta
Resumen
Actividades



VISITE NUESTRA WEB

EN NUESTRO SITIO PODRÁ ACCEDER A UNA PREVIEW DIGITAL DE CADA LIBRO Y TAMBIÉN OBTENER, DE MANERA GRATUITA, UN CAPÍTULO EN VERSIÓN PDF, EL SUMARIO COMPLETO E IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA.

RedUSERS
COMUNIDAD DE TECNOLOGIA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios y todos los elementos necesarios para asegurar un aprendizaje exitoso.

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com usershop@redusers.com + 54 (011) 4110-8700

Introducción

El objetivo principal de este libro es introducir al lector en el desarrollo de aplicaciones web, tomando como base dos de las herramientas más populares de la actualidad: el lenguaje de programación PHP y la base de datos relacional MySQL.

Una enorme cantidad de sitios web utilizan estas tecnologías, algo muy motivante para aquellos estudiantes de sistemas o simplemente aficionados que quieran insertarse en el mercado y que necesiten hacerlo a través de tecnologías probadamente aceptadas.

PHP y MySQL brindan una alternativa más a quienes quieren desarrollar soluciones altamente sofisticadas orientadas a todo tipo de emprendimientos, desde los más simples a los más complejos. Este manual tiene la finalidad de dar algunas respuestas que a su vez lleven a nuevas preguntas, es decir, disparadores que obliguen al lector a continuar permanentemente la búsqueda de nuevos niveles de especialización.

Los requisitos para el lector son mínimos: tener experiencia en el uso de algún sistema operativo y una idea básica acerca del funcionamiento de la arquitectura cliente/servidor; esto implica realizar una petición a un servidor web a través de un navegador y obtener una respuesta adecuada.

En definitiva, tanto PHP como MySQL están transitando un camino iniciado hace ya mucho tiempo, y los resultados son realmente beneficiosos para todos los que de alguna manera están relacionados con el mundo del desarrollo de aplicaciones web.

*01

PHP

En este capítulo veremos algunos puntos fundamentales sobre el lenguaje de programación PHP: cómo adquirirlo, sus versiones, sus principales características y demás temas que servirán de base para los capítulos siguientes.

Características de PHP **12**
PHP 5 13
Qué versión de PHP utilizar 16

Extensiones en PHP **16**
Ventaja de trabajar con extensiones/bibliotecas 17
Extensiones incorporadas 18
Bases de datos 19

Portabilidad **21**

Resumen **23**

Actividades **24**

Características de PHP

PHP, acrónimo de **PHP Hypertext Preprocessor**, es un lenguaje de programación que se utiliza principalmente para el desarrollo de sitios web, pero que para muchos es un lenguaje de propósito general, y el uso que se le dé dependerá en parte de lo que necesite el programador.

Entre las características que hacen de PHP un lenguaje popular y muy poderoso para desarrollar aplicaciones, podemos citar las siguientes:

- Programación de páginas dinámicas en servidores.
- Programación de aplicaciones de escritorio con GTK (PHPGTK).
- Soporte para trabajar con múltiples bases de datos.
- Soporte para múltiples plataformas.
- Soporte para múltiples servidores.
- Facilidad de aprendizaje.
- Portabilidad de código entre diferentes plataformas.
- Total libertad para distribuir las aplicaciones.

Para obtener una copia, deberemos ingresar a su sitio web, **www.php.net/downloads**, y seleccionar la opción acorde con nuestro sistema operativo.

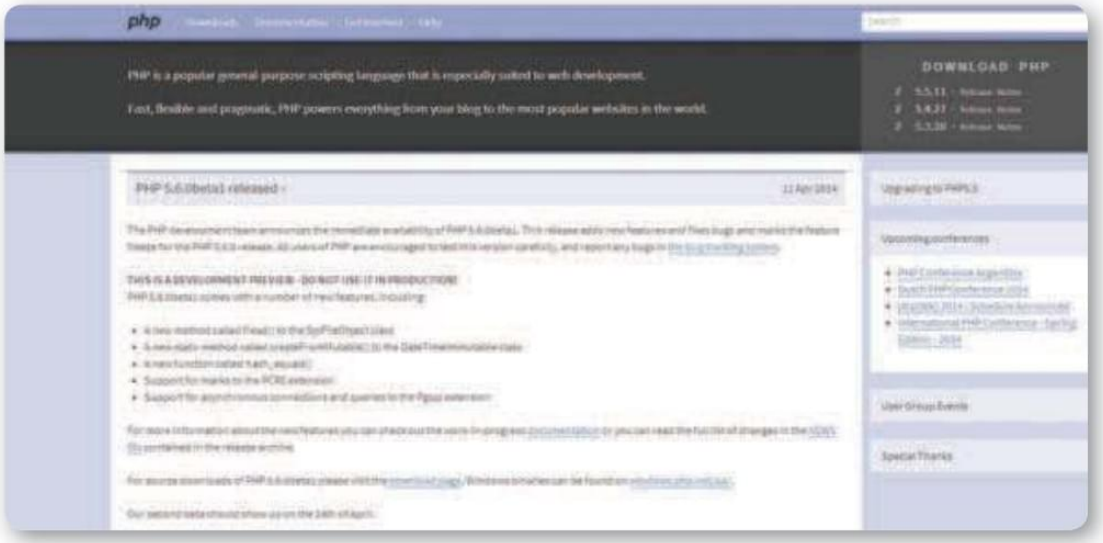


Figura 1. Área de descargas en el sitio web de PHP.

PHP 5

PHP 5 experimentó cambios importantes que revolucionaron el lenguaje. La primera versión de PHP 5 se liberó en junio de 2003, pero la primera considerada estable es la de julio de 2004. En 2014, contamos con la versión PHP 5.5.10, de la que a continuación destacaremos algunas de sus características.

Generalidades

- Podemos ver las últimas actualizaciones en: **http://php.net/ChangeLog-5.php#5.5.10**.
- En cuanto a la programación orientada a objetos, PHP ofrece notables mejoras que lo hacen una alternativa totalmente competente en este aspecto en comparación a otros lenguajes con historia en este campo.
- Mejoras en el soporte de XML (gestión de información): **https://wiki.php.net/internals/windows/libs/libxml2**.
- Mejoras en el manejo de LDAP (manejo de directorios): **https://wiki.php.net/internals/windows/libs/ldap**.
- Mejoras sobre el motor Zend (desempeño del lenguaje): **http://zend.com/en/company/community/php**.

Otras de las características principales que debemos tener en cuenta son el acceso a las bases de datos y las herramientas disponibles para ello.

REDUSERS PREMIUM

Para obtener material adicional gratuito, ingrese a la sección Publicaciones/Libros dentro de **http://premium.redusers.com**. Allí encontrará todos nuestros títulos y podrá acceder a contenido extra de cada uno, como sitios web relacionados, programas recomendados, ejemplos utilizados por el autor, apéndices, archivos editables o de código fuente. Todo esto ayudará a comprender mejor los conceptos desarrollados en la obra.

SQLite

SQLite es una librería desarrollada en C que implementa un motor de base de datos accesible por varios lenguajes: PHP, PYTHON, C, etc.

No permite que múltiples usuarios ingresen simultáneamente en modo escritura a la base de datos; si esto sucede, el mecanismo de bloqueo la inmoviliza por completo. Por eso, esta librería está especialmente indicada cuando se requiera una gran rapidez en las consultas y sea suficiente que un único usuario realice modificaciones. Es entre dos y tres veces más rápida que bases de datos como MySQL o PostgreSQL, ya que no es un servidor, sino una base de datos de escritorio, por lo que el acceso a los datos es más directo, entre otras razones. Brinda soporte para un máximo de 2 terabytes de datos. Además, es un software libre, y podremos descargarlo y obtener más información en su sitio oficial: **www.hwaci.com/sw/sqlite**.

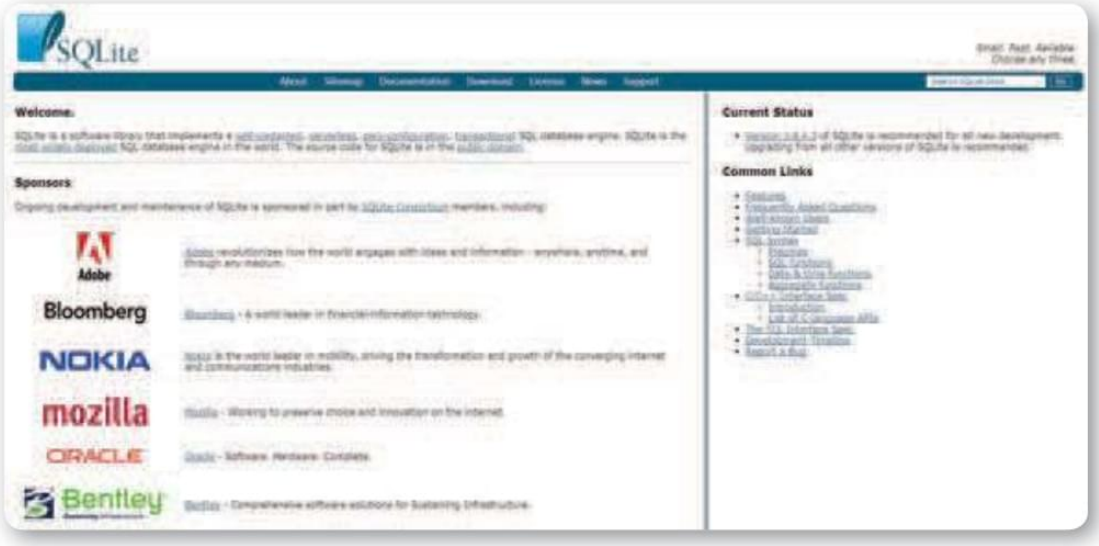


Figura 2. Página oficial de SQLite; podemos observar los diferentes patrocinadores.

MySQLi

MySQLi (con **i** de **improved**, mejorado) es una extensión que permite acceder a las funcionalidades provistas por MySQL a partir de la versión 4.1.2. Se ha probado que ofrece mayor velocidad, en algunas operaciones es hasta cuarenta veces más rápida que la extensión MySQL anterior.

En la actualidad, se está trabajando para que la migración desde la extensión de MySQL a MySQLi sea lo más sencilla posible. Esto se logra, entre otras cosas, manteniendo las funciones y sus nombres lo más parecidos que se pueda en ambas extensiones. Si observamos códigos programados con las dos extensiones, no notaremos, en general, grandes diferencias.

Podemos encontrar una explicación acerca de cómo instalar la extensión MySQLi en: **www.php.net/mysqli**. Para tener más clara la diferencia entre estas librerías, en la siguiente tabla veremos sus características.

DIFERENCIAS ENTRE SQLITE Y MYSQL

▼ CARACTERÍSTICA	▼ SQLITE	▼ MYSQL
Uso	Puede funcionar perfectamente para sitios de tráfico bajo-medio.	Funciona con sitios de gran envergadura.
Acceso	Es monousuario, no permite concurrencia de conexiones.	Permite múltiples consultas y modificaciones al mismo tiempo.
Tipo	No es una base de datos, sino un fichero con datos organizados.	Gestor de base de datos.
Datos	No tiene tipos de datos.	Varios tipos de datos.
Usuarios	No hay gestión de usuarios.	Posibilidad de gestionar usuarios con diferentes niveles de acceso.
Seguridad	La seguridad se basa en el sistema de permisos de ficheros establecido por Unix/Linux.	Gestión propia de seguridad.
Facilidad	Para hacer una copia o migrar la aplicación, basta con mover de sitio el fichero.	Debe realizar el proceso de backup y restauración como gestor de base de datos.

Tabla 1. Comparación entre librerías de almacenamiento de datos.

Qué versión de PHP utilizar

Si nuestra aplicación ya está funcionando y cumple con los propósitos, no hay razón para migrar a otra versión de PHP. Ahora bien, si queremos desarrollar un sitio desde cero, lo que conviene es utilizar la versión reciente del lenguaje. Quizás no aprovechemos todas las mejoras ni todas sus características, pero si quisiéramos hacerlo no tendríamos que actualizar a la versión más reciente, ni reescribir el código, ni entrar en temas complejos, como es la compatibilidad del lenguaje entre versiones.

Aunque a veces se diga que esta decisión depende de los requerimientos del desarrollador, PHP 5 es un tema aparte porque, como se dijo, ha sido mejorado de manera sustancial y continua. El soporte para la programación orientada a objetos es algo que ya resulta muy popular en otros lenguajes, y PHP no es la excepción. Gran cantidad de código está siendo escrito bajo este paradigma y, mientras más rápido se acostumbre el programador de PHP, más en carrera estará a la hora de actualizar sus conocimientos, incluso, en el momento de buscar empleo.

Extensiones en PHP

Cuando programamos en un lenguaje, no solo en PHP, nos valemos de funciones o procedimientos, ya sea para resolver problemas o modular el código y hacerlo más legible o reutilizarlo sin escribir lo mismo más de una vez.

GESTOR DE BASE DE DATOS

Un sistema gestor de base de datos o **SGBD** es un programa encargado de administrar la información que se va a registrar, también los usuarios y permisos de acceso, y programar consultas/registros/actualización/eliminación. En resumen, entre los grandes administradores de bases de datos, podemos nombrar: Oracle, Microsoft SQL Server, MySQL.

Las extensiones no son ni más ni menos que conjuntos de funciones disponibles para programar, pero, para ser más precisos, podríamos dividir las funciones en dos grupos. Por un lado, están las que vienen incorporadas con el lenguaje, llamadas **built in**. Por otro lado, tenemos las que se encuentran en las bibliotecas añadidas, que debemos incluir en el sistema de manera específica, estas son las **extensiones** o **bibliotecas**. Se podría decir que para utilizar ciertas funciones hay que extender el lenguaje.

Una vez que se instalan y se habilitan esas bibliotecas (el procedimiento podemos encontrarlo en el libro *PHP desde cero*), el comportamiento de las funciones componentes dentro del código de los programas es idéntico al de cualquier función o procedimiento, o sea que la programación se vuelve independiente y transparente al origen de las funciones.

Las extensiones en PHP pueden agruparse por funcionalidad; podríamos encontrarnos con una extensión para manipular cadenas de caracteres, otra para acceder a bases de datos, otra para trabajar con archivos XLS y demás.

Ventaja de trabajar con extensiones/bibliotecas

Solo cargamos las bibliotecas cuando las usamos, es decir, no hay sobrecarga de trabajo innecesaria. PHP ya tiene demasiadas funciones incorporadas, y sería poco recomendable iniciar el motor para soportar cientos de funciones de las cuales probablemente necesitemos solo algunas.

CÓDIGO DE BARRAS

Una de las equivocaciones frecuentes a la hora de publicar archivos PHP en un servidor con un sistema operativo diferente del nuestro es la de escribir las rutas a archivos o directorios de manera incorrecta. Los sistemas Unix compatibles usan el símbolo **/** (barra hacia la derecha) para acceder a subdirectorios, y los sistemas Windows usan el símbolo **** (barra hacia la izquierda).

Además, al añadir una biblioteca no hace falta reinstalar PHP, solo habilitar desde el archivo **php.ini** lo que necesitamos. Esto es la modularidad.

Extensiones incorporadas

PHP contiene, sin necesidad de ningún tipo de instalación ni habilitación adicional, entre otras, las siguientes funciones:

- Manejo de matrices.
- Funciones matemáticas.
- BCMath.
- Manejo de clases/objectos.
- Manejo de variables de tipo carácter.
- Tratamiento de fecha/hora.
- Acceso directo a entrada/salida.
- Funciones de directorio.
- Gestión de errores y registros.
- Funciones de sistema de archivos.
- Utilización del protocolo FTP.
- Utilización del protocolo HTTP.
- Funciones de correo.
- Funciones de red.
- Ejecución de programas.

EXTENSIÓN DE LA DISTRIBUCIÓN

En el momento de descargar la última versión de PHP desde su sitio oficial, encontraremos archivos con distintas extensiones (**.ZIP**, **.TAR**, **.BZ**, **.TGZ**). Debemos saber que todos contienen la misma información, y la elección de descargar uno u otro dependerá de si tenemos o no un programa descompresor que pueda trabajar con dichas extensiones.

- Manejo de sesiones.
- Funciones de secuencia.
- Funciones de cadenas.
- Funciones URL.
- Manejo de variables.

Para poder acceder a las demás extensiones, debemos activarlas desde el archivo **php.ini**, o bien, incorporarlas en el momento de compilar PHP e instalar las bibliotecas por separado (en sistemas operativos que requieren compilar PHP para su instalación). Para saber qué bibliotecas tenemos activas en nuestro sistema, podemos utilizar la función **PHPinfo()** de la siguiente manera:

```
<?PHP
// funcion PHPinfo
phpinfo();
?>
```

Normalmente, cuando no recordamos cómo instalar o habilitar una extensión/biblioteca en particular en nuestro sistema, bastará con buscar la referencia en el manual de PHP (**www.php.net/manual**).

Bases de datos

En este libro lo principal es la explicación del manejo de bases de datos. El lenguaje PHP soporta, entre otras, el manejo de las siguientes bases:

- dBase
- Informix
- InterBase
- MS SQL Server
- MySQL

- msql
- Oracle
- PostgreSQL
- Sybase

Además con ODBC (*Open Database Connectivity*, **conectividad abierta de bases de datos**) se puede acceder a casi cualquier base de datos existente en el mercado. Este brinda un conjunto de comandos que son traducidos a instrucciones específicas de una base de datos en particular a través de drivers provistos por estas.

Es claro que utilizar funciones nativas da más réditos en cuanto a la velocidad de respuesta, comparado con trabajar con algún mediador tipo ODBC. Por otro lado, cuando se emplean bases de datos con una gran cantidad de prestaciones como **Oracle** y se utiliza ODBC, OLE, ADO, etcétera, se pierde gran parte del poder, puesto que hay funciones propias de la base de datos que no se pueden utilizar con un mediador genérico como estos, por no estar implementadas. Por lo tanto, si se quiere utilizar todo el potencial de la base de datos, es preferible acceder con funciones nativas, como las que ofrece PHP en sus extensiones

Por ejemplo, si en nuestro proyecto sabemos que vamos acceder solo a la base de datos PostgreSQL, no hay necesidad de acceder con ODBC, ya que PHP nos provee de una extensión (**php_pgsql.dll**) para que podamos hacerlo de modo nativo.

AVANZAR EN DISTINTAS DIRECCIONES A LA VEZ

Dentro de la amplia oferta de lenguajes de programación –destinados al desarrollo de aplicaciones tanto web como de escritorio–, existen actualmente algunos que tienen una sintaxis muy similar a la de PHP. Esta es, quizás, otra de las ventajas de este lenguaje: aprender a programar en él, da la posibilidad de hacerlo luego en otros.

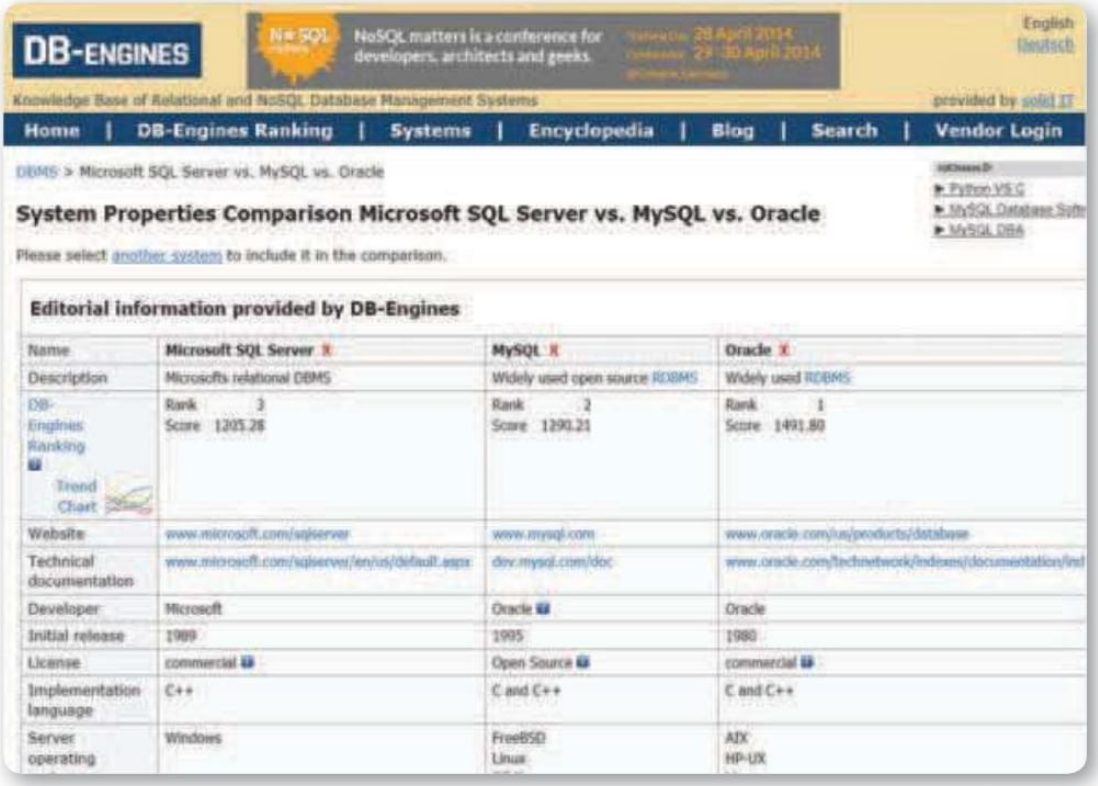


Figura 3. Una página muy útil es **http://db-engines.com**, que compara distintas bases de datos.

Portabilidad

PHP es un lenguaje multiplataforma, lo que significa que está preparado para trabajar sobre distintos sistemas operativos. Pero la portabilidad incide también en el hecho de que no es necesario realizar grandes cambios en el código fuente de una aplicación escrita en PHP en el momento de trasladarla de una plataforma a otra. Por ejemplo, si lo deseamos, se puede desarrollar una aplicación en Windows o MAC, pero luego subir el mismo código a un servidor que esté corriendo Linux.

La portabilidad de PHP es, sin duda, un punto fuerte frente a lenguajes como ASP/ASP.NET, que necesitan de componentes adicionales para correr en algunas plataformas. PHP funciona en una gran cantidad de sistemas operativos y sin necesidad de un componente adicional que debamos comprar.

	Windows	Mac OSX	Linux	BSD	Unix	z/OS
Adaptive Server Enterprise	Si	Si	Si	Si	Si	No
ANTs Data Server	Si	Si	Si	Si	Si	?
DB2	Si	No	Si	No	Si	Si
Firebird	Si	Si	Si	Si	Si	Quizá
HSQLDB	Si	Si	Si	Si	Si	Si
Infomix	Si	Si	Si	Si	Si	No
Ingres	Si	?	Si	?	Si	Quizá
InterBase	Si	No	Si	No	Si(Solaris)	No
SapDB	Si	No	Si	No	Si	?
MaxDB	Si	No	Si	No	Si	?
Microsoft SQL Server	Si	No	No	No	No	No
My SQL	Si	Si	Si	Si	Si	Quizá
Oracle	Si	Si	Si	Si	Si	Si
Postgre SQL	Si	Si	Si	Si	Si	No
Small SQL	Si	Si	Si	Si	Si	Si
SQLite	Si	Si	Si	Si	Si	Quizá

Figura 4. Comparación de diferentes recursos para el manejo de base de datos y su compatibilidad en diferentes sistemas operativos.

Actualmente, se puede ejecutar bajo los servidores web Apache (incluso en la versión 2.0), IIS (*Internet Information Server*), PWS (*Personal Web Server*), AOLServer, Roxen, OmniHTTPd, O'Reilly Website Pro,

MIRAR Y APRENDER

Por lo general, se logra un mayor entendimiento del lenguaje cuando se lee código escrito por otras personas. Un mismo problema puede resolverse de muchas maneras, y no quedarse solo con un punto de vista ayuda a abrir nuestra mente e incorporar nuevas formas de encarar la escritura de un código.

Sambar, Xitami, Caudium, Netscape Enterprise Server, THTTPD, y otros. Las posibles incompatibilidades entre plataformas son las siguientes:

- Ruta a archivos/directorios.
- Bibliotecas que solo funcionan en algunos sistemas, por ejemplo:
  - Funciones W32api (solo para plataformas Windows de 32 bits).
  - Funciones para el manejo de impresoras (solo para Windows 9.x, ME, NT4 y 2000).
  - Las funciones COM para Windows.
  - Funciones de acceso directo a E/S (no disponibles para sistemas Windows).
  - Funciones GMP (que permiten trabajar con enteros de longitud variable; no disponibles para sistemas Windows).
  - Funciones para el control de procesos (no disponibles para sistemas Windows).
  - Funciones FAM (notifican cambios en archivos y directorios; no disponibles para sistemas Windows).
  - Funciones POSIX (no disponibles para sistemas Windows).
  - Funciones para Ncurses (no disponibles para sistemas Windows).

Incluso, hay bibliotecas compatibles con Windows. Cuando no recordamos si una biblioteca es compatible o no con nuestro sistema, basta con buscar la referencia en el manual de PHP y verificar si hay algún requerimiento.

RESUMEN

En este capítulo hemos realizado una reseña de las principales virtudes del lenguaje de programación PHP: qué hace, para qué sirve y sus extensiones principales para bases de datos. En definitiva, un punto de partida para conocer lo que el lenguaje nos tiene preparado para el manejo de bases de datos.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es una extensión?
2. ¿Cuál es la ventaja de trabajar con extensiones?
3. ¿En qué directorio de su sistema se encuentran las extensiones de PHP?
4. ¿En qué versión de PHP se incorporó la extensión MySQLi? ¿Para qué sirve?
5. Nombre tres de las nuevas características que vienen con PHP.
6. ¿Qué es SQLite?
7. ¿PHP puede trabajar con múltiples servidores web o solo con Apache?

EJERCICIOS PRÁCTICOS

1. ¿Cuál es su versión de PHP? Responda utilizando la función de PHP que devuelve tal información.
2. ¿Qué extensiones tiene habilitadas en su sistema? Responda inspeccionando el archivo **php.ini**, o bien, a través de la función **phpinfo**.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: **profesor@redusers.com**.

*02

Bases de datos

En este capítulo, aprenderemos a confeccionar bases de datos (tablas, relaciones, claves, etc.) y crearemos las primeras herramientas para el almacenamiento masivo de información. Concluiremos con la instalación y el uso inicial del gestor de bases de datos MySQL, que nos permitirá digitalizar la creación de nuestros diseños.

Qué es una base de datos.....26
Arquitectura de bases de datos 26

Modelo de datos relacional.....27
Base de datos relacional 27
Tipos de relaciones entre tablas...34

Gestor de base de datos: MySQL.....40
Por qué MySQL 40
Obtener MySQL.....40
Licencia de uso 41

Resumen.....41

Actividades.....42

Qué es una base de datos

Una **base de datos** es una estructura organizada de datos relacionados entre sí que nos permite obtener, eventualmente, información actualizada acerca de una organización. Se dice **eventualmente** porque por sí misma no produce ninguna mejora: el beneficio se logra diseñándola en forma correcta a partir de lo que necesitamos (requerimientos) y manteniendo sus datos actualizados.

Arquitectura de bases de datos

En 1975, el comité **ANSI-SPARC** (*American National Standard Institute-Standards Planning and Requirements Committee*) propuso una arquitectura de tres niveles para los sistemas de bases de datos, cuyo objetivo es el de separar las aplicaciones de la base de datos física. Los tres niveles son:

- **Interno**: es el nivel más bajo de abstracción. Define todos los detalles de cómo funciona el almacenamiento en la base de datos y los métodos de acceso a la información.
- **Conceptual**: este nivel se aleja de los detalles técnicos y se concentra en las necesidades de los usuarios. Se describen entidades, atributos, relaciones, operaciones de los usuarios y restricciones a sus acciones. Es una representación de los datos desde el punto de vista de una organización.

ABSTRACCIÓN DE DATOS

Así como en la POO la abstracción de información implica obtener del mundo real datos útiles, en este caso, la abstracción se utiliza para el almacenamiento de registros. Por ejemplo, de un par de zapatillas podemos determinar: marca, tipo, material, tamaño, etc.

- **Externo**: aquí se define qué partes de la base de datos podrán ser vistas y cuáles serán ocultas a qué usuarios. Es el nivel de mayor abstracción.

Modelo de datos relacional

La idea primaria de los sistemas informáticos es poder plasmar, a través de diferentes herramientas, situaciones del mundo real. Para traspasar la realidad de una situación concreta a una base de datos existe una serie de pasos intermedios que facilitan esta tarea. Una de estas herramientas es el modelo de datos, que permite describir de modo abstracto en qué forma se va a almacenar y a recuperar la información existente en una base. Quizás el más conocido sea el **relacional**, pero existen otros modelos de administración de datos:

- Modelo jerárquico
- Modelo de red
- Modelo orientado a objetos

Nos centraremos en el modelo que vamos a utilizar para confeccionar bases de datos: el modelo relacional.

UNA BASE DE DATOS ES UNA ESTRUCTURA DE DATOS RELACIONADOS ENTRE SÍ

Base de datos relacional

El modelo relacional es, sin dudas, el más popular desde hace tiempo. Tomó notoriedad en 1970 y fue creado por **Edgar Frank Codd**. Con él se imponen conceptos tales como tabla (arreglo bidimensional), fila y columna. Los datos se recuperan a través de lenguajes de consulta (el más popular es SQL, pero existen otros) que mantienen la compatibilidad aun entre sistemas gestores de bases de datos de distintas compañías o sistemas operativos.

Entendemos el modelo relacional como una propuesta para ver los datos como objetos del mundo real, diferenciables entre sí por sus características básicas. Un objeto dado puede ser descripto por la colección de características que tiene (llamadas **atributos**) y distinguido de otros objetos a partir de eso mismo. Este modelo admite relaciones uno a varios, uno a uno y varios a varios, haciendo referencia a la información que guardaremos, por ejemplo: (mundo real) una factura de una casa de computación alberga los datos de uno o muchos productos; esto equivaldría a una relación de uno (factura) a muchos (productos).

Para convertir los datos del mundo real al abstracto, debemos considerar las agrupaciones de información; tomando el ejemplo anterior, la **factura** se convertiría en una **tabla**.

A continuación, explicaremos los diversos componentes de una base de datos relacional, como el esquema de los datos por almacenar, denominado **tabla**; las características, llamadas **atributos** y cómo se generan las **relaciones** entre los diversos registros incluidos dentro de las tablas.

Tabla

Una tabla es una colección de una o más columnas y cero o más filas. Puede entenderse como una estructura de datos simple que se asemeja a una matriz de dos niveles: el primero podría representar el número de fila, y el segundo, el número de columna. Para acceder a un valor de

LOS MODELOS MÁS UTILIZADOS

Si bien el modelo de datos más utilizado en la actualidad es el **relacional**, en determinados ámbitos de la programación está imponiéndose el modelo **orientado a objetos** (**POO**). Por otra parte, ambos toman cosas del otro y, aunque mantienen sus principios fundamentales, intentan dar a los usuarios lo que estos buscan.

esta matriz, debemos indicar alguno de esos datos. Como se dijo, una tabla nos permite organizar los datos en **filas** (también llamadas tuplas o registros) y **columnas** (atributos o campos). Cada una de las filas (los datos contenidos en ellas) se diferencia de las otras en algún dato almacenado. Cada columna dentro de una fila es el valor de un atributo de esa fila.

Continuando con nuestro ejemplo, podemos decir que **factura** es nuestra tabla, que, al ser diseñada y almacenada, la nombraremos como **entidad**. Veamos cómo quedaría la tabla factura en una vista de diseño:

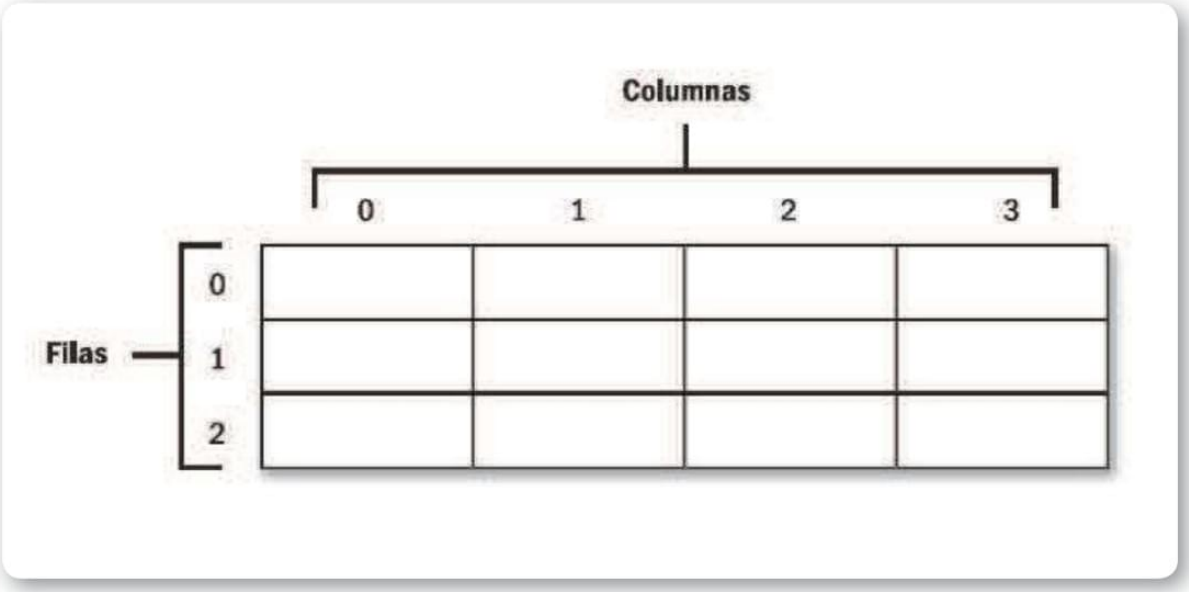


Figura 1. Estructura de una tabla.

Tabla Entidad: Factura		
Campo/Atributo	**Tipo de dato**	**Descripción**
Columna1	Texto	información relacionada con el contenido de datos.
Columna2	Número	
Columna3	Fecha	

Atributo

Un atributo define y diferencia una entidad de otra. Los atributos pueden verse como características de las entidades. Veamos un ejemplo: un perro puede tener un nombre, un dueño, un color de pelo, una edad, una comida preferida y demás.

Todos estos son atributos de un perro (entidad). Nótese que no definimos ningún valor particular para estas características, el atributo es el nombre del perro y no un perro. Veamos algunos atributos que podemos definir para la entidad **factura**.

Tabla Entidad: Factura

Campo/Atributo	Descripción
NumeroFactura	Los atributos son abstracciones de la vida real, información que nos será útil manipular y almacenar.
FechaVenta	
Categoría	
...	
Total	

Relación

Una relación es una vinculación entre ideas, un modo de asociar entidades/tablas para lograr el objetivo que tienen en común. Es decir que mediante esa vinculación es posible realizar una función.

De manera más concreta, es posible considerar que una relación es una asociación entre entidades. Por ejemplo, podemos decir que la entidad **factura** tiene una relación con la entidad **productos** (de donde se obtienen los detalles de lo que vamos a registrar).

Claves primarias

Para diferenciar una entidad de otra, debemos remitirnos a sus atributos y, para distinguir una fila de otra dentro de una tabla, nos guiamos por la clave primaria. Una clave primaria ofrece la particularidad de no tener valores repetidos. Para decidir cuál de todos los atributos formarán parte de la clave primaria, debemos buscar uno que cumpla con el requisito anterior, y, si no existe, deberemos crear una clave primaria **artificial**. Veamos ejemplos de cada situación.

En la tabla **libros**, tenemos la siguiente estructura.

Nombre
Cantidad_páginas
Editorial
ISBN
Autor

En este ejemplo, al repasar los atributos uno por uno, notaremos que solo hay uno que no va a repetirse, y es ISBN. Podemos tomarlo como clave primaria: no habrá dos filas con el mismo ISBN, no habrá dos libros con el mismo ISBN. Una clave primaria puede estar formada por más de un atributo, en este caso se dice que es una clave compuesta. ¿Podríamos tomar como clave primaria a ISBN y cualquier otro atributo, como por ejemplo **Editorial**? Sí, se podría, pero no se debería por la estabilidad de la información que almacenaremos.

Nada nos impide pasar por alto la clave ISBN y crear otra, que podría llamarse **codigo_libro**. En este caso **codigo_libro** no significa nada, no tiene ninguna implicancia para lo que es un libro: es solo un atributo con una restricción básica que será el hecho de no poder repetirse. Se trata de respetar ciertas técnicas de diseño de bases de datos y, a la vez, tener en cuenta lo que

nuestro sistema necesita; las decisiones finales corren por cuenta de las personas que se encarguen de ello. Veamos otro caso en el cual tenemos la tabla **alumnos**, cuya estructura inicial es la siguiente:

Nombre_alumno
Dirección
Curso
Colegio
Nacionalidad

En este ejemplo no hay ningún atributo que no pueda llegar a repetirse en los distintos alumnos. La solución es crear una clave y asignarla a cada alumno. Podríamos llamarla **codigo_alumno** y, al generarla, nos aseguramos de que no pueda tomar valores repetidos. Repasemos el ejemplo de la tabla **factura**:

Tabla Entidad: Factura	
Campo/Atributo	**Descripción**
Campo/Atributo	El campo NumeroFactura podría ser nuestra clave principal, ya que no debe repetirse el registro de una factura en nuestra base de datos, y una organización no puede generar facturas con la misma numeración.
NumeroFactura	
FechaVenta	
Categoría	
...	
Total	

Claves candidatas

Acabamos de ver que una fila es diferenciada de las demás a través de un número de atributos. Esos atributos forman lo que se llama una clave candidata. Puede haber varias claves candidatas dentro de una tabla.

La clave principal o primaria es la clave candidata que elijamos para representar cada fila de manera **unívoca**. Luego, nos permitirá identificar y localizar un registro de manera rápida.

> LA CLAVE PRINCIPAL NOS PERMITE IDENTIFICAR Y LOCALIZAR UN REGISTRO

Para identificar las claves candidatas de una tabla no hay que fijarse en un estado o instancia de la base de datos: el hecho de que en un momento dado no haya duplicados para un atributo o conjunto de atributos no garantiza que los duplicados no sean posibles.

Sin embargo, vale lo recíproco: la presencia de duplicados en un estado de la base de datos sí es útil para demostrar que cierta combinación de atributos no es una clave candidata válida.

Índices

Un índice es una estructura de datos en donde se almacena información adicional acerca de una columna (una columna indexada). Cuando se encuentra un índice en una columna, el gestor de base de datos lo usará para no recorrer todos los registros de una tabla.

Vistas

Existen ocasiones en las que no nos interesa que todos los usuarios de una base de datos puedan ver todos los datos de la base (ya sea por temas de seguridad, orden o, simplemente, para mostrar a cada usuario lo que estrictamente precise para realizar su trabajo). Para esto existen las llamadas **vistas**, que nos permiten **recortar** una tabla y restringir el acceso a ella. Por ejemplo, si tomamos en cuenta la siguiente tabla:

Tabla Empleados				
Cod_e	Nom_e	Ape_e	Sueldo_e	Dirección_e

Y tenemos un sector en nuestra empresa que se dedica a enviar bonificaciones por presentismo a los empleados, seguramente no se necesitará conocer el sueldo de cada uno de ellos, así que solo permitiremos que se vean los siguientes datos a través de una vista:

Vista Tarjeta_Empleados			
Cod_e	Nom_e	Ape_e	Dirección_e

Esquemas

Un esquema es una forma de visualizar una base de datos más allá de los datos que contenga actualmente. Para representar el esquema de una base de datos relacional se debe dar el nombre de sus tablas, los atributos de cada una de estas, los dominios sobre los que se definen estos atributos, las claves primarias y las claves foráneas (la definición y los ejemplos acerca de claves foráneas los encontraremos en el **capítulo 5**).

Tipos de relaciones entre tablas

Una base de datos está compuesta por tablas que se relacionan de alguna manera para darle un sentido al sistema y plasmar lo más fielmente posible lo que sucede en el mundo real.

Las relaciones pueden catalogarse según sus propiedades. En esta sección veremos dos de ellas: la **cardinalidad** y la **modalidad**.

La cardinalidad especifica el número de instancias de una entidad que se puede relacionar con un número de instancias de otra entidad. Quizás esto suene confuso, pero con la siguiente explicación veremos que no lo es tanto. Hay tres tipos de cardinalidad:

- Uno a uno
- Uno a varios
- Varios a varios

Cardinalidad uno a uno

Una relación de uno a uno entre dos tablas tiene lugar cuando, a cada elemento de la clave de la tabla A, se le asigna un único elemento de la tabla B y, para cada elemento de la clave de la tabla B, existe un único elemento en la tabla A. Por ejemplo, cada país tendrá solo un presidente, y cada presidente tendrá a su cargo solo un país.

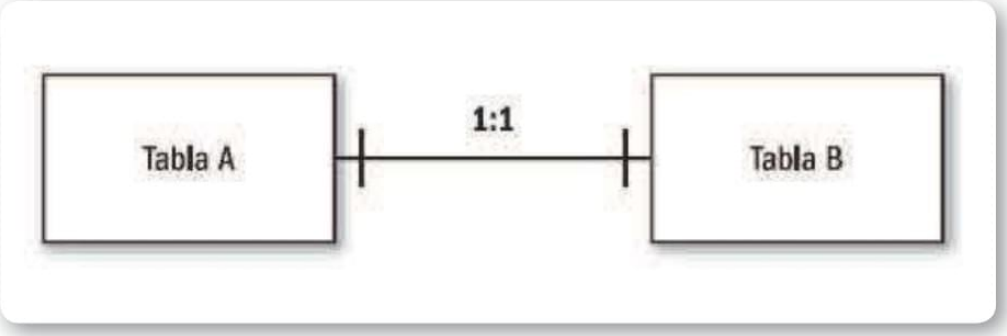


Figura 2. Relación uno a uno.

Llegado el caso, existe la posibilidad de unificar las tablas A y B en una sola, pero esto dependerá de las siguientes razones, que debemos tener cuenta:

- Las necesidades de nuestro sistema.
- Si en esa situación se respetarían o no las reglas de normalización (explicadas más adelante en este mismo capítulo).
- Si la tabla resultante es demasiado grande (gran cantidad de campos) y difícil de manejar, o si es adecuada para nuestro fin.

EN UNA BASE DE DATOS, LAS TABLAS SE RELACIONAN PARA DARLE SENTIDO AL SISTEMA

Cardinalidad uno a varios

Supongamos nuevamente que tenemos dos tablas: una llamada A y la otra B. Se dice que existe una relación de uno a varios entre las tablas A y B cuando una clave de la tabla A posee varios elementos relacionados en la tabla B y cuando una clave de la tabla B posee un único elemento relacionado en la tabla A. Por ejemplo, cada ciudadano vota por un candidato, y cada candidato tendrá la posibilidad de ser votado por más de un ciudadano.

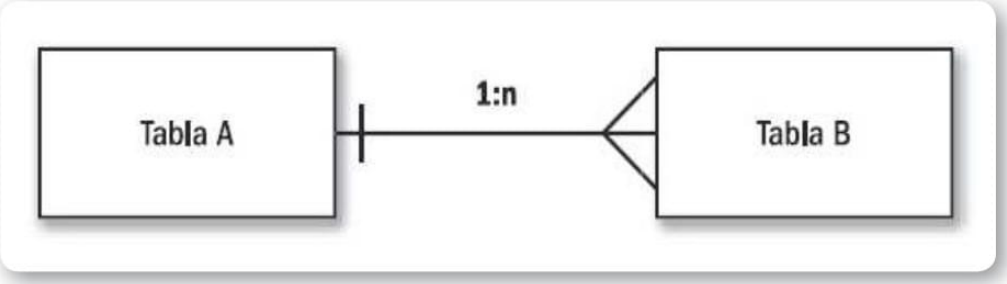


Figura 3. Relación uno a varios.

Cardinalidad varios a varios

Una relación es de varios a varios entre las tablas A y B cuando una clave de la tabla A posee varios elementos relacionados en la tabla B y, a su vez, una clave de la tabla B posee varios elementos relacionados en la tabla A. Por ejemplo, un libro pudo haber sido escrito por varios autores, y cada autor pudo haber escrito varios libros.

VERSIONES DE MYSQL

Si bien es recomendable utilizar la última versión de cualquier aplicación, hay en ocasiones motivos para no hacerlo. Uno de ellos es que las empresas que desarrollan software suelen liberar las versiones BETA, que se consideran de prueba y no son recomendables para sistemas en etapa de producción, pero, tal vez, sí para los que estén en desarrollo.

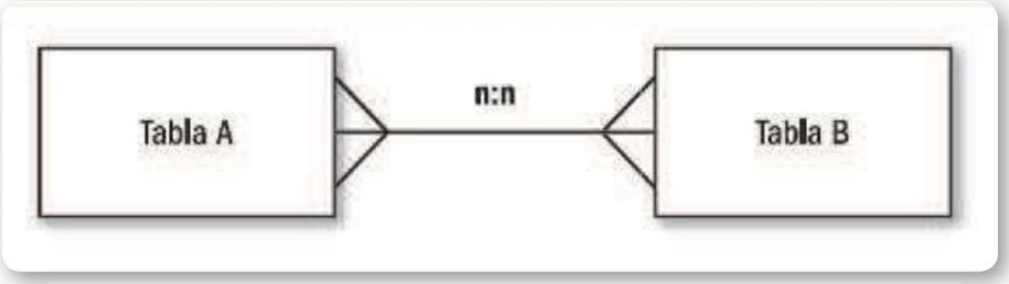


Figura 4. Relación varios a varios.

Por lo general, no veremos este tipo de relaciones en los diseños de los sistemas de bases de datos, porque son difíciles de tratar y de mantener. Directamente se transforman en dos relaciones de tipo uno a varios. Esto se hace creando una tercera tabla (llamémosle C) que hace de intermediaria entre la A y la B, que permanecen intactas.

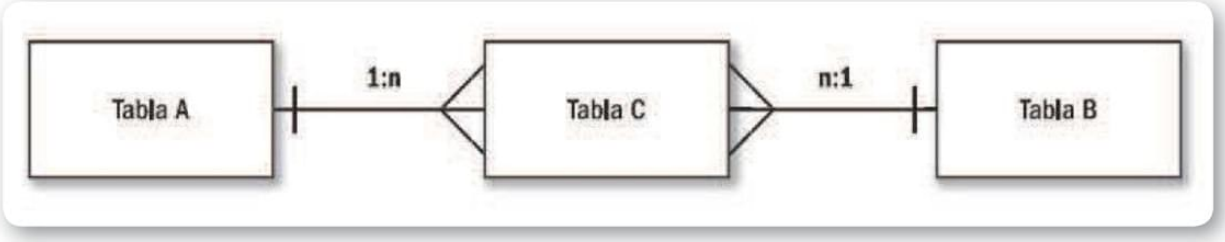


Figura 5. Transformación de relaciones varios a varios.

A continuación, veamos un ejemplo de esto último (identificamos con # al campo que actúa como clave primaria):

PONGA EN PRÁCTICA SUS CONOCIMIENTOS

En internet circulan, frecuentemente, gran cantidad de ejercicios –en general, de universidades– que pueden servirnos de ayuda a la hora de poner en práctica nuestros conocimientos acerca del diseño de sistemas. Recordemos que la mejor manera de afianzar conocimientos y adquirir otros es enfrentándose a situaciones y desafíos nuevos.

Tabla Libro	
#	Cod_libro
	Título
	Páginas

Tabla Autor	
#	Cod_autor
	Nombre y Apellido
	Nacionalidad

Hasta aquí tenemos una relación de varios a varios. Para transformar esta relación hacemos lo siguiente: tomamos las claves de ambas tablas y construimos una tercera. Luego armamos las relaciones.

Tabla Autor_libro	
#	Cod_autor
#	Cod_libro

Las relaciones quedarían como se muestra en la **figura 6**.

Figura 6. Transformación de relaciones varios a varios.

Modalidad

La modalidad indica si es obligatorio o no que una instancia participe en la relación. Vimos antes que, según la cardinalidad, existen relaciones del tipo uno a varios, uno a uno y varios a varios. Lo que la modalidad agrega a este tipo de relaciones es definir si es obligatorio o no la participación de cada instancia en esa relación. Volviendo a los ejemplos anteriores:

cada ciudadano vota por un candidato, y cada candidato tendrá la posibilidad de ser votado por más de un ciudadano. Ahora bien, ¿es obligatorio que cada ciudadano vote por un candidato? ¿Y que un candidato sea votado por una persona al menos? Respondiendo esta clase de preguntas podremos determinar la modalidad en las relaciones.

La forma gráfica de definir la modalidad en las distintas relaciones depende del autor, no hay solo una manera, pero la forma clásica o la más usada es la que vemos en la **figura 7**.

Figura 7. Simbología de modalidad en las relaciones.

Veamos algunos ejemplos. En la **figura 8**, todos los registros de la tabla B deberán tener un (y solo uno) registro asociado en la tabla A. Por su lado, los registros de la tabla A podrán tener (o no) uno o varios registros asociados en la tabla B.

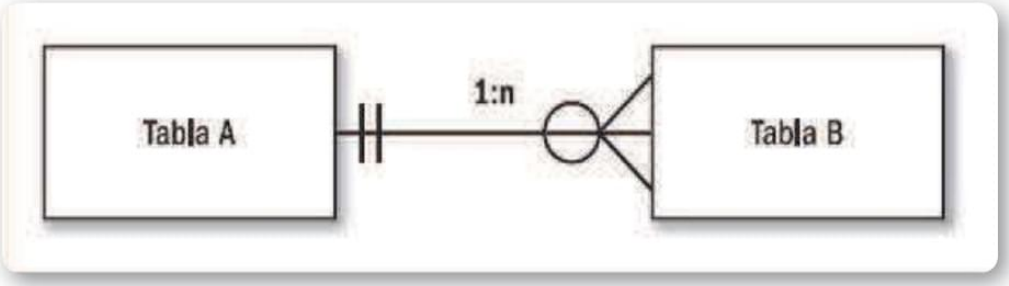


Figura 8. Ejemplo de modalidad en las relaciones.

Al finalizar la lectura de los párrafos anteriores, conocemos cómo debemos confeccionar una base de datos; llevémoslo a la práctica conociendo qué gestor de base de datos podemos utilizar.

Gestor de base de datos: MySQL

Disponemos de varias herramientas para la gestión de base de datos, para este libro elegimos una muy popular y con mucho futuro por delante que es: **MySQL**. Analizaremos cuáles son las razones para que se haya convertido en el complemento ideal de PHP.

Por qué MySQL

MySQL es un sistema gestor de bases de datos muy utilizado en la actualidad por los siguientes motivos:

- Rapidez
- Posibilidad de trabajar en diferentes plataformas
- Múltiples formatos de tablas para cada necesidad
- Seguridad
- Gran estabilidad
- Administración simple
- Soporte técnico (con el licenciamiento comercial)

Si bien todavía le queda mucho camino por recorrer, en corto tiempo ha logrado darse a conocer en el ámbito informático y fue afianzándose progresivamente en el mundo de las bases de datos relacionales.

Desde hace algún tiempo, se ha ido dando una particular unión entre esta base de datos y el lenguaje de programación PHP; por este motivo, MySQL se utiliza principalmente en sitios web.

Obtener MySQL

MySQL puede obtenerse de manera completamente libre a través de cualquier tipo de distribución: revistas, Internet, copias en CD/DVD provistas por amigos, etcétera. Pero, sin dudas, una de las formas más utilizadas es acceder al sitio **www.mysql.com**, y desde allí descargar la

versión más adecuada al sistema sobre el cual desarrollaremos nuestras aplicaciones. Esta es una buena forma de tener acceso a la última versión del programa. MySQL está disponible, entre otros, para los siguientes sistemas operativos:

- Linux
- Windows (9x, Me, NT, 2000, XP, Vista, 7)
- Solaris
- BSD (FreeBSD, NetBSD, OpenBSD, BSD/OS)
- Mac OS
- Novell NetWare (6.0 o superior)
- OS/2
- BeOS
- RISC OS
- SGI IRIX 6.5.x
- AS/400

Licencia de uso

MySQL pone a disposición de los usuarios dos tipos de licenciamiento: una licencia comercial y una licencia GPL (**General Public License**). La licencia comercial brinda, entre otras cosas, soporte técnico y garantía.

Diseñar una base de datos es una tarea que necesariamente combina experiencia práctica y conocimientos teóricos. En este capítulo hemos recorrido los conceptos fundamentales para introducirnos en este apasionante tema y poder sacar provecho de las opciones que nos ofrecerá MySQL.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuáles son las ventajas de trabajar con bases de datos?
2. ¿Qué es una tabla? Dé tres ejemplos.
3. ¿A qué se denomina atributo? Defina posibles atributos para las tablas de la pregunta anterior.
4. ¿Qué es una clave primaria? Defina posibles claves primarias para las tablas de la pregunta 3. ¿Con qué criterio los elegirá?

EJERCICIOS PRÁCTICOS

1. Nombre dos casos en los cuales aplicaría vistas.
2. Describa un ejemplo en donde intervengan relaciones uno a uno.
3. Describa un ejemplo en donde intervengan relaciones uno a varios.
4. Cree el diseño de una base de datos en donde intervengan relaciones uno a varios e implemente un esquema de integridad referencial como se hizo en este capítulo.
5. Determine la relación entre las tablas de ejemplo **factura** y **producto**.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: **profesor@redusers.com**.

*03

MySQL

En este capítulo, para poner en práctica lo aprendido, trabajaremos con MySQL. Utilizaremos sus características generales; para esto debemos saber que la creación de tablas es un proceso posterior al diseño de la base de datos, requiere atención, análisis y un conocimiento profundo.

- Tipos de tablas 44
- Tipos de datos 50
- Referencia de funciones 59
- El uso del monitor de MySQL ... 98
- Resumen 101
- Actividades 102

Tipos de tablas

Cuando trabajamos con MySQL, existe la posibilidad de variar el tipo de tabla una vez creada, salvo en el caso de las tablas del sistema –llamadas MySQL y test–, que por defecto son MyISAM, y no se recomienda modificarlas. Para indicar el tipo de tabla al crearla, usamos la siguiente sintaxis:

```
CREATE TABLE tabla1 (
   campo1 INT(4),
) ENGINE=MYISAM;
```

Si intentamos crear un tipo de tabla no disponible en la versión que tenemos de MySQL, se generará una tabla tipo MyISAM. A continuación, haremos un recorrido por los distintos tipos de tablas soportados.

ISAM

MySQL empezó utilizando este tipo de tablas y, actualmente, se las considera en desuso. Entre sus desventajas figura el hecho de no poder transportar ficheros entre máquinas con distinta arquitectura (tiene un formato distinto para cada arquitectura/sistema operativo, lo cual resulta más rápido, pero presenta el problema de la incompatibilidad) y el de no manejar ficheros de tablas superiores a 4 gigabytes. Los índices se guardan en archivos **.ISM** y los datos en archivos **.ISD**. MySQL recomienda actualizar este tipo de tablas hacia las de tipo MyISAM. Esto puede hacerse con la siguiente instrucción SQL:

```
ALTER TABLE nombre_de_la_tabla ENGINE = MYISAM;
```

MyISAM

Es el tipo de tabla por defecto en MySQL desde la versión 3.23 y está basado en las ISAM, por supuesto, que ofrecen más opciones.

Proveen un almacenamiento independiente: se pueden copiar tablas de una máquina a otra de distintas plataformas. Cuando se usan estas tablas, los datos se almacenan físicamente en dos archivos: uno que tiene la extensión **.MYI** (**MYISAM INDEX**), para los índices, y otro **.MYD** (**MYISAM DATA**), para los datos.

Si estamos trabajando con MySQL en forma local, normalmente, se podrá ver en el propio equipo que estos archivos se encuentran en una carpeta que tiene por nombre a una base de datos (estas carpetas están en el directorio **DATA** dentro del directorio en donde se instaló MySQL, y allí hay una carpeta por cada base de datos). Esto vale también para otros tipos de tablas.

```
c:\wamp\bin\mysql\mysql5.5.8\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.5.8-log MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> use ejemplos;
Database changed
mysql> create table tabla (
    -> campo1 int(4),
    -> campo2 varchar(100)
    -> ) engine=myisam;
Query OK, 0 rows affected (0.05 sec)

mysql> _
```

Figura 1. Creación de una tabla **MyISAM** desde el monitor de **MySQL**.

Entre sus características, destacamos las siguientes:

- Soportan archivos de gran tamaño (63 bits, ficheros de tablas superiores a 4 gigas) en comparación con los que soportaban las ISAM.
- Optimizadas para sistemas operativos de 64 bits.
- Posibilidad de indexar campos **BLOB** y **TEXT**.
- Se permiten valores **NULL** en columnas indexadas.
- Inserts concurrentes (pueden insertarse varios registros al mismo tiempo).
- Cada tabla guarda un registro que indica si fue cerrada correctamente o no, y al iniciar MySQL existe la opción de indicarle que se verifique ese registro, y se repare la tabla de ser necesario de forma automática.

Esto se logra iniciando MySQL con la opción:

```
--myisam-recover
```

Si se encontró un error, trata de repararlo de forma rápida ordenando los registros de la tabla. Si persiste el error, se vuelve a crear el archivo que contiene la tabla, y, si continúa sin corrección, se intenta repararlo escribiendo los registros sin un ordenamiento.

MERGE

Este tipo de tabla es muy utilizada en los casos en que se precisa tratar a un número N de tablas MyISAM (de idéntica estructura y pertenecientes a la misma base de datos, de la que también deberá formar parte la tabla MERGE) como si fuera una sola. Esto podría aplicarse si la tabla MyISAM original fuera de gran tamaño y acceder a su contenido llevara una cantidad considerable de tiempo y recursos. Obviamente, estamos hablando de una tabla muy grande. Entre otras características, destacamos:

- Solo se pueden aplicar instrucciones **SELECT**, **DELETE** y **UPDATE**.
- La definición de la tabla se almacena en un archivo **.FRM**, y el listado de las tablas MySAM en un archivo **.MRG** (aquí hay en realidad un índice de los archivos **.MYI** usados. Ver tablas MyISAM).
- Permite de algún modo burlar el tamaño máximo de una tabla y el tamaño máximo de un archivo en un sistema operativo específico.
- Si quisiéramos borrar una de las tablas MyISAM que forma parte de la tabla **MERGE**, no podríamos hacerlo bajo el sistema operativo Windows, ya que este no permite borrar archivos que estén abiertos, y la tabla al formar parte de la tabla **MERGE** se considera abierta.
- Por la misma razón se producen inconvenientes al aplicar instrucciones tales como **DROP TABLE**, **ALTER TABLE**, **DELETE FROM** sin **WHERE**,

REPAIR TABLE, **TRUNCATE TABLE**, **OPTIMIZE TABLE** y **ANALYZE TABLE**.

Una manera de solucionar esto es borrar el contenido de la tabla **MERGE** (aplicando la instrucción **DELETE** sin **WHERE** a una tabla **MERGE**, no se borra el contenido de las tablas MyISAM, sino que se las quita del listado de componentes de la tabla **MERGE**). Así, estas tablas se considerarán cerradas. Cuando se crea una tabla de este tipo, hay que especificar (con la instrucción **UNION**) la lista de tablas asociadas.

Devolvería algo similar a la tabla siguiente:

a	mensaje
1	uno
2	dos
3	tres
1	cuatro
2	cinco
3	seis

El parámetro **INSERT_METHOD** especifica en qué tabla se realizarán los **INSERTS**, si en la primera de la lista (poniendo **INSERT_METHOD=FIRST**) o en la última (poniendo **INSERT_METHOD=LAST**). En nuestro caso, la tabla **t1** es la primera, y **t2** es la última.

HEAP

Este tipo de tablas tienen una particularidad que las distingue del resto: son tablas en memoria, temporales y desaparecen cuando el servidor se cierra. Esto las hace realmente rápidas y, a diferencia de una tabla TEMPORARY, que solo puede ser accedida por el usuario que la crea, pueden ser utilizadas por diversos usuarios.

Algunas de sus características son:

- No soportan columnas de tipo **BLOB** o **TEXT**.
- No soportan columnas de tipo **AUTO_INCREMENT**.
- No se permiten valores **NULL** en columnas indexadas (antes de MySQL versión 4.0.2).
- Siempre conviene especificar el número máximo de filas (**MAX_ROWS**) cuando se crea la tabla, para no usar toda la memoria disponible.

InnoDB

Estas tablas, al igual que las Berkeley DB, son **TST** (*Transactions Safe Tables* o tablas para transacciones seguras). Las tablas de este tipo son más lentas y ocupan más memoria, pero a cambio ofrecen mayor seguridad frente a fallas durante la consulta. Fueron agregadas en la versión 4.0 de MySQL y tienen las siguientes características:

- Proveen la posibilidad de transacciones seguras, **ACID** (**autenticación**, **confidencialidad**, **integridad**, y **disponibilidad**).
  - **Atomicidad**: consultas tratadas como una sola, de tal forma que solo se ejecutan cuando todas ellas tienen éxito. En caso de que alguna falle, no se ejecuta ninguna.
  - **Consistencia**: solo datos válidos pueden ser escritos en la base de datos.
  - **Separación**: las transacciones que tengan lugar simultáneamente deben ejecutarse aisladas unas de otras hasta que finalizan.
  - **Durabilidad**: cuando una transacción se completa con éxito, los cambios son permanentes y no se podrá volver atrás.
  - Soporta operaciones **COMMIT** y **ROLLBACK** (beneficio propio de ser **TST**).

- Recuperación ante fallas.
- Soporta **FOREIGN KEY** (claves foráneas). Es la primera vez que esto se da en MySQL.
- Bloqueo a nivel de fila.

- Permite realizar copias de seguridad mientras la base está funcionando.
- Alta eficacia en el procesamiento de grandes volúmenes de información.
- No permite crear claves sobre columnas de tipo **BLOB** o **TEXT**.
- Una tabla no puede tener más de 1000 columnas.
- Al borrar todas las filas de una tabla lo hace una por una, lo que produce problemas relacionados con la velocidad. Hasta ahora puede truncar tablas.

BerkeleyDB

Estas tablas pueden ser usadas independientemente de MySQL; son desarrolladas por otra empresa, **Sleepycat**, y MySQL ofrece una interfaz para trabajar con ellas como una posibilidad más.

- Soportan operaciones **COMMIT** y **ROLLBACK**.
- Es de tipo TST (*Transactions Safe Tables*). (Ver tablas INNODB para más información acerca de esto).
- Normalmente, para instalarlas hay que buscar una versión de MySQL que incluya soporte para este tipo de tablas y habilitar la opción en el momento de la instalación (****with*berkeley*dboption**).
- En el archivo en donde se almacenan los datos se guarda la ruta a ese mismo archivo, de modo que no es posible cambiar de directorio la base.

SELECCIÓN DE TIPOS DE DATOS

En el momento de definir una tabla, debemos prestar particular atención al tipo de dato que destinamos a cada campo y a su longitud: esto nos ayudará a preservar recursos y a mantener una coherencia dentro del diseño de las estructuras de datos del sistema. Encontraremos más información acerca del diseño de bases de datos relacionales en el **capítulo 2** de este libro.

En qué casos usar cada tabla

Como siempre, la respuesta depende de lo que tengamos que hacer.

Las tablas que normalmente se usan hoy en día son las MyISAM, pero pronto (quizás muy pronto) se comenzarán a usar las INNODB, especialmente por la posibilidad de crear relaciones entre tablas (fundamental en el modelo relacional) y ofrecer mayores prestaciones con respecto a la seguridad, además de las transacciones.

Las ISAM están prácticamente en desuso (incluso la empresa que desarrolla MySQL admite la posibilidad de que en su versión 5 ya no estén disponibles), y las demás tienen usos muy específicos e incluso compatibles con otros tipos. La clave está en estudiar los problemas que necesitamos solucionar, y ver en cada caso qué es lo que más conviene.

Tipos de datos

En el momento de crear tablas de bases de datos, es muy importante definir los tipos de datos. MySQL soporta una gran variedad de ellos, uno para cada necesidad.

Cadenas de caracteres

Los subtipos de datos existentes aquí son **CHAR**, **VARCHAR**, **BLOB**, **TEXT**, **ENUM**, y **SET**.

CHAR y VARCHAR

Son muy similares, quizás la diferencia más notable es el modo de almacenamiento. Cuando definimos una columna tipo **CHAR** de tamaño **N**, e ingresamos un valor (de menos de **N** caracteres) en esa columna, MySQL rellenará con espacios lo que sobra, mientras que si hacemos lo mismo con una columna de tipo **VARCHAR**, no se rellenará con espacios. Al obtener información a través de una consulta SQL, no aparecen los espacios sobrantes, MySQL

los remueve. Si se ingresa una cadena de mayor cantidad de caracteres que el tamaño prefijado al definir la columna, la cadena se truncará al llegar al límite. Por defecto, las comparaciones (en cualquiera de los dos tipos de datos) son insensibles a mayúsculas y minúsculas.

```
CREATE TABLE tabla1 (
      campo1 INT(4),
      campo2 VARCHAR(25) NOT NULL
)
```

BLOB y TEXT

Se utilizan para cadenas con un rango que dependerá del tamaño que queramos almacenar. La diferencia entre ambos es que **TEXT** permite comparar dentro de su contenido sin distinguir mayúsculas y minúsculas, y **BLOB** las distingue. Además, **TEXT** tiende a ser usado para cadenas de texto plano (sin formato), mientras que **BLOB** se usa para objetos binarios, o sea, cualquier tipo de datos o información, desde un archivo de texto con todo su formato, incluyendo las imágenes, hasta archivos de sonido o video.

BLOB (*Binary Large OBject*, objeto binario de gran tamaño) se subdivide en cuatro tipos que difieren solo en la capacidad máxima de almacenamiento. Con **TEXT** sucede lo mismo e, incluso, hay correspondencia entre la capacidad máxima de almacenamiento de unos y otros.

AUSTERIDAD EN SQL

Cuando realizamos una consulta a un servidor de bases de datos, este nos devuelve resultados que viajan a través de la red. Mientras más datos sean, más tráfico generaremos; de ahí la importancia de construir nuestras instrucciones SQL con eficacia, de manera de obtener estrictamente solo lo que necesitamos.

SUBTIPOS DE DATOS BLOB Y TEXT		
▼ DIVISIONES DE BLOB	▼ DIVISIONES DE TEXT	▼ LÍMITE EN CARACTERES
TINYBLOB	TINYTEXT	255
BLOB	TEXT	65.535
MEDIUMBLOB	MEDIUMTEXT	16.777.215
LONGBLOB	LONGTEXT	4.294.967.295

Tabla 1. Equivalencias de tamaños entre variables de tipo **BLOB** y **TEXT**.

Ninguna columna de estos tipos puede tener valor por defecto.

ENUM

Este tipo de string puede seleccionar su valor únicamente de una lista finita (máximo de 65.535 elementos) de opciones definidas por el usuario y otras dos por defecto (índice **0** que significa error, por ingresos fuera de rango, por ejemplo, y está representado por **“”**, e índice **NULL** con valor **NULL**). Por ejemplo, definimos una columna con **ENUM(“argentina”,“mexico”,“paraguay”)**:

```
CREATE TABLE user (
  nombre enum('juan','pedro') NOT NULL,
  pais enum('argentina','paraguay', 'mexico'),
);
```

SET

Similar a **ENUM** en su funcionamiento, solo que se puede seleccionar ninguno o más de un valor de la lista (hasta 64, la muestra separada por comas).

Numéricos

Existen los subtipos **DECIMAL** (**NUMERIC** o **DEC**), **INTEGER** (**INT**), **TINYINT**, **BIT**, **BOOL**, **MEDIUMINT**, **BIGINT**, **SMALLINT**, **FLOAT** y **DOUBLE** (**DOUBLE PRECISION** o **REAL**). Todos los tipos numéricos pueden definirse con dos parámetros opcionales: **UNSIGNED** (impide que los campos numéricos acepten signo negativo, es decir, solo se aceptarán el cero y los valores positivos) y **ZEROFILL** (completa con ceros a la izquierda hasta la longitud máxima). La forma de uso es:

```
TIPO_DATO [UNSIGNED] [ZEROFILL]
```

Por ejemplo,

```
INT(4) UNSIGNED ZEROFILL
```

Si tenemos almacenado el número **12**, al hacer un **SELECT** se mostrará **0012**. Si tenemos almacenado el número **-12**, al hacer un **SELECT** se mostrará **0000**. A continuación, definiremos cada uno de los subtipos numéricos. En todos los casos, **M** es el número de dígitos que serán visibles al mostrar el contenido del campo. Si el campo que se va a mostrar sobrepasa los **M** dígitos, se mostrarán solo **M** dígitos.

TINYINT[(M)], BIT

Un tipo de datos BIT nos es útil para almacenar valores de un bit. Si se omite o se sobrepasa la capacidad de **TINYINT**, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo, va desde **-128** a **127**, en cambio, sin signo (**UNSIGNED**), va desde **0** hasta **255**.

BIT es un **TINYINT** de un dígito **(TINYINT(1))**.

SMALLINT[(M)]

Este es un tipo numérico exacto que podemos utilizar. Si se omite o se sobrepasa la capacidad de **SMALLINT**, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo, va desde **-32768** a **32767**, en cambio, sin signo (**UNSIGNED**), va desde **0** hasta **65535**.

MEDIUMINT[(M)]

Es otro tipo numérico exacto que podemos utilizar, de mayor capacidad al anterior. Si se omite o se sobrepasa la capacidad de **MEDIUMINT**, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo va desde **-8388608** a **8388607**, en cambio, sin signo (**UNSIGNED**), va desde **0** hasta **16777215**.

INT[(M)] e INTEGER[(M)]

Este tipo de dato es el utilizado con mayor frecuencia en los lenguajes de programación y se destina para albergar un valor entero bastante grande. Si se omite o se sobrepasa la capacidad de **INT**, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo va desde el valor **-2147483648** al valor **2147483647**, en cambio, cuando carece de signo (**UNSIGNED**), va desde **0** hasta **4294967295**. **INTEGER[(M)]** es sinónimo del tipo de dato **INT**.

BIGINT[(M)]

Es uno de los tipos que soportan un valor entero realmente enorme para transacciones de cálculos científicos o valores muy grandes.
Si se omite o se sobrepasa la capacidad de **BIGINT**, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo va desde **-9223372036854775808** a **9223372036854775807**, en cambio, sin signo (**UNSIGNED**) va desde **0** hasta **18446744073709551615**. Todas las funciones matemáticas trabajan internamente con valores **BIGINT**.

FLOAT[(M,D)]

Sirven para definir números con coma, con menos precisión que **DOUBLE**. El rango de posibles valores va desde **-3.402823466E+38** a **-1.175494351E-38**,y**0** y desde **1.175494351E-38** a **3.402823466E+38**.

DOUBLE[(M,D)], DOUBLE PRECISION[(M,D)] y REAL[(M,D)]

Sirven para definir números con coma, con más precisión que **FLOAT**. El rango de posibles valores va desde **-1.7976931348623157E+308** a **-2.2250738585072014E-308**,**0** y desde **2.2250738585072014E-308** a **1.7976931348623157E+308**. **DOUBLE PRECISION[(M,D)]** y **REAL[(M,D)]** son sinónimos de **DOUBLE**.

DECIMAL[(M[,D])], DEC[(M[,D])] y NUMERIC[(M[,D])]

Aquí el número es almacenado internamente como una cadena de caracteres (un carácter por cada dígito). Ni el separador decimal (**,**) ni el signo menos (**-**) para números negativos son parte de **M**. Si no se le da ningún argumento, por defecto, **M** es igual a **10**, tomando un rango de **-9999999999** hasta **99999999999** para números con signo, y **D** es igual a **0**. **DEC[(M[,D])]** y **NUMERIC[(M[,D])]** son sinónimos de **DECIMAL**.

ANSI SQL

Es un estándar que permite normalizar las consultas realizadas sobre una base de datos. En general, posibilita el manejo de los datos (inserción, actualización, eliminar, etc.) de una base de datos. La mayoría de motores de bases de datos lo implementan: SQL server, Oracle, Sybase, Postgres, entre otros.

VARIABLES DE TIPO NUMÉRICO			
▼ TIPO	▼ BYTES	▼ VALOR MÍNIMO (CON SIGNO/SIN SIGNO)	▼ VALOR MÁXIMO (CON SIGNO/SIN SIGNO)
TINYINT	1 0	-128 255	127
SMALLINT	2	-32768 0	32767 65535
MEDIUMINT	3	-8388608 0	8388607 16777215
INT	4	-2147483648 0	2147483647 4294967295
BIGINT 01844674407 3709551615	8 01844674407 3709551615	-9223372036854775808 018446744073709551615	9223372036854775807 018446744073709551615

Tabla 2. Equivalencias de tamaños entre variables de tipo numérico.

Fecha y hora

Los subtipos de datos existentes son **DATETIME**, **DATE**, **TIMESTAMP**, **TIME** y **YEAR**.

DATETIME

Se utiliza para trabajar con fechas y horarios a la vez. El formato por defecto es **'YYYY-MM-DD HH:MM:SS'**, pero se le puede dar otro formato si por algún motivo necesitáramos hacerlo, por ejemplo: **'YYYY/MM/DD HH%MM%SS'**. El rango va desde **'1000-01-01 00:00:00'** a **'9999-12-31 23:59:59'**. Si en el momento de ingresar un **DATETIME** definimos un valor inválido (por ejemplo, minuto superior a **60**), se almacenará la fecha nula; para el formato **'YYYY-MM-DD HH:MM:SS'** sería **'0000-00-00 00:00:00'**

DATE

Se utiliza para trabajar solo con fechas. El formato por defecto es **'YYYY-MM-DD'**, pero se le puede dar otro formato si por algún motivo necesitáramos hacerlo, por ejemplo: **'YYYY*MM*DD'**. El rango va desde **'1000-01-01'** a **'9999-12-31'**. Si en el momento de ingresar un **DATE** definimos un valor inválido (por ejemplo, mes superior a **12**), se almacenará la fecha nula; para el formato **'YYYY-MM-DD'** sería **'0000-00-00'**.

TIMESTAMP

Combinación de fecha y hora. El rango va desde el **01-enero-1970** al año **2037**. El formato de almacenamiento depende del tamaño del campo y se visualiza como un número.

FORMATO DE ALMACENAMIENTO DE FECHA Y HORA

▼ TAMAÑO	▼ FORMATO
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

Tabla 3. Formatos de la función **TIMESTAMP**.

Si en el momento de crear la tabla se define un **TIMESTAMP** mayor a **14**, se redondea a **14**; si se define un número impar, se redondea al par

inmediatamente superior. Si definimos un valor inválido (por ejemplo, minuto superior a **60**) se almacenará la fecha nula. Por ejemplo:

Para **TIMESTAMP(2)** sería **00**.
Para **TIMESTAMP(12)** sería **000000000000**.

Otro punto importante es que, al ingresar una fecha o un horario, se toman los datos ausentes como ceros. Esto supone un problema para las fechas, ya que si tenemos un **TIMESTAMP(2)**, no podemos ingresar solo el año porque eso supondría algo como **990000** (día y mes no pueden ser cero, estarían fuera de rango). En cambio la hora, los minutos y los segundos sí pueden ser cero. Es decir que no podemos insertar cadenas de menos de 6 caracteres. En PHP hay una gran cantidad de funciones que precisan trabajar con el formato **TIMESTAMP**.

TIME

Se utiliza para trabajar solo con horarios. El formato por defecto es **'HH:MM:SS'** (aunque también soporta **'HHH:MM:SS'** para períodos largos de tiempo), pero se le puede dar otro formato si por algún motivo necesitáramos hacerlo, por ejemplo: **'HH*MM*SS'**. El rango va desde **'-838:59:59'** a **'838:59:59'** (el hecho de poder almacenar **TIME** negativos nos da la pauta de que existen más usos que el de simplemente almacenar el horario de un determinado suceso). Si en el momento de ingresar un **TIME**, definimos un valor inválido (por ejemplo, minuto superior a **60**) se almacenará la fecha nula, para el formato **'HH:MM:DD'** sería **'00:00:00'**. Si ingresamos valores fuera de rango, estos valores se reemplazan por el extremo más cercano.

YEAR

Se usa para representar años. Su formato es por defecto **'YYYY'** (puede definirse como **'YY'**). El rango va desde **1901** hasta **2155**. Si se representa el año con solo dos dígitos, surge la siguiente particularidad: si se define

el campo tipo **YEAR** como un número, no podemos representar el año **2000** con **00** (sería interpretado como el año **0000**), debemos hacerlo con la cadena **00** o con **0**. Si ingresamos un valor inválido, este será convertido a **0000**.

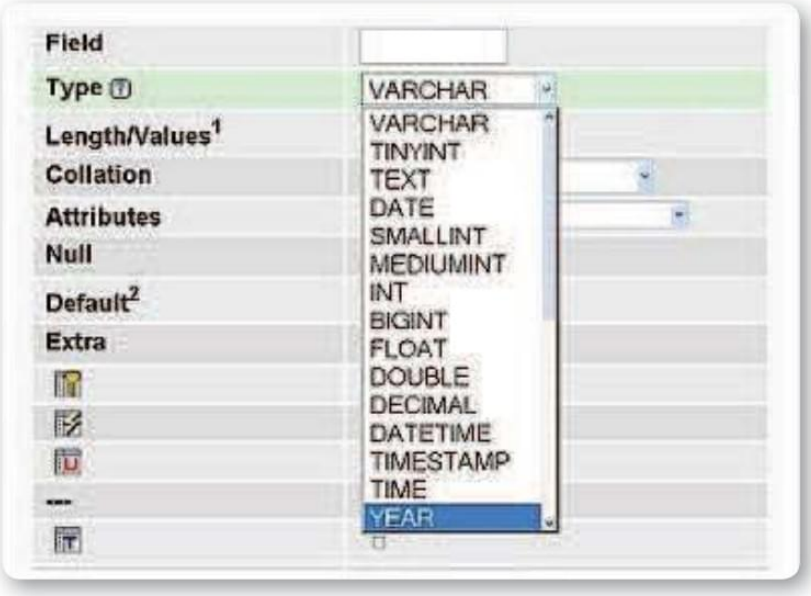


Figura 2. Gestor de tipos de datos para un atributo de una tabla en MySQL.

Referencia de funciones

Los sistemas gestores de bases de datos, normalmente, vienen con funciones incorporadas para mejorar y hacer más sencillas las consultas que se realizan a una base de datos. La mayoría de estas funciones forman parte del lenguaje SQL estándar, pero hay otras que son específicas de algún gestor de bases de datos, lo que significa que existen grandes posibilidades de que, al migrar de un sistema gestor de bases de datos a otro, estas funciones no sean reconocidas, es decir, no sean compatibles.

Una de las cosas que hacen popular a una base de datos, dentro del ámbito informático, es el hecho de que respeten el estándar **ANSI SQL**, pero eso no se contradice con que cada base mejore las prestaciones e incorpore funciones propias: queda a criterio de los usuarios utilizar

estas funciones y conocer las virtudes y defectos de su uso. MySQL no es la excepción y nos brinda una gran variedad de funciones para hacernos más fácil recuperar o manipular datos de una base.

Cabe destacar que, en los ejemplos que daremos al referenciar estas funciones, los parámetros (o argumentos) son genéricos, es decir, se les da un nombre cualquiera para que resulte más sencillo entender qué hace la función. Pero, para sacarles partido a las funciones, los parámetros deben ser otras funciones, o bien, campos de las tablas referenciadas en la instrucción SQL. Los argumentos entre corchetes (**[arg]**) son opcionales. A continuación, veremos cada una de las funciones ordenadas alfabéticamente.

SELECT	Consulta de datos
INSERT **UPDATE** **DELETE**	Lenguaje de manipulación de datos (**DML**)
CREATE **ALTER** **DROP** **RENAME**	Lenguaje de definición de datos (**DDL**)
COMMIT **ROLLBACK**	Control de transacciones
GRANT **REVOKE**	Lenguaje de control de datos (**DCL**)

Figura 3. Ejemplos básicos de sentencias SQL ANSI.

Funciones para trabajar con cadenas de caracteres

Estas funciones normalmente trabajan con posiciones dentro de una cadena de caracteres: por defecto, la primera posición (el primer carácter) está asociada al número **1**; la segunda, al **2**, y así sucesivamente.

ASCII(str)

Devuelve el valor del código ASCII del primer carácter del parámetro. Por ejemplo, la consulta:

```
SELECT ASCII('2');
//Devuelve 50 que es el código ASCII que representa al carácter '2'.
```

Esta función devuelve **0** si **str** es una cadena vacía y **NULL** si es nula.

BINARY

Es en realidad un operador y puede utilizarse para forzar comparaciones sensibles a mayúsculas y minúsculas. Por ejemplo:

```
SELECT "a" = "A";
//Devuelve 1 (verdadero)

SELECT BINARY "a" = "A";
//Devuelve 0 (falso)
```

NÚMEROS Y CADENAS DE CARACTERES

Es una buena costumbre definir como numéricos solo aquellos campos sobre los cuales se realizarán operaciones matemáticas, como el número de productos de una empresa. Pero en el caso de, por ejemplo, números de documento o códigos postales, esto no sería necesario, recordemos que estas decisiones dependerán particularmente del sistema.

BIT_LENGTH(str)

Devuelve la longitud de la cadena **str** en bits.

CHAR(N1, N2, N3, ...)

Recibe como parámetros números enteros (si recibe números con coma o cadenas que contienen números, solo considera los enteros) y los toma como valores del código ASCII, devolviendo los símbolos correspondientes. Por ejemplo:

```
SELECT CHAR(77,121.4,83,81,'76');
//Devuelve 'MySQL'
```

77 representa el carácter **M** según el código ASCII, el **121** representa el carácter **y**, y así sucesivamente.

CHAR_LENGTH(str), CHARACTER_LENGTH(str), LENGTH(str) y OCTET_LENGTH(str)

Devuelven la longitud de **str**.

CONCAT(cadena1, cadena2, cadena3, ...)

Devuelve una concatenación (unión) de las cadenas pasadas como parámetros. Si alguna cadena es nula, devuelve **NULL**. Si algún argumento es numérico, se lo convierte a cadena de caracteres. Por ejemplo:

```
SELECT CONCAT('una ','sola',' cadena',' – ejercicio ', 1);
//Devuelve 'una sola cadena – ejercicio 1'
```

CONCAT_WS(separador, cadena1, cadena2, cadena3, ...)

Hace lo mismo que la función **CONCAT** solo que el primer parámetro actúa como separador. Por ejemplo:

```
SELECT CONCAT_WS(',', 'uno', 'dos', 'tres');
//Devuelve      'uno,dos,tres'
```

CONV(N, base_origen, base_destino)

Convierte números pasándolos entre diferentes bases numéricas y devuelve una representación a través de una cadena de caracteres del número en la nueva base o **NULL** si cualquier argumento es nulo. La base mínima es **2** y la máxima es **36**. **N** es interpretado como entero, pero puede ser especificado como entero o como cadena (para representar números hexadecimales –base 16– como para poner un ejemplo). Hay otras funciones derivadas de esta:

- **BIN(N)**: devuelve una representación binaria de **N** o **NULL** si **N** es nulo.
- **HEX(P)**: si **P** es un número, la función devuelve una cadena que contiene el valor hexadecimal (base 16) de **P**; si es una cadena de caracteres, devuelve una cadena hexadecimal en donde cada carácter de **P** es convertido a dos dígitos hexadecimales. Devuelve **NULL** si **N** es nulo.
- **OCT(N)**: devuelve una cadena de caracteres que contiene la representación octal (base 8) del número **N** (que es de tipo **LONG**) o **NULL** si **N** es nulo.

ESCRIBIR O NO ESCRIBIR

Si bien hay muchas aplicaciones que nos permiten acceder a las funciones de MySQL de forma gráfica, la línea de comandos de MySQL no deja de ser útil y muy utilizada hoy en día.

Su sintaxis es más sencilla, pero la funcionalidad puede obtenerse utilizando **CONV**.

INSERT (str, pos, len, nuevacadena)

Devuelve la cadena **str** reemplazando su contenido desde la posición **pos** hasta la posición **pos + len** por la cadena **nuevacadena**.

```
SELECT INSERT('el rio dulce', 4, 3, 'Mar');
//Devuelve      'el Mar dulce'
```

ELT(N, str1, str2, str3, ...)

Devuelve **str1** si **N** es igual a **1**, **str2** si **N** es igual a **2**, y así sucesivamente. Si **N** está fuera de índice, devuelve **NULL**.

FIELD(str, str1, str2, str3, ...)

Devuelve el índice (comenzando a contar desde 1) de la cadena **str** en **str1**, **str2**, etcétera, y **0** si no encuentra la cadena. Por ejemplo:

```
SELECT FIELD('bbb', 'aaa', 'bbb', 'ccc', 'ddd', 'eee');
//Devuelve      2
```

FIND_IN_SET(str, strlista)

Devuelve la posición de **str** dentro de la lista **strlista**, **0** si no está en la lista o si la lista está vacía, y **NULL** si algún parámetro es nulo. Por ejemplo:

```
SELECT FIND_IN_SET('b','a,b,c,d');
//Devuelve 2
```

Esta función podría no funcionar correctamente si en **str** hay alguna coma.

INSTR(str,subcadena)

Vea la función **POSITION**.

LCASE(str) y LOWER(str)

Devuelven la cadena **str** en minúsculas. Para pasar a mayúsculas ver **UCASE**.

LEFT(str, N)

Devuelve **N** caracteres de la cadena **str** empezando desde la izquierda.

LIKE

Es muy utilizada y trae muchos beneficios al permitirnos incorporar las llamadas expresiones regulares (que podemos encontrar en el estándar ANSI SQL) en nuestras consultas SQL. Devuelve **1** si hubo coincidencias y **0** si no.

OPERADORES	
▼ CARÁCTER	▼ DESCRIPCIÓN
%	Representa un número cualquiera de caracteres, incluso ninguno.
_	Busca exactamente un carácter.

Tabla 4. Operadores de la función **LIKE**.

Por ejemplo:

```
SELECT 'Buscar!' LIKE 'Buscar_';
//Devuelve 1

SELECT 'Buscar!' LIKE '%B%c%';
//Devuelve 1

SELECT 'Buscar!' LIKE 'Buscar!_';
//Devuelve 0

SELECT 'Buscar!' LIKE '%G%x%';
//Devuelve 0

SELECT 10 LIKE '1%';
//Devuelve 1
```

Si quisiéramos buscar cadenas que contuvieran el carácter **%** o _, tendríamos que anteponer la barra **** (carácter de escape) antes. Se puede modificar el carácter de escape por medio de la cláusula especial **ESCAPE**. Por ejemplo:

```
SELECT 'Buscar_' LIKE 'Buscar|_' ESCAPE '|';
//Devuelve 1
```

Las comparaciones con **LIKE** son insensibles a mayúsculas y minúsculas. Podemos usar la función **BYNARY** para forzar lo contrario. Por ejemplo:

MONITOR DE MYSQL

Para ingresar al monitor de MySQL se puede usar la sintaxis: **MySQL –u nombre_de_usuario –pcontraseña <enter>**, o bien, **MySQL –u nombre_de_usuario –p <enter>**.

```
SELECT 'buscar' LIKE BINARY 'BUSCAR';
//Devuelve 0
```

LOAD_FILE(ruta_completa_a_nombre_archivo)

Lee el archivo y devuelve su contenido como una cadena. Por ejemplo:

```
UPDATE nombre_tabla
   SET columna=LOAD_FILE("/tmp/picture")
   WHERE id=1;
```

LOCATE(subcadena, str)

Devuelve la posición de la primera ocurrencia de **subcadena** en **str** (**0** si no hay coincidencias). Es similar a **INSTR** solo que los argumentos están en orden inverso (primero **str** y luego **subcadena**). Si se le da un tercer argumento (**POS**), la función comenzará a buscar coincidencias a partir de **POS**.

LPAD(str, LEN, COMPLETAR_CON)

Devuelve la cadena **str**, completada por la izquierda con la cadena **COMPLETAR_CON** hasta que **str** sea de **LEN** caracteres de longitud.

Si **str** tiene **LEN** caracteres, la función devuelve **str** sin modificaciones.

Si **str** tiene más caracteres que **LEN**, la función trunca la cadena **str** y devuelve los primeros **LEN** caracteres de **str**. Por ejemplo:

```
SELECT LPAD('cadena',10,'x');
//Devuelve 'xxxxcadena'
```

Para completar por la derecha ver la función **RPAD**.

LTRIM(str)

Devuelve la cadena **str** sin los espacios a la izquierda (si es que los tiene):

```
SELECT LTRIM('  por quien doblan las campanas  ');
//Devuelve 'por quien doblan las campanas  '
```

Ver también **RTRIM**, **TRIM**.

MAKE_SET(índice, str1, str2, ...)

Devuelve las cadenas que corresponden a índice (índice **1** es **str1**, índice **2** es **str2**, ...)

```
SELECT MAKE_SET(1,'a','b','c');
//Devuelve 'a'
```

MATCH (col1, col2, ...) AGAINST (cadena_a_buscar

Devuelve un número positivo si se encontró alguna **cadena_a_buscar** en **col1**, **col2**, ... Para poder utilizar esta opción, hay que definir la tabla en el momento de crearla de una forma particular:

```
CREATE TABLE articulos (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
col1 VARCHAR(200),
col2 TEXT,
FULLTEXT (col1, col2)
);
```

Donde las columnas que se utilizan en **MATCH** tienen que haber sido previamente definidas en **FULLTEXT** en el momento de crear la tabla.

MID(str, POS, LEN)

Devuelve una subcadena de **LEN** caracteres de longitud, partiendo desde el carácter de la posición **POS** de la cadena **str**. Por ejemplo:

```
SELECT SUBSTRING('tesoro', 4, 3);
//Devuelve 'oro'
```

Esta función es idéntica a **SUBSTRING**.

POSITION(subcadena IN str)

Devuelve la posición de la primera ocurrencia de **subcadena** en **str** o **0** si no hay coincidencias. Esta función tiene la misma funcionalidad que **LOCATE**, solo cambia la forma de pasarle los argumentos.

Por ejemplo:

```
SELECT LOCATE('robin' IN 'batman y robin');
//Devuelve 10
```

QUOTE(str)

Esta función fue agregada en MySQL versión 4.0.3. Reemplaza los caracteres que correspondan por sus respectivas secuencias de escape.

```
SELECT QUOTE("D'alessandro");
//Devuelve 'D\'alessandro'
```

REPEAT(str, N)

Devuelve **str** concatenada **N** veces o **NULL** si **str** o **N** es nulo.

```
SELECT REPEAT('lapiz azul ', 2);
//Devuelve 'lapiz azul lapiz azul '
```

REPLACE(str, BUSCAR, REEMPLAZAR)

Devuelve **str** reemplazando todas las ocurrencias de la cadena **BUSCAR** por la cadena **REEMPLAZAR**. Por ejemplo:

```
SELECT REPLACE('la hoja blanca', 'blanca', 'negra');
//Devuelve 'la hoja negra'
```

REVERSE(str)

Devuelve la cadena **str** en orden inverso.

RIGHT(str, N)

Devuelve **N** caracteres de la cadena **str** empezando desde la derecha.

RPAD(str, LEN, COMPLETAR_CON)

Devuelve la cadena **str**, completada por la derecha con la cadena **COMPLETAR_CON** hasta que **str** sea de **LEN** caracteres de longitud.
Si **str** tiene **LEN** caracteres, la función devuelve **str** sin modificaciones. Si **str** tiene más caracteres que **LEN**, la función trunca la cadena **str** y devuelve los primeros **LEN** caracteres de **str**.

```
SELECT RPAD('cadena',10,'x');
//Devuelve 'cadenaxxxx'
```

RTRIM(str)

Devuelve la cadena **str** sin los espacios a la derecha (si es que los tiene). Como el ejemplo que aparece a continuación. Ver también **LTRIM**, **TRIM**.

```
SELECT RTRIM(' adios a las armas  ');
//Devuelve ' adios a las armas'
```

SPACE(N)

Devuelve una cadena compuesta por **N** espacios.

STRCMP(str1, str2)

Devuelve **0** si **str1** y **str2** son iguales, **–1** si **str1** es más chico (de acuerdo al ordenamiento actual) que **str2**, y **1** si **str2** es más chico (de acuerdo al ordenamiento actual) que **str1**. Por ejemplo:

```
SELECT STRCMP('text', 'text2');
//Devuelve -1

SELECT STRCMP('text2', 'text');
//Devuelve 1

SELECT STRCMP('text', 'text');
//Devuelve 0
```

SUBSTRING(str,pos,len)

También admite la forma **SUBSTRING(str FROM pos FOR len)**. Ver la función **MID**.

SUBSTRING_INDEX(str, delim, count)

Si **count** es positivo, busca las ocurrencias de **delim** dentro de **str** comenzando desde la izquierda. A la ocurrencia número **count**, devuelve lo que está a la izquierda de esta. Si **count** es negativo, busca las ocurrencias de **delim** dentro de **str** comenzando desde la derecha. A la ocurrencia número **count**, devuelve lo que está a la derecha de esta. Por ejemplo:

```
SELECT SUBSTRING_INDEX('www.MySQL.com', '.', 2);
//Devuelve 'www.MySQL'

SELECT SUBSTRING_INDEX('www.MySQL.com', '.', -2);
//Devuelve 'MySQL.com'
```

TRIM([[BOTH | LEADING | TRAILING] [cadena_a_borrar] FROM] str)

Si se utiliza sin opciones (o con la opción **BOTH**, es lo mismo) devuelve la cadena **str** eliminando las ocurrencias de **cadena_a_borrar** al final y al principio.

ANIDAR FUNCIONES

La utilidad de las funciones provistas por MySQL es muy poderosa, pero se puede incrementar anidando las funciones para obtener el resultado buscado.

Si **cadena_a_borrar** no se especifica, se asume por defecto que es un espacio. Con la opción **LEADING** se borran solo las ocurrencias de **cadena_a_borrar** desde la izquierda. Con la opción **TRAILING** se borran solo las ocurrencias de **cadena_a_borrar** desde la derecha. Por ejemplo:

```
SELECT TRIM(' algo  ');
//Devuelve 'algo'

SELECT TRIM(LEADING 'x' FROM 'xxxalgoxxx');
//Devuelve 'algoxxx'

SELECT TRIM(BOTH 'x' FROM 'xxxalgoxxx');
//Devuelve 'algo'

SELECT TRIM(TRAILING 'xyz' FROM 'algoxxyz');
//Devuelve 'algox'
```

UCASE(str) y UPPER(str)

Devuelven la cadena **str** en mayúsculas. Para pasar a minúsculas ver **LCASE**.

Funciones para trabajar con campos numéricos

Todas las funciones matemáticas devuelven **NULL** en casos de error.

Si una operación excede el rango del tipo de dato **BIGINT** (64 bits), el resultado que se obtiene es **0**. Son válidos los operadores elementales (**+**, **-**, *, */*) y la regla de los paréntesis (por ejemplo: **((a + b) * c)** es distinto de **(a + b * c)**).

-(X)

Cambia de signo el argumento. Por ejemplo:

```
SELECT –(-2);
//Devuelve 2
```

ABS(X)

Devuelve el valor absoluto del número **X**.

ACOS(X)

Devuelve el arco coseno de **X** (**X** son radianes).

ASIN(X)

Devuelve el arco seno de **X** (**X** son radianes).

ATAN(X)

Devuelve el arco tangente de **X** (**X** son radianes).

CEILING(X)

Devuelve el entero (en formato **BIGINT**) inmediatamente superior a **X**. Por ejemplo:

```
SELECT FLOOR(1.23);
//Devuelve 2

SELECT FLOOR(-1.23);
//Devuelve -1
```

COS(X)

Devuelve el coseno de **X** (**X** son radianes).

COT(X)

Devuelve la cotangente de **X** (**X** son radianes).

DEGREES(X)

Devuelve **X** (se suponen radianes) convertido a grados.

DIV

Operador, realiza una división y devuelve como resultado un número entero. Por ejemplo:

```
SELECT 7 DIV 2
//Devuelve 3
```

Para realizar una división real se utiliza el operador **/**.
Por ejemplo:

```
SELECT 3/5;
//Devuelve 0.60
```

DIV está disponible desde la versión 4.1.0 de MySQL.

EXP(X)

Devuelve el número entero (**2.718281828** ...) elevado a la **X**.

FLOOR(X)

Devuelve el entero (en formato **BIGINT**) inmediatamente inferior a **X**.
Por ejemplo:

```
SELECT FLOOR(1.23);
//Devuelve 1

SELECT FLOOR(-1.23);
//Devuelve -2
```

GREATEST(X,Y, ...)

Similar a **LEAST**, devuelve el mayor de los argumentos.

INTERVAL(N, N1, N2, N3, ...)

Devuelve **0** si **N** es menor a **N1**, **1** si **N** es menor a **N2**, **2** si **N** es menor a **N3**, y así sucesivamente. Se detiene a la primera verdad (si **N** es menor a **N1** devuelve **0** y no compara **N** con **N2**). Los argumentos deben ser enteros (serán tratados como tales). Es obligatorio que **N1< N2< N3< ...< NN**. Por ejemplo:

```
SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
//Devuelve 3

SELECT INTERVAL(10, 1, 10, 100, 1000);
//Devuelve 2

SELECT INTERVAL(22, 23, 30, 44, 200);
//Devuelve 0
```

LEAST(X,Y, ...)

Similar a **GREATEST**, pero devuelve el menor de los argumentos. Por ejemplo:

```
SELECT LEAST(2,0);
//Devuelve 0

SELECT LEAST(34.0,3.0,5.0,767.0);
//Devuelve 3.0

SELECT LEAST("B","A","C");
//Devuelve "A"
```

LN(X)

Devuelve el logaritmo natural de **X**. Disponible desde la versión 4.1.3.

LOG([B,] X)

Devuelve el logaritmo en base **B** (si **B** no se pasa, se asume logaritmo natural) de **X**. La opción de la base arbitraria está disponible desde la versión 4.0.3.

MOD(N,M)

Devuelve el resto de dividir **N** por **M**. Puede utilizarse también el signo **%**.

```
SELECT MOD(10, 2);
//Devuelve 0

SELECT MOD(29,9);
//Devuelve 2

SELECT 29 % 9;
//Devuelve 2
```

Otro modo de usar la función, disponible desde la versión 4.1, es:

```
SELECT 29 MOD 9;
//Devuelve 2
```

PI()

Devuelve el valor de **PI**.

POW(X,Y) y POWER(X,Y)

Devuelven el valor que se obtiene al elevar **X** a la potencia **Y**.

RADIANS(X)

Devuelve **X** (se suponen grados) convertido a radianes.

RAND([N])

Si no se le da ningún argumento, devuelve un valor aleatorio entre **0** y **1**. Si se le da el argumento **N** (**entero**) hace lo mismo, pero mantiene el valor obtenido para sucesivas llamadas. Por ejemplo:

```
SELECT RAND();
//Devuelve 0.17573482386203

SELECT RAND(20);
//Devuelve 0.15888261251047

SELECT RAND(20);
//Devuelve 0.15888261251047
```

```
SELECT RAND();
//Devuelve 0.44566677782312
```

Otro uso interesante para la función es:

```
SELECT * FROM tabla1 ORDER BY RAND();
```

El orden de los registros devueltos variará aleatoriamente cada vez que se ejecute la consulta. Solo para MySQL 3.23 o superiores.

ROUND(X [,D])

Devuelve el número **X** y lo redondea tomando **D** posiciones decimales (**0** si no se da **D**). Por ejemplo:

```
SELECT ROUND(1.58);
//Devuelve 2

SELECT ROUND(1.298, 1);
//Devuelve 1.3
```

SIGN(X)

Devuelve el signo de **X** (**-1** si es negativo, **0** si es **0**, **1** si es positivo).

SIN(X)

Devuelve el seno de **X** (**X** son radianes).

SQRT(X)

Devuelve la raíz cuadrada de **X.**

TAN(X)

Devuelve la tangente de **X** (**X** son radianes).

TRUNCATE(X,D)

Devuelve **X** truncado a **D** decimales. Truncar no significa redondear, sino tomar **D** decimales. Si **D** es negativo, devuelve la parte entera de **X** y pone a **0** los últimos **D** dígitos de **X**. Por ejemplo:

```
SELECT TRUNCATE(1.999, 1);
//Devuelve 1.9
SELECT TRUNCATE(1.999,0);
//Devuelve 1
SELECT TRUNCATE(12345.9876, -2);
//Devuelve 12300
SELECT TRUNCATE(122, -2);
//Devuelve 100
```

¿EN QUÉ SE USAN LAS FUNCIONES MATEMÁTICAS?

Hay funciones matemáticas que tal vez la mayoría de los usuarios no utilicemos nunca, pero son muy útiles para programar aplicaciones orientadas al cálculo de probabilidades y estadísticas de sucesos, al poder unir el hecho de manejar una enorme cantidad de datos con una gran precisión en los cálculos.

Funciones para trabajar con fecha y hora

Cuando en algunas funciones se representa el año con solo dos dígitos, MySQL lo maneja de la siguiente manera:

- Si el año está entre **00** y **69**, es tratado como **2000-2069**.
- Si el año está entre **70** y **99**, es tratado como **1970-1999**.

ADDDATE(date,INTERVAL expr type)

Ver la función **SUBDATE**.

CURDATE() y CURRENT_DATE

Devuelven la fecha actual en formato **'YYYY-MM-DD'** o **YYYYMMDD** (si se pone **CURRENT_DATE() + 0**).

CURRENT_TIME y CURTIME()

Devuelven el horario actual en formato **'HH:MM:SS'** o **HHMMSS** (si se pone **CURRENT_TIME() + 0**).

CURRENT_TIMESTAMP

Ver la función **NOW**.

DATE_ADD(date,INTERVAL expr type)

Ver la función **SUBDATE**.

DATE_FORMAT(date,formato)

Da formato a una fecha. Existen las siguientes opciones (atención a las mayúsculas y minúsculas):

FORMATO DE FECHAS

▼ SIGNO	▼ DESCRIPCIÓN
%M	Nombre del mes en inglés (January...December).
%W	Nombre del día de la semana en inglés (Sunday...Saturday).
%D	Número de día del mes en inglés (0th, 1st, 2nd, 3rd...).
%Y	Año en cuatro dígitos.
%y	Año en dos dígitos.
%X	Cada año tiene una cantidad de semanas. Cada semana tiene un número. Una semana perteneciente a un año (que comienza en ese año) puede terminar en el año siguiente. Lo que hace este formato es devolver el año (4 dígitos) al que pertenece la semana de la fecha en cuestión. Toma como primera semana a la 01 y como primer día de la semana al domingo.
%x	Lo mismo que %X, pero toma como primer día de la semana el lunes.
%a	Nombre del día de la semana abreviado en inglés (Sun...Sat).
%d	Número de día del mes (00...31), siempre de dos dígitos.
%e	Número de día del mes (0...31), uno o dos dígitos según corresponda.
%m	Número de mes (00...12), siempre de dos dígitos.
%c	Número de mes (0...12), uno o dos dígitos según corresponda.
%b	Nombre del mes abreviado en inglés (Jan...Dec).
%j	Número de día del año (001...366).
%H	Hora (00...23), siempre de dos dígitos.
%k	Hora (0...23), uno o dos dígitos según corresponda.
%h	Hora (01...12).

FORMATO DE FECHAS (CONTINUACIÓN)	
%l	Hora (01…12).
%I	Hora (1…12), uno o dos dígitos según corresponda.
%i	Minutos (00...59).
%r	Formato 12 horas (hh:mm:ss AM o hh:mm:ss PM).
%T	Formato 24 horas (hh:mm:ss).
%S	Segundos (00...59).
%s	Segundos (00…59).
%p	AM o PM.
%w	Número de día de la semana en inglés (0=Sunday...6=Saturday).
%U	Número de semana del año (00...53) (domingo como primer día).
%u	Número de semana del año (00...53) (lunes como primer día).
%V	Número de semana del año (01...53) (domingo como primer día).
%v	Número de semana del año (01...53) (lunes como primer día).

Tabla 5. Parámetros de **DATE_FORMAT**.

¿QUÉ HORA ES?

Cuando trabajamos con funciones para el manejo de fechas y horas, debemos tener en cuenta que toman como tiempo actual los horarios del lugar en donde se encuentre instalado el servidor de bases de datos. Un ejemplo es la función **NOW()**.

Por ejemplo:

```
SELECT DATE_FORMAT('1997-10-04 22:23:00','%W %M %Y');
//Devuelve 'Saturday October 1997'

SELECT DATE_FORMAT('1997-10-04 22:23:00','Son las %H:%i:%s horas');
//Devuelve 'Son las 22:23:00 horas'

SELECT DATE_FORMAT('1997-10-04 22:23:00',
'%D %y %a %d %m %b %j');
//Devuelve '4th 97 Sat 04 10 Oct 277'

SELECT DATE_FORMAT('1997-10-04 22:23:00',
'%H %k %I %r %T %S %w');
//Devuelve '22 22 10 10:23:00 PM 22:23:00 00 6'

SELECT DATE_FORMAT('1999-01-01','%X %V');
//Devuelve '1998 52'

SELECT DATE_FORMAT(NOW(),'%Y');
//Devuelve el año actual utilizando 4 dígitos. Ver función NOW()
```

DAYNAME(date)

Devuelve el nombre (en inglés) del día de la fecha date. Por ejemplo:

```
SELECT DAYNAME("1998-02-05");
//Devuelve 'Thursday'
```

DAYOFMONTH(date)

Devuelve el número del día dentro del mes (desde **1** a **31**) de la fecha date.

DAYOFWEEK(date)

Devuelve el número de día de la fecha date (**1** para domingo, **2** para lunes... **7** para sábado). Es similar a **WEEKDAY**, pero numera los días de otra manera.

DAYOFYEAR(date)

Devuelve el número del día dentro del año (desde **1** a **366**) de la fecha date.

EXTRACT(type FROM date)

Extrae **type** de la fecha **date**. Por ejemplo:

```
SELECT EXTRACT(YEAR FROM "1999-07-02");
//Devuelve 1999

SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
//Devuelve 199907
```

Para ver los posibles valores de **type**, ir a la función **SUBDATE**.

FROM_DAYS(N)

Es la función inversa a **TO_DAYS**: devuelve la fecha que se obtiene sumando **N** días al año **0**. Por ejemplo:

```
SELECT FROM_DAYS(729669);
//Devuelve '1997-10-07'
```

FROM_UNIXTIME(unix_timestamp [,formato])

Devuelve el **unix_timestamp** (ver función **UNIX_TIMESTAMP**) con formato. Si no se da el argumento formato, la función devuelve **'YYYY-MM-DD HH:MM:SS'** o **YYYYMMDDHHMMSS**. Los posibles formatos están definidos en la función **DATE_FORMAT**. Por ejemplo:

```
SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),'%Y %D %M %h:%i:%s');
//Devuelve '2005 6rd February 21:57:10'
```

El resultado obtenido al ejecutar esta consulta no será el mismo que el que figura aquí, ya que **UNIX_TIMESTAMP()** depende de la fecha y horario actuales.

HOUR(time)

Devuelve la hora del horario **time** (desde **0** a **23**). Por ejemplo:

```
SELECT HOUR('10:05:03');
//Devuelve 10
```

MINUTE(time)

Devuelve los minutos del horario **time** (desde **0** a **59**).

MONTH(date)

Devuelve el número de mes (desde **1** a **12**) de la fecha **date**.

MONTHNAME(date)

Devuelve el nombre (en inglés) del mes de la fecha **date**. Por ejemplo:

```
SELECT MONTHNAME("1998-02-05");
//Devuelve 'February'
```

NOW() y SYSDATE()

Devuelve la fecha y hora actuales en formato **'YYYY-MM-DD HH:MM:SS'** o **YYYYMMDDHHMMSS** (si se pone **NOW() + 0**).

PERIOD_ADD(P, N)

Suma **N** meses a **P**. **P** es un período de formato **YYMM** o **YYYYMM** (como agosto de **1999** puede ser **9908** o **199908**). El valor devuelto es de tipo **YYYYMM**. Por ejemplo:

```
SELECT PERIOD_ADD(9801,2);
//Devuelve 199803
```

PERIOD_DIFF(P1,P2)

Devuelve el número de meses entre el período **P1** y **P2** (que solo pueden ser de formato **YYMM** o **YYYYMM**). Por ejemplo:

```
SELECT PERIOD_DIFF(9802,199703);
//Devuelve 11
```

QUARTER(date)

Divide al año en cuatro períodos de tres meses cada uno y devuelve el período al que pertenece la fecha date. Por ejemplo:

```
SELECT QUARTER('98-04-01');
//Devuelve 2
```

SECOND(time)

Devuelve los segundos del horario time (desde **0** a **59**). Por ejemplo:

```
SELECT SECOND ('10:05:03');
//Devuelve 3
```

SEC_TO_TIME(segundos)

Convierte los segundos al formato **'HH:MM:SS'** o **HHMMSS** (si se pone **SEC_TO_TIME(segundos) + 0**).

SUBDATE(date,INTERVAL expr type)

El argumento date es un campo de tipo **DATETIME** o **DATE** que especifica la fecha de comienzo; **expr** es el intervalo que se va a sumar o restar a date; **type** indica que es **expr**. Si queremos sumar un intervalo a la fecha date, hay dos maneras de hacerlo:

- Anteponiendo a **interval** el **signo +**.
- Utilizando la función **DATE_ADD**.

Si queremos restar un intervalo a la fecha **date**, hay dos maneras de hacerlo:

- Anteponiendo a **interval** el **signo -**.
- Utilizando la función **DATE_SUB**.

VALORES QUE PUEDE TOMAR TYPE	
▼ VALORES SOPORTADOS POR TYPE	▼ FORMATO DE EXPR
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
MONTH	MONTHS
YEAR	YEARS
MINUTE_SECOND	"MINUTES:SECONDS"
HOUR_MINUTE	"HOURS:MINUTES"
DAY_HOUR	"DAYS HOURS"
YEAR_MONTH	"YEARS-MONTHS"
HOUR_SECOND	"HOURS:MINUTES:SECONDS"
DAY_MINUTE	"DAYS HOURS:MINUTES"
DAY_SECOND	"DAYS HOURS:MINUTES:SECONDS"

Tabla 6. Valores de **`type`**.

Por ejemplo, para sumar un segundo a la fecha dada:

- Utilizando signos:

```
SELECT "1999-10-31 22:49:49" + INTERVAL 1 SECOND;
```

- Utilizando la función:

```
SELECT DATE_ADD("1999-10-31 22:49:49", INTERVAL 1 SECOND);
```

Ambas devuelven **1999-10-31 22:49:50**.
Para restar un minuto a la fecha dada:

- Utilizando signos.

```
SELECT "1999-10-31 22:49:49" - INTERVAL 1 MINUTE;
```

- Utilizando función.

```
SELECT DATE_SUB("1999-10-31 22:49:49", INTERVAL 1 MINUTE);
```

Ambas devuelven **1999-10-31 22:48:49**.
Para sumar 1 minuto y un segundo a la fecha dada:

```
SELECT DATE_ADD("1997-12-31 23:59:59",
INTERVAL "1:1" MINUTE_SECOND);
```

Devuelve **1998-01-01 00:01:00**.
Para sumar una hora a la fecha dada:

```
SELECT DATE_ADD("1999-01-01", INTERVAL 1 HOUR);
```

Devuelve **1999-01-01 01:00:00**.
Para sumar 1 mes a la fecha dada:

```
SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
```

Devuelve **1998-02-28**.

En esta última consulta, sumamos un mes a partir del 30 de enero. Como febrero no tiene día 30 se asume el día más cercano, o sea, su último día (28). Las funciones **ADDDATE()** y **DATE_ADD()** son idénticas.

Las funciones **SUBDATE()** y **DATE_SUB()** son idénticas.

TIME_FORMAT(time, formato)

Funciona del mismo modo que la función **DATE_FORMAT()**, solo que puede utilizar nada más que los formatos referidos a horas, minutos y segundos.

TIME_TO_SEC(time)

Convierte time a segundos. Por ejemplo:

```
SELECT TIME_TO_SEC('22:23:00');
//Devuelve 80580
```

TO_DAYS(date)

Es la función inversa a **FROM_DAYS**. Devuelve la cantidad de días desde el año 0 hasta la fecha date. Por ejemplo:

```
SELECT TO_DAYS('1997-10-07');
//Devuelve 729669
```

UNIX_TIMESTAMP(date)

El **Unix timestamp** es muy utilizado en funciones de PHP que trabajan con fechas. Son los segundos transcurridos desde el 1 de enero de 1970 a la hora 00:00:00. Si la función **UNIX_TIMESTAMP** se llama sin argumentos, devuelve los segundos transcurridos desde **1970-01-01 00:00:00** hasta ahora **(NOW())**. Si se la llama con un argumento, devuelve los segundos transcurridos desde **1970-01-01 00:00:00** hasta date. Date puede ser de tipo **DATE**, **DATETIME**, **TIMESTAMP** o un número tipo **YYMMDD** o **YYYYMMDD**. Por ejemplo:

```
SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
//Devuelve 875996580
```

WEEK(date [, inicio])

Si se le da solo el primer argumento, devuelve el número de semana del año correspondiente a **date**. Si se le dan los dos argumentos, el segundo permite definir el primer día de la semana (domingo o lunes), y si la primera semana del año se define con el índice **1** o **0**.

- Si inicio es igual a **0**, la semana empieza en **domingo** y la primera semana es **0**.
- Si inicio es igual a **1**, la semana empieza en **lunes** y la primera semana es **0**.
- Si inicio es igual a **2**, la semana empieza en **domingo** y la primera semana es **1**.
- Si inicio es igual a **3**, la semana empieza en **lunes** y la primera semana es **1**.

WEEKDAY(date)

Devuelve el número de día de la fecha date (**0** para **lunes**, **1** para **martes**, **2** para **miércoles**... **6** para **domingo**). Es similar a **DAYOFWEEK**, pero numera los días de otra manera.

YEAR(date)

Extrae el año de la fecha **date**. Por ejemplo:

```
SELECT YEAR('98-02-03');
//Devuelve 1998
```

YEARWEEK(date [, inicio])

Devuelve el año y el número de semana de la fecha **date**. El segundo argumento es opcional y funciona igual que el segundo argumento de la función **WEEK**.

Funciones de conversión

Estas funciones nos permiten tomar el valor almacenado en otras variables y convertirlos en el tipo de datos que deseemos (en la medida que lo permita la función). Veamos su uso.

CAST(expresion AS tipo)

La función **CAST** se utiliza para convertir un tipo de dato en otro. El argumento tipo puede ser:

SOPORTE PARA FUNCIONES

Al utilizar una u otra función debemos considerar la versión del servidor de bases de datos MySQL, en donde se alojará el sitio web. Una función no soportada nos obligaría a replantear partes del código fuente de la aplicación. Podemos recurrir a la ayuda y a los programas de MySQL para conocer la versión del servidor y los requerimientos de las funciones.

- **BINARY**
- **CHAR** (desde versión 4.0.6)
- **DATE**
- **DATETIME**
- **SIGNED** (entero con signo)
- **TIME**
- **UNSIGNED** (entero sin signo)

Por ejemplo: tenemos un campo tipo texto (definido como **VARCHAR(2)**) que contiene caracteres numéricos desde **1** hasta **10** y queremos ordenar estos caracteres. Si hiciéramos algo como:

```
SELECT * FROM tabla ORDER BY campo_texto;
```

Obtendríamos la secuencia 1,10, 2, 3, 4, 5, 6, 7, 8, 9, pero usando **CAST**:

```
SELECT * FROM tabla ORDER BY CAST(campo_texto AS BINARY);
```

Obtendríamos la secuencia 1,2,3,4,5,6,7,8,9,10. **CAST** es sinónimo de la función **CONVERT(expresion,tipo)**. Ambas funciones fueron añadidas en la versión 4.0.2.

Funciones agregadas o estadísticas

Estas funciones se aplican en un grupo de registros y devuelven un único valor. Se usan dentro de la cláusula **SELECT**. Las más utilizadas son las que describimos a continuación.

- **AVG**: calcula el promedio de los valores de un campo determinado.
- **COUNT**: devuelve el número de registros de la consulta **SELECT**.
- **SUM**: devuelve la suma de todos los valores de un campo determinado.

- **MAX**: devuelve el valor más alto de un campo especificado.
- **MIN**: devuelve el valor más bajo de un campo especificado.

Operadores de comparación

Las comparaciones pueden devolver tres valores **0** (falso), **1** (verdadero) y **NULL** (nulo). Si en una comparación por lo menos uno de los elementos que se están comparando es **NULL**, la comparación dará como resultado **NULL**, excepto si se usa el operador de comparación **<=>**.

OPERADORES DE COMPARACIÓN

SIGNIFICADO	OPERADOR	EJEMPLO
Igualdad	=	SELECT 1 = 1, 1 = 0, 1 = NULL; //Devuelve 1, 0, NULL.
Desigualdad	<> !=	SELECT 1 <> 1, 1 <> 0, 1 <> NULL; //Devuelve 0, 1, NULL
Menor o igual que	<=	SELECT 1 <= 1, 1 <= 10, 1 <= NULL; //Devuelve 1, 1, NULL
Menor que	<	SELECT 1 < 1, 1 < 10, 1 < NULL; Devuelve 0, 1, NULL
Mayor o igual que	>=	SELECT 1 >= 1, 1 >= 10, 1 >= NULL; //Devuelve 1, 0, NULL
Mayor que	>	SELECT 1 > 1, 1 > 10, 1 > NULL; //Devuelve 0, 0, NULL
Trata de forma especial a NULL	<=>	SELECT 1 <=> 1, 1 <=> NULL; //Devuelve 1, 0
¿Es nulo?	IS NULL	SELECT ISNULL(1+1), ISNULL(1/0); Devuelve 0 1

OPERADORES DE COMPARACIÓN (CONTINUACIÓN)		
¿Es no nulo?	**IS NOT NULL**	**SELECT IS NOT NULL(1+1), IS NOT NULL(1/0); //Devuelve 1, 0**
¿Está en la lista?	**IN**	**SELECT 'wefw' IN (0,3,5,'wefw'); //Devuelve 1**
¿No está en la lista?	**NOT IN**	**SELECT 'wefw' NOT IN (0,3,5,'wefw'); //Devuelve 0**

Tabla 7. Lista de operadores.

Operadores lógicos

Los operadores lógicos, al igual que en el desarrollo, son muy útiles para realizar comparaciones de valores o estados de diferentes elementos. En SQL, todos los operadores lógicos se evalúan como **TRUE**, **FALSE**, o **NULL**. En MySQL, se implementan como **1** (**TRUE**), **0** (**FALSE**), y **NULL**.

NOT o !

Negación. Por ejemplo:

```
SELECT !(1+1);
//Devuelve 0 (1+1=2, 2 es distinto de 0, 2
es verdadero, lo distinto de verdadero es falso)

SELECT ! 1+1;
// Devuelve 1 (atencion los parentesis! Es equivalente a !(1) + 1:
lo distinto de 1 es falso, o sea 0, y 0+1=1, que es verdadero)
```

AND o **&&** devuelve **1** si todos los operandos son verdaderos, si no, devuelve **0**.

Por ejemplo:

```
SELECT 1 && 1;
//Devuelve 1

SELECT 1 && 0;
//Devuelve 0
```

OR o ||

Devuelve **1** si alguno de los operandos es verdadero, **NULL** si alguno es **NULL** y el otro es **0**; si no, devuelve **0**. Por ejemplo:

```
SELECT 1 || 1;
//Devuelve 1

SELECT 1 || 0;
//Devuelve 1

SELECT 0 || 0;
//Devuelve 0

SELECT 0 || NULL;
//Devuelve NULL

SELECT 1 || NULL;
//Devuelve 1
```

XOR

Devuelve **1** si y solo si únicamente uno de los operandos es **1**, a menos que el otro sea **NULL**, en cuyo caso devuelve **NULL**; si no, devuelve **0**. Por ejemplo:

```
SELECT 1 XOR 1;
//Devuelve 0

SELECT 1 XOR 0;
//Devuelve 1

SELECT 1 XOR NULL;
//Devuelve NULL

SELECT 1 XOR 1 XOR 1;
//Devuelve 1. Primero evalua 1 XOR 1
(que devuelve 0), luego el resultado anterior y 1 (0 XOR 1)
```

El uso del monitor de MySQL

Por lo general, para acceder a bases de datos MySQL utilizaremos nuestras propias páginas PHP o algún programa para administración (**PHPMyAdmin** o **MySQLAB**, por ejemplo), pero MySQL trae un programa cliente que nos permite acceder a los datos. No tiene un nombre específico, algunos lo llaman **prompt** o línea de comandos o monitor de MySQL. Nos servirá para dar los primeros pasos con la base de datos, conocer a fondo su sintaxis y familiarizarnos con los mensajes de error.

Para ponerlo en marcha (primero debemos asegurarnos de que el servidor MySQL esté activo), lo único que tenemos que hacer es abrir una terminal en Linux o una ventana de DOS en Windows, ir hasta el directorio donde instalamos MySQL y entrar en la carpeta **bin**.

Ahora, según el caso, procedemos del siguiente modo:

- Si no creamos ningún usuario, lo más probable es que podamos entrar escribiendo simplemente:

```
MySQL <enter>
```

- Si ya creamos un usuario, tenemos que entrar escribiendo:

```
MySQL –u nombre_de_usuario –pcontraseña          <enter>
```

No debemos dejar espacio entre **–p** y **contraseña**, sino escribirlo todo junto. Si todo funcionó bien, veremos un mensaje de bienvenida y el programa estará preparado para recibir nuestras instrucciones. Podremos ingresar sentencias SQL de cualquier tipo. Las sentencias terminan con **;** (punto y coma), esto significa que, aunque presionemos la tecla **ENTER** (**<enter>**), esto no producirá ningún cambio salvo el de introducir una nueva línea. Para ejecutar una sentencia tenemos que escribirla, finalizarla con **;** y presionar **ENTER** (**<enter>**).

PROMPT DE MYSQL

▼ SENTENCIA	▼ DESCRIPCIÓN
MySQL>	Listo para un nuevo comando.
->	Esperando una nueva línea.
'>	Esperando la siguiente línea, hay una cadena abierta con ' (comillas simples).
">	Esperando la siguiente línea, hay una cadena abierta con " (comillas dobles).

Tabla 8. Lista de posibles **prompt**.

La tercera y cuarta opción de la tabla se dan, por ejemplo, cuando estamos ingresando una cadena de texto. Todo lo que escribamos será parte de la cadena sin importar si es un **;** (punto y coma) o un **<enter>** o cualquier otra cosa que no sea el correspondiente cierre de la cadena (si abrimos la cadena con comillas dobles, el cierre será comillas dobles; si abrimos la cadena con comilla simple, el cierre será otra comilla simple). Para cerrar el programa y volver a DOS o al prompt de Linux, existen las directivas **quit** y **exit**. Para cancelar una directiva (antes de finalizarla con **;**) existe la directiva **\c**. Por ejemplo:

```
MySQL> SELECT * FROM alumnos
-> WHERE
-> \c
MySQL>
```

Algunos ejemplos:

```
MYSQL> CREATE DATABASE ejemplo;

MYSQL> USE ejemplo;
MYSQL> CREATE TABLE tabla1 (
-> campo1 INT(4) UNSIGNED,
-> campo2 VARCHAR(25) NOT NULL
-> );

MYSQL> INSERT INTO tabla1 VALUES (2,'CADENA A');

MYSQL> INSERT INTO tabla1 VALUES (1,'CADENA B');

MYSQL> SELECT * FROM tabla1 WHERE campo1 >= 1;

MYSQL> DROP TABLE tabla1;

MYSQL> EXIT
```

Hay comandos que devuelven un gran número de filas. Para guardarlas, ingresamos al monitor de MySQL escribiendo una de estas opciones:

```
MySQL –u nombre_de_usuario –pcontraseña --tee="uno.txt" <enter>
```

```
MySQL --tee="uno.txt" <enter>
```

Todos los comandos y sus salidas se guardarán en el archivo **uno.txt** (en el mismo directorio donde se encuentra MySQL), que no podrá ser abierto mientras la conexión esté activa. Por último, para obtener un listado completo de las opciones para ingresar al monitor podemos escribir:

```
MySQL –?<enter>
```

```
MySQL –? > opciones.txt<enter>
```

La primera muestra las opciones por pantalla, mientras que la segunda las guarda en el archivo **opciones.txt**, dentro de la misma carpeta **bin**.

RESUMEN

Realizamos una introducción a MySQL y analizamos las posibilidades que puede brindarnos. Algunas de ellas son: los tipos de tablas y los tipos de datos soportados para definir campos. Por último, repasamos las funciones e iniciamos el uso del monitor MySQL.

Actividades

TEST DE AUTOEVALUACIÓN

1. Nombre tres tipos de tablas y sus características principales.
2. ¿Cuáles son los principales tipos de datos provistos por MySQL?
3. ¿Cuál es la diferencia entre los subtipos de datos **DATETIME** y **DATE**?
4. ¿Qué valor representa el **timestamp** de Unix?
5. ¿Con que función podemos saber el **timestamp** actual?

EJERCICIOS PRÁCTICOS

1. Utilizando las funciones **DATE_FORMAT()** y **NOW()** obtenga la siguiente salida: **'este año xxxx será maravilloso para los que aprendan a usar mysql) !!!!',** donde **xxxx** es el año actual en cuatro dígitos.
2. Eleve el número 10 al cuadrado, luego calcule la raíz cuadrada del resultado obtenido y por último cámbiele el signo al resultado final. Todo esto utilizando las funciones matemáticas de MySQL.
3. Dada la cadena **'Carlos Marx'** obtenga **'Groucho Marx'** utilizando las funciones para manejo de cadenas de caracteres de MySQL.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: **profesor@redusers.com**.

*04

PHP y MySQL

En este capítulo, conoceremos las distintas alternativas que nos ofrece el lenguaje PHP para acceder e, incluso, definir estructuras de bases de datos en MySQL. Comenzaremos con las sintaxis de funciones, analizaremos sus diferencias, y veremos algunos ejemplos y comentarios sobres ellas.

Conectar PHP con MySQL.....104
mysql_connect 104
mysql_pconnect 107
Ejecución de sentencias 108
Creación de base de datos a través de PHP.....110
Selección de una base de datos 113
Creación de tablas a través de PHP.....114
Obtener listados de bases de datos y tablas a través de PHP.....116
Uso de múltiples bases de datos en una aplicación 119
Cerrar conexión con la base de datos 121
Ejemplo práctico: autos.php.....123

Resumen.....125

Actividades.....126

Conectar PHP con MySQL

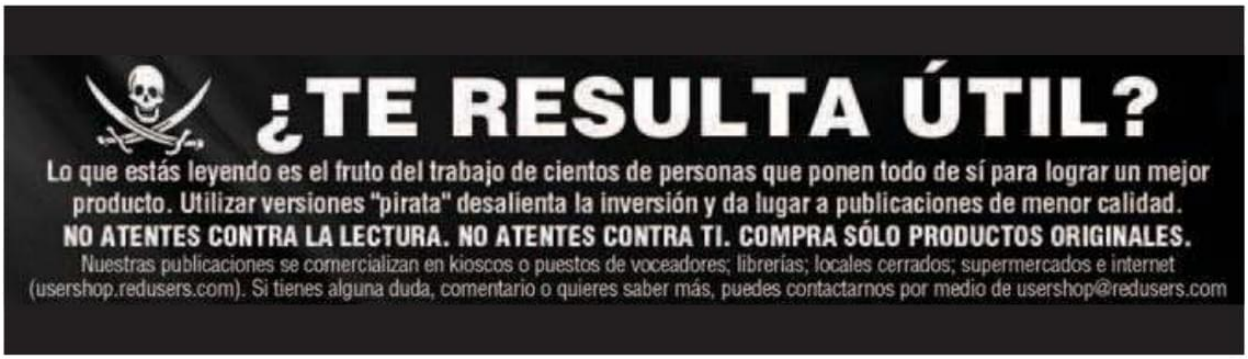
Antes de empezar a interactuar con una base de datos, tenemos que establecer una conexión entre PHP y el servidor de bases de datos MySQL. Para hacerlo, PHP nos ofrece dos funciones: **mysql_connect** y **mysql_pconnect**, que, aunque son similares, difieren en algunos aspectos.

mysql_connect

Esta función intenta conectar a una base de datos y requiere que se le pasen, de manera opcional, los siguientes argumentos (en este orden):

```
mysql_connect("servidor", "nombre de usuario", "password");
```

- **servidor**: cadena de caracteres que debe contener el nombre del servidor o bien su dirección IP. Cuando hablamos de servidor, nos referimos a la máquina donde se encuentra instalado el servidor de bases de datos MySQL. Si trabajando en forma local, podemos utilizar como nombre de servidor **localhost** o **127.0.0.1** como dirección IP. También, se puede agregar el puerto de conexión de la siguiente forma: **servidor:puerto** (por ejemplo **localhost:80**).
- **nombre de usuario**: nombre de usuario válido para acceder a la base de datos.
- **password**: contraseña correspondiente al nombre de usuario que hemos ingresado anteriormente.



Como dijimos, los argumentos son opcionales y, si no se especifica alguno, se asumen los valores por defecto que se encuentran en el archivo **php.ini**. Allí encontraremos las siguientes líneas:

mysql.default_port =
(puerto por defecto)

mysql.default_host =
(nombre del servidor por defecto)

mysql.default_user =
(nombre de usuario por defecto)

mysql.default_password =
(clave de usuario por defecto)

Para completar las opciones del **php.ini**, simplemente tenemos que escribir la información a continuación del signo igual. Por ejemplo, para completar solo el usuario y el nombre del servidor:

mysql.default_port =
mysql.default_host = "168.22.22.9"
mysql.default_user = "pepe"
mysql.default_password =

Si tenemos valores por defecto en el archivo, pero de igual manera completamos los argumentos de **mysql_connect**, se tendrán en cuenta estos últimos por sobre los primeros. Los argumentos son opcionales pero piramidales:

- Solo el nombre del servidor.
- Solo el nombre del servidor y el nombre de usuario.
- El nombre del servidor, el nombre de usuario y la contraseña.
- Dejarlos sin argumentos.

Es decir, no se puede, por ejemplo, ingresar la contraseña y dejar vacíos los demás argumentos.

```
[PHP]

;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; PHP's initialization file, generally called php.ini, is responsible for
; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a number of locations.
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of PHP 5.2.0)
; 3. A number of predefined registry keys on Windows (As of PHP 5.2.0)
; 4. Current working directory (except CLI)
; 5. The web server's directory (for SAPI modules), or directory of PHP
; (otherwise in Windows)
; 6. The directory from the --with-config-file-path compile time option, or the
; Windows directory (C:\windows or C:\winnt)
; See the PHP docs for more specific information.
; http://php.net/configuration.file

; The syntax of the file is extremely simple.  Whitespace and Lines
; beginning with a semicolon are silently ignored (as you probably guessed).
; Section headers (e.g. [Foo]) are also silently ignored, even though
; they might mean something in the future.

; Directives following the section heading [PATH=/www/mysite] only
; apply to PHP files in the /www/mysite directory.  Directives
; following the section heading [HOST=www.example.com] only apply to
; PHP files served from www.example.com.  Directives set in these
; special sections cannot be overridden by user-defined INI files or
; at runtime. Currently, [PATH=] and [HOST=] sections only work under
; CGI/FastCGI.
; http://php.net/ini.sections
```

Figura 1. Configuración de las opciones de **MySQL** en el archivo **`php.ini`**.

PHP tiene valores por defecto (no incluidos en el **php.ini**) que se utilizan cuando un argumento fue omitido en la llamada a **mysql_connect** y, además, no tiene un valor asociado en el **php.ini**. Estos valores son:

- **Nombre del servidor**: **localhost**
  (que es equivalente a escribir **127.0.0.1**).
- **Nombre de usuario**: propietario del servidor.
- **Password**: vacío.

Si programamos de forma local, por ejemplo, para realizar pruebas de desarrollos, no hay mayores inconvenientes en completar los valores por

defecto del **php.ini**, pero tengamos en cuenta que cualquiera que tenga acceso a ese archivo podrá ver esa información. Incluso a través de PHP se puede utilizar la función **get_cfg_var("mysql.default_password")** y ver la contraseña escrita allí. Esta función devuelve **1** (verdadero) si nos conectamos y **0** (falso) si algo falló. Cuando se logra la conexión, devuelve, además, un **identificador de conexión**. Por ejemplo, en la siguiente conexión, **$id_conexion** sería el identificador de conexión:

```
$id_conexion = mysql_connect("168.22.22.3", "pepe", "grillo");
```

Uno de los argumentos de **mysql_connect** es el servidor al que nos queremos conectar (allí estarán guardadas las bases de datos), y en el cual deberá estar instalado y activo el servidor de bases de datos.

mysql_pconnect

Funciona de manera similar a **mysql_connect** con la diferencia de que las conexiones abiertas con **mysql_pconnect** (la **p** significa persistente) no se cierran –persisten– cuando termina la ejecución del archivo PHP (con **mysql_connect** sucede esto). Si al llamar a esta función ya existía una conexión abierta idéntica, se utiliza esa conexión, de lo contrario, se la crea. La calificación **idéntica** quiere decir que los argumentos

TRABAJAR CON BASES DE DATOS

Recordemos que, para empezar a interactuar con una base de datos (es decir, a realizar consultas en el lenguaje SQL), debemos primero asegurarnos de que los servidores (el de páginas web y el de bases de datos) estén funcionando, y que la conexión se haya realizado con éxito (mediante una de las funciones **mysql_connect** o **mysql_pconnect**).

"servidor", "nombre de usuario" y "password" son iguales en las dos conexiones, tanto la que se intenta abrir como en la que está abierta.

Esta función, al igual que **mysql_connect**, también devuelve un identificador de conexión. Su sintaxis es la siguiente:

```
mysql_pconnect("servidor", "nombre de usuario", "password");
```

Ejecución de sentencias

Luego de crear la instrucción SQL, hay que enviarla al servidor para que este la resuelva. PHP nos ofrece dos funciones para hacerlo: **mysql_db_query** y **mysql_query**.

mysql_db_query

Los argumentos que espera son:

```
mysql_db_query("base de datos", "instrucción", identificador);
```

- **base de datos**: nombre de la base de datos sobre la cual queremos realizar la consulta.
- **instrucción**: cadena que contiene la consulta SQL por realizar. Las instrucciones no pueden terminar con punto y coma (**;**).
- **identificador**: un identificador de conexión. Este argumento es opcional y si se opta por no incluirlo, la función intentará encontrar una conexión abierta al servidor MySQL. Si no la encuentra, tratará de crear una como si se llamara a **mysql_connect()** sin argumentos.

Ejemplos de la función:

Consulta de selección enviando una variable de cadena:

```
<?php
$consulta_sql = "select * from alumnos";
mysql_db_query("base_ejemplo", $consulta_sql);
?>
```

Consulta de selección escribiéndola directamente en la petición a la base de datos:

```
<?php
mysql_db_query("base_ejemplo", "select * from alumnos");
?>
```

Esta función devuelve **1** (verdadero) si la instrucción pudo ejecutarse y **0** (falso) si algo falló:

```
<?php
$identificador = mysql_connect("168.22.22.3", "pepe", "grillo");

$consulta_sql = "insert into alumnos values (1, 'julian maidana')";

mysql_db_query("base_ejemplo", $consulta_sql, $identificador);
?>
```

Antes de enviar una instrucción SQL, debemos establecer una conexión con el servidor.

mysql_query

Su sintaxis es la siguiente:

```
mysql_query("instruccion", identificador);
```

- **instrucción**: cadena que contiene la consulta SQL por realizar. Las instrucciones no pueden terminar con punto y coma (**;**).
- **identificador**: un identificador de conexión. Este argumento es opcional y, si se opta por no incluirlo, la función asume la última conexión abierta con el servidor MySQL. Si no la encuentra, intentará crear una como si se llamara a **mysql_connect()** sin argumentos.

Esta función devuelve **1** (verdadero) si la instrucción pudo ejecutarse con éxito y **0** (falso) si algo falló. Antes de enviar una instrucción SQL, debemos establecer una conexión con el servidor.

Creación de base de datos a través de PHP

Existen distintos programas para crear bases de datos. A continuación, abordaremos la manera de hacerlo a través de un **script PHP** con la función **mysql_create_db**.

mysql_create_db

La sintaxis para utilizar esta función es la siguiente:

```
mysql_create_db("base de datos", identificador);
```

¿QUÉ PASÓ?

Algunas instrucciones SQL no devuelven registro y no podemos saber si se llevaron a cabo correctamente. Para estos casos, las funciones **mysql_db_query** y **mysql_query** devuelven valores **0** o **1**, según se haya concretado lo requerido en la instrucción o haya fallado.

- **base de datos**: es el nombre de la base de datos que vamos a crear. Si ese nombre ya existe o el identificador no es válido, la función devuelve **0** (falso) y, obviamente, no crea ninguna base.
- **identificador**: un identificador de conexión. Este argumento es opcional y, si se opta por no incluirlo, la función asume la última conexión abierta con el servidor MySQL. Si no la encuentra, intentará crear una como si se llamara a **mysql_connect()** sin argumentos.

Para más información acerca de identificadores de conexión y cómo conectarse al servidor, veremos **mysql_connect** en este mismo capítulo.

Esta función devuelve **1** (verdadero) si la instrucción pudo ejecutarse con éxito y **0** (falso) si algo falló. Debemos tener en cuenta que esta función no está disponible en todos los servidores y que, en su lugar, podemos utilizar la función **mysql_query**, tal cual veremos en el ejemplo presentado hacia el final de este mismo capítulo. Ejemplo de la función **mysql_create_db**:

```
<?php
$con = mysql_connect("localhost", "u", "p");
$bas = mysql_create_db("mibase");
//en este caso se crea la base utilizando el
//identificador $con, que fue el último abierto.
?>

<?php

$con1 = mysql_connect("198.92.34.1", "u", "p");
$con2 = mysql_connect("localhost", "u", "p");
$bas = mysql_create_db("mibase", $con1);
//en este caso se crea la base utilizando el
//identificador $con1.

?>

<?php
```

```
$link = mysql_connect("localhost", "u", "p");
if (mysql_create_db ("mibase2"))                    {
   //la funcion mysql_create_db devuelve 1
   print("base de datos creada\n");
} else
                                        {
   //la funcion mysql_create_db devuelve 0
   print("no pude crear la base de datos\n");
}
//utilizando un condicional podemos saber
si accedimos o no a la base de datos
?>
```

Cada vez que se crea una base, se almacenará normalmente en el directorio **mysql\datas\nombre_base**, **mysql**, que es el directorio en donde se instaló MySQL por defecto.

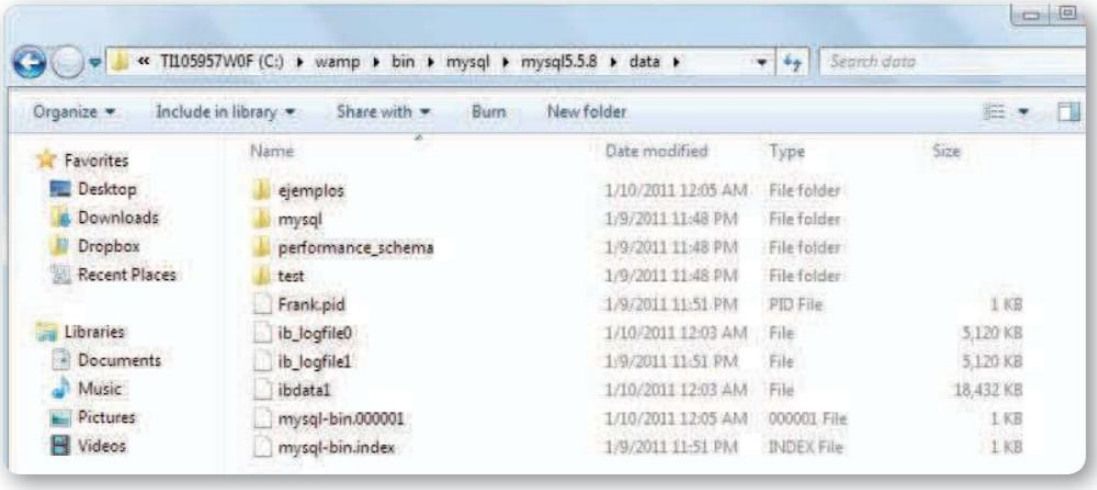


Figura 2. Bases de datos almacenadas en el servidor.

Para eliminar una base de datos desde PHP, puede usarse la función **mysql_drop_db**:

```
mysql_drop_db("base de datos", identificador);
```

Debemos tener cuidado al eliminar una base de datos, ya que en algunos casos no podremos recuperar el contenido.

Selección de una base de datos

Normalmente, en un servidor hay almacenadas más de una base de datos. Por eso, debemos indicar cuál de todas ellas vamos a utilizar en nuestras consultas. PHP nos provee para ello de una función llamada **mysql_select_db**.

mysql_select_db

Su sintaxis es la siguiente y los parámetros son iguales a los utilizados en **mysql_create_db**:

```
mysql_select_db("base de datos", identificador);
```

Algo importante es que esta función **liga o enlaza** a la base de datos seleccionada y al identificador. Esto significa que, cuando por medio de cualquier función hagamos referencia al identificador, se tomará como base de datos la seleccionada y no habrá que volver a indicarla cada vez que deseemos utilizarla.

NO ES EL ÚNICO

Si bien SQL (**Structured Query Language**) es el lenguaje de consulta de bases de datos más utilizado de la actualidad, existen otros, como **QBE** (**Query By Example**) y **QUEL (Query Language)**, que intentan competir con él aunque no lo logren, al menos por ahora.

```
<?php
$link = mysql_connect("localhost", "u", "p");
$sel_base = mysql_select_db("base1");
?>
```

Esta función devuelve **true** (verdadero) si tuvo éxito y **false** (falso) si algo falló (por ejemplo, si no hubiese ninguna base de datos con ese nombre).

Creación de tablas a través de PHP

En PHP no existe una función específica para crear tablas. Para hacerlo, debemos utilizar las funciones **mysql_db_query** o **mysql_query**, explicadas anteriormente en este capítulo, y escribir la instrucción SQL que corresponda. Veamos algunos ejemplos:

```
<?php
//cargamos en una variable toda la consulta
$tabla =  "create table t1 (";
$tabla .= "c1 int(4) unsigned not null,";
$tabla .= "c2 varchar(100))";
$tabla .= "ENGINE=myisam";

//la ejecutamos
if (mysql_query($tabla)) {
        echo "tabla creada";
} else                                          {
        echo "error al crear la tabla";
}

?>
```

Para eliminar una tabla podemos hacer lo siguiente:

```
<?php
//creamos la consulta
$borrar_tabla = "drop table t1";

//la ejecutamos
if (mysql_query($borrar_tabla))   {
        echo "tabla eliminada";
} else                                          {
        echo "error al eliminar la tabla";
}
?>
```

Para modificar la estructura de la tabla hacemos lo siguiente:

```
<?php
//creamos la consulta
$sql = "ALTER TABLE t1 MODIFY c2 BIGINT NOT NULL";

//la ejecutamos
if (mysql_query($sql))   {
        echo "tabla modificada";
} else                                          {
        echo "error al modificar la tabla";
}
?>
```

Por lo general, un error en este tipo de consultas se debe a una sintaxis incorrecta en la instrucción SQL, o bien, a un problema al establecer la conexión al servidor. Usualmente veremos en nuestros programas una gran cantidad de consultas a una base de datos; por esta razón, aunque sea en las primeras aplicaciones, sería conveniente trabajar como en el último ejemplo, es decir, usar mensajes de confirmación de tareas. Esto permite saber dónde está el error cometido y además resolverlo de manera rápida.

Obtener listados de bases de datos y tablas a través de PHP

Para saber qué bases de datos hay en el servidor, PHP nos ofrece la función **mysql_list_dbs** y, para conocer qué tablas hay, la función **mysql_list_tables**.

mysql_list_dbs

La sintaxis de la función es:

```
mysql_list_dbs(identificador);
```

- **identificador**: un identificador de conexión. Este argumento es opcional, y, si se opta por no incluirlo, la función asume la última conexión abierta con el servidor MySQL. Si no la encuentra, intentará crear una como si se llamara a **mysql_connect()** sin argumentos. Esta función devuelve un puntero de resultado que contiene las bases disponibles en relación con la actual conexión (**identificador**).

Antes de continuar, explicaremos la función **mysql_tablename.**

```
mysql_tablename(lista, i);
```

PHPMYADMIN

Para darse una idea de hasta dónde se puede llegar utilizando en conjunto **PHP** y **MySQL,** podemos poner como ejemplo una aplicación como **PHPMyAdmin**, que, más allá de ser utilizada por un gran número de usuarios, es realmente muy completa y detallada en lo que respecta a sus funcionalidades.

Esta función toma dos argumentos: el primero es el puntero que indica el primer elemento de la lista de bases de datos devuelta por la función **mysql_list_dbs**; el segundo es un índice para recorrer esa lista.

Puede resultar confuso, pero esta función –a pesar de su nombre– nos sirve tanto para recorrer y obtener los nombres de las tablas de una lista, como también para obtener los nombres de las bases de datos de una lista, según se use **mysql_list_dbs** o **mysql_list_tables** (explicada más adelante). Veamos el siguiente código de ejemplo y su explicación:

```
<?php

$res = mysql_list_dbs($link);
$i = 0;
while ($i < mysql_fetch_row($res))  {
echo mysql_tablename($res, $i);
echo "<BR />";
$i++;
}

?>
```

- La tercera línea guarda en **$res** el puntero correspondiente a la lista de las bases de datos disponibles en el servidor MySQL, utilizando la conexión **$link** (definida previamente).

LA VIDA DESPUÉS DE INTERNET

Las bases de datos no son propiedad exclusiva de internet: su ámbito depende, como casi todas las cosas, de qué queramos hacer con ellas. Tengamos presente este concepto a la hora de elegir el lugar en donde guardar los datos, puede que no siempre se precise utilizar las prestaciones de un servidor.

- La cuarta línea inicializa la variable **$i**.
- La quinta línea recorre la lista de bases de datos utilizando la función **mysql_fetch_row**.
- La sexta línea, mediante la función **mysql_tablename,** muestra el nombre de la base de datos correspondiente a la posición **$i** de la lista.
- La octava línea incrementa **$i** para pasar a la siguiente base de datos (si es que la hay).

En lugar de **mysql_fetch_row** podrían haberse usado otras funciones, como, por ejemplo, **mysql_fetch_object**:

```
<?php
$res = mysql_list_dbs($link);
while ($i = mysql_fetch_object($res)) {
   echo $i->Database."<br />";
}
?>
```

Cada vez que se llama a **mysql_fetch_object**, se obtiene un elemento de la lista y se lo asigna a **$i**. De ese elemento (que se trata como objeto gracias a **mysql_fetch_object**), tomamos la propiedad **Database** (que representa el nombre) y la mostramos.

CREACIÓN DE BASES Y TABLAS I

Si bien es posible crear tanto bases de datos como tablas desde PHP, también podemos hacerlo desde aplicaciones dedicadas a trabajar con MySQL. La ventaja de trabajar de esta forma es que estas aplicaciones nos brindan posibilidades avanzadas y estandarizadas para mantener bases de datos. Además, podremos ejecutarlas sin tener instalado PHP.

mysql_list_tables

Cumple la misma función que **mysql_list_dbs**, pero en vez de devolver un puntero al primer elemento de una lista de bases de datos, lo devuelve al primer elemento de una lista de tablas de una base de datos. Aquí también es posible usar la función **mysql_tablename** explicada en **mysql_list_dbs**:

```
mysql_list_tables(base de datos, identificador);
```

Veamos un ejemplo:

```
<?php
$res = mysql_list_tables("base1", $link);
$i = 0;
while ($i < mysql_fetch_row($res))  {
echo mysql_tablename($res, $i);
echo "<BR />";
$i++;
}
?>
```

Uso de múltiples bases de datos en una aplicación

Normalmente, en nuestras aplicaciones, accedemos a una base de datos por vez, pero PHP nos ofrece la posibilidad de hacerlo a varias al mismo tiempo. Veremos algunas premisas para tener en cuenta sobre el tema.

Cuándo hacerlo

Hay ocasiones en las que, quizás por temas de seguridad o simplemente de lógica de negocios, una empresa guarda sus datos en distintas bases. Por ejemplo, podría suceder que el sector de contaduría y el de

recursos humanos almacenaran sus datos en distintos lugares y que un tercer sector precisara datos de los dos sectores anteriores para realizar su tarea (por ejemplo, para liquidación de sueldos y jornales).

También, podría darse el caso de dos o más empresas distintas que trabajan en un proyecto en común y necesitan acceder a datos de forma concurrente. En definitiva, es una cuestión de diseño del sistema de una empresa (o varias) y se da con bastante frecuencia.

Cómo hacerlo

Acceder a dos o más bases de datos desde PHP es, por cierto, muy sencillo e intuitivo si sabemos cómo acceder a una por vez. A continuación, daremos algunos ejemplos:

```
<?php
//ejemplo con dos bases de datos que están guardadas
//en diferentes servidores (net1 y 127.0.0.1)

//establecemos las conexiones
$con1 = mysql_connect("net1", "w", "x");
$con2 = mysql_connect("127.0.0.1", "y", "z");

//consulta a una tabla de la base de datos base1
//que esta guardada en "net1"
mysql_select_db("base1", $con1);
$sql = "select * from productos";
mysql_db_query("base1", $sql, $con1);

//consulta a una tabla de la base de datos base2
//que esta guardada en "127.0.0.1"
mysql_select_db("base2", $con2);
$sql = "select * from clientes";
mysql_db_query("base2", $sql, $con2);
```

```
?>

<?php
//ejemplo con dos bases de datos que están guardadas
//en el mismo servidor (abracadabra)

//establecemos las conexiones
$con1 = mysql_connect("abracadabra", "u", "p");
$con2 = mysql_connect("abracadabra", "y", "z");

//consulta a una tabla de la base de datos base1
//que esta guardada en "abracadabra"
mysql_select_db("base1", $con1);
$sql = "select * from productos";
mysql_db_query("base1", $sql, $con1);

//consulta a una tabla de la base de datos base2
//que esta guardada en "abracadabra"
mysql_select_db("base2", $con2);
$sql = "select * from clientes";
mysql_db_query("base2", $sql, $con2);

?>
```

Como se ve, la forma de trabajo es muy parecida a la que usamos al trabajar solo con una base: lo único que hay tener presente es utilizar la función **mysql_select_db** cada vez que cambiemos de base. También recordemos que la declaración de variables tiene que ser comprensible en el momento de desarrollar, y no debemos mezclar o crear nombres similares.

Cerrar conexión con la base de datos

Para cerrar una conexión, utilizamos la función **mysql_close()**. Tengamos en cuenta que se cerrará la conexión o no, de acuerdo con la manera en que fue abierta (ver funciones **mysql_connect** y **mysql_pconnect**).

mysql_close

Si abrimos la conexión con **mysql_connect**, podemos cerrarla utilizando la función **mysql_close**, aunque debemos aclarar que este tipo de conexiones se cierran en forma automática al terminar de ejecutarse cada página PHP en la cual hayamos abierto una conexión previamente.

En cambio, la función no cerrará la conexión si esta fue abierta utilizando la función **mysql_pconnect**. Su sintaxis es la siguiente:

```
mysql_close(identificador);
```

- **identificador**: un identificador de conexión. Este argumento es opcional y, si se opta por no incluirlo, la función asume la última conexión abierta con el servidor MySQL.

```
<?php
$link = mysql_connect("localhost", "u", "p");

if ($link)      {
                echo "Conexión Abierta!\n";
} else          {
        echo "No puedo establecer la conexión\n";
}


        if ($link) {
                $cerrar = mysql_close($link);
                if ($cerrar)                    {
                echo "Conexión cerrada!\n";
        } else                          {
                echo "No pude cerrar la Conexión!\n";
        }
}
?>
```

Esta función devuelve **true** (verdadero) si tuvo éxito y **false** (falso) si algo falló (por ejemplo si el identificador no era válido o si no se había abierto ninguna conexión).

Ejemplo práctico: autos.php

Realizaremos una ejercitación aplicando las diferentes sentencias que hemos aprendido hasta aquí; para esto, podemos utilizar un bloc de notas o un entorno de desarrollo.

```
<html>
<head>
      <title>Autos.php!</title>
</head>
<body>

<?php
//se establece una conexión con MySQL
$link = mysql_connect("localhost", "j", "m") or die("Error en la conexión");

//se verifica si hay o no hay una base de datos llamada "autos"
$lista_bd = mysql_list_dbs($link);
while ($i = mysql_fetch_object($lista_bd)) {
   if ($i->Database == 'autos'){
     echo 'Ya existe una base con ese nombre.';
     exit;
   }
}

//se crea la base de datos
$crear_bd = mysql_query("create database
autos") or die("Error al crear la base");

//se selecciona la base
mysql_select_db("autos", $link);

//se verifica si hay o no hay una tabla llamada "modelo"
```

```
$lista_tablas = mysql_list_tables("autos");
$i = 0;
while ($i < mysql_fetch_row($lista_tablas))  {
        if ('modelo' == mysql_tablename ($lista_tablas, $i))  {
                echo 'Ya existe una tabla con ese nombre.';
                exit;
        }
        $i++;
}

//se arma la instrucción para crear una tabla
$sql = "CREATE TABLE modelo (";
$sql .= "cod_mod INT(4) UNSIGNED NOT NULL,";
$sql .= "desc_mod VARCHAR(100),";
$sql .= "PRIMARY KEY (cod_mod)";
$sql .= ") ENGINE=MYISAM";

//se intenta crear la tabla
if (mysql_query($sql))
        echo 'Base de datos y tabla creadas con exito';
else
        echo 'Error a crear la tabla \'modelo\';

//se cierra la conexión
mysql_close();
?>
</body>
</html>
```

CREACIÓN DE BASES Y TABLAS II

¿Por qué crear bases de datos o tablas desde PHP? Si distribuimos una aplicación a usuarios finales, esta se encargará de crear las diferentes estructuras para su funcionamiento.

Si la base de datos y la tabla no existían y se logran crear satisfactoriamente, se obtiene la salida que vemos en la **figura 3**.

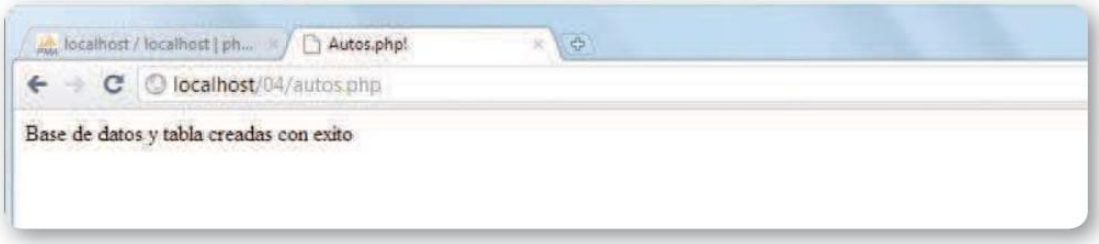


Figura 3. Salida de **`autos.php`**.

Entonces, podremos ver la nueva carpeta **autos**:

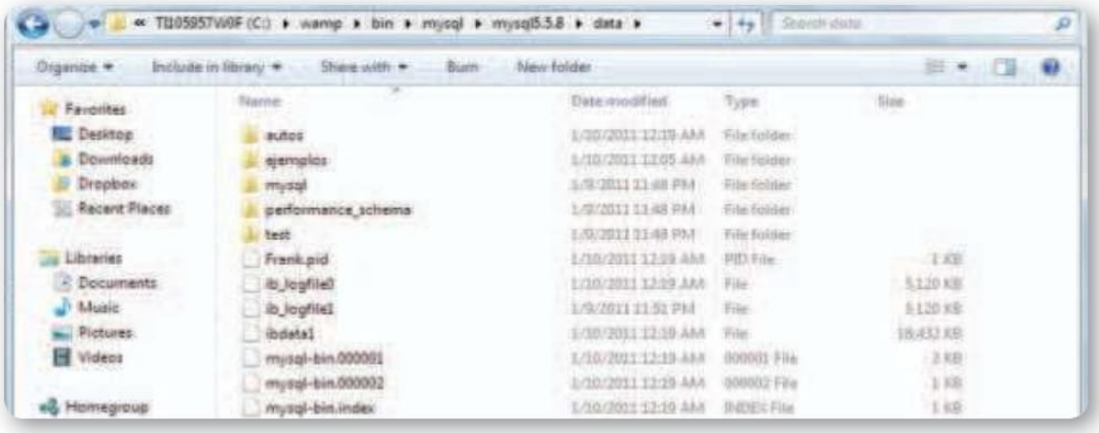


Figura 4. Bases de datos **`autos`** almacenada en el servidor.

RESUMEN

Hemos aprendido cómo ejecutar sentencias SQL para definir la estructura de una base de datos en MySQL: crear y eliminar bases de datos y tablas, establecer conexión y desconexión, cómo listar tablas y bases de datos disponibles en el servidor, utilizar una o varias bases.

En el próximo capítulo nos alejaremos de las estructuras y las conexiones, y empezaremos a manipular los datos de la base.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuál es la diferencia entre **mysql_connect** y **mysql_pconnect**?
2. Cuando nos conectamos a un servidor de bases de datos, ¿qué significa poner como nombre de servidor **localhost**? ¿Qué diferencia hay entre **localhost** y **127.0.0.1**?
3. ¿Qué es un identificador de conexión?

EJERCICIOS PRÁCTICOS

1. En el código del archivo **Cap4_practica1**, ¿qué conexiones (si es que hay alguna) quedan abiertas?
2. Se quiere crear la tabla **t1**. En el código del archivo **Cap4_practica2**, suponiendo que la conexión con el servidor sea válida y la base de datos se haya creado satisfactoriamente, ¿qué error se comete?, ¿qué falta?

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: **profesor@redusers.com**.

*05

Utilizar SQL en PHP

En este capítulo, haremos un recorrido por las distintas funciones que PHP nos ofrece para realizar consultas de selección y de modificación (el ingreso, la baja y la actualización de registros).

- Consultas de selección.....128
- Recorrer las filas devueltas de una consulta.....129
- Moverse entre registros.....142
- Número de registros y campos devueltos.....144
- Insert.....148
- Delete.....150
- Update.....152
- Ejemplo práctico: libros.php ...152
- Resumen.....167
- Actividades.....168

Consultas de selección

El lenguaje de consulta SQL es sintético, muy potente y utilizado por distintos gestores de bases de datos. Cuando queremos realizar consultas de selección (esto es, recuperar registros de una o más tablas), podemos obtener millones de registros con solo una línea de código.

Veremos las distintas posibilidades que existen para extraer datos y luego trabajar con ellos.

Selección simple

Nos referimos a consultas a una sola tabla por vez. Este tipo de consultas son poco frecuentes en sistemas que tienen un diseño complejo.

```
SELECT * FROM TABLA1;
```

El asterisco * tomará todos los valores que contiene la **TABLA1**.

Consultas multitabla

Son consultas que implican varias tablas a la vez, o sea, que recuperan datos de varias fuentes. Son muy comunes. A lo largo del capítulo, veremos cómo solucionar el problema que representa tener dos o más columnas de varias tablas diferentes con el mismo nombre.

```
SELECT * FROM TABLA1, TABLA2 WHERE TABLA1.A = TABLA2.B;
```

En esta consulta, tomaremos todos los valores de las **TABLA1** y **TABLA2**, pero solamente las que cumplan con la condición **TABLA1.A = TABLA2.B**.

Subconsultas

Las subconsultas se implementaron en MySQL a partir de la versión 4.1, lo cual potencia el motor de base de datos para selecciones de datos complejas.

```
SELECT * FROM TABLA1 WHERE TABLA1.A =
(SELECT B FROM TABLA2 WHERE C > 3);

//compara todos los valores de A con los valores que
//devuelve la segunda consulta.
```

Si el motor de base de datos no soporta subconsultas, no debemos preocuparnos, son perfectamente reemplazables: solo tendremos que realizar en lugar de una consulta, dos o más.

Recorrer las filas devueltas de una consulta

Al realizar una consulta a una base de datos, PHP permite guardar los resultados en forma de objetos o también de arrays (matrices) para acceder a ellos utilizando índices, ya sea numéricos o alfanuméricos. A continuación, haremos un recorrido por las distintas posibilidades.

mysql_fetch_array

Esta función nos permite recuperar filas y acceder a los valores de cada campo como si se tratase de una matriz. Su sintaxis es la siguiente:

```
mysql_fetch_array(id_consulta, tipo_indice);
```

Cuando se envía una consulta al servidor (por ejemplo, con **mysql_query** o **mysql_db_query**) de bases de datos, el resultado devuelto por alguna de estas funciones se denomina **id_consulta**.

Para recuperar los datos devueltos por la consulta (si es que los hay), una de las funciones que podemos utilizar es **mysql_fetch_array**, a la cual se le pasa como argumento el identificador de la consulta (en este caso **id_consulta**).

Cada vez que se llama a **mysql_fetch_array**, el puntero interno avanza una fila. La primera vez que se la llama, se posiciona en la primera fila (si es que hay primera fila). En el ejemplo que presentamos a continuación cada vez que se llama a la función, se asignan a la matriz **$fila** los campos de la consulta **$sql**. Cuando no haya más filas, la matriz toma el **valor** FALSE y el bucle **WHILE** termina.

```
<?php
$sql = "select * from tabla";
$res = mysql_query($sql);
while ($fila = mysql_fetch_array($res))    {
    ......
}
?>
```

Como esta función trata a cada fila como si fuera una matriz, el argumento **tipo_índice** permite definir el tipo de índice. Las opciones son **MYSQL_ASSOC**, **MYSQL_NUM** y **MYSQL_BOTH**.

MATRIZ DEVUELTA POR MYSQL_FETCH_ARRAY

La función **mysql_fetch_array** devuelve un array que contiene los valores de cada registro devuelto por una consulta SQL. Este array es tratado por el lenguaje PHP como cualquier otro, y es posible aplicar sobre él funciones como **count**, **in_array**, **sort** y otras.

Este argumento es opcional y, si se decide no incluirlo, se asume por defecto **MYSQL_ASSOC**. Debe escribirse con letras mayúsculas.

MYSQL_ASSOC

Se utiliza el nombre del campo (columna) como índice. Por ejemplo:

```
<?php
$sql = "select c1, c2 from alumnos";

$res = mysql_query($sql);

while ($fila = mysql_fetch_array($res, MYSQL_ASSOC))    {
    echo $fila["c1"];
    echo "<br />";
    echo $fila["c2"];
    echo "<br />";
}
?>

<?php
$sql = "select * from alumnos";

$res = mysql_query($sql);

while ($fila = mysql_fetch_array($res))    {
    echo $fila["c1"];
    echo "<br />";
    echo $fila["c2"];
    echo "<br />";
}
?>
```

Los dos ejemplos anteriores realizan la misma acción de recorrer la consulta mientras se cumpla la condición. Es importante destacar que los nombres de campos son sensibles a mayúsculas y minúsculas, no es lo mismo **$fila["campo1"]** que **$fila["Campo1"]**. El modo correcto de escribir los nombres de los campos dependerá de cómo los hayamos definido en el momento de crear las tablas.

MYSQL_NUM

Se usa el número del campo (columna) como índice, según el orden dado en la consulta SQL. El índice **0** corresponde al primer campo. Por ejemplo:

```
<?php
$sql = "select FFF, XXX, YYY from alumnos";

$res = mysql_query($sql);

while ($fila = mysql_fetch_array($res, MYSQL_NUM))    {
    echo $fila[0];    //muestra FFF
    echo "<br />";
    echo $fila[2];    //muestra YYY
    echo "<br />";
    echo $fila[1];    //muestra XXX
    echo "<br />";
}
?>
```

DISEÑAR LAS APLICACIONES

Antes de escribir un programa, es conveniente dividirlo en funciones. Esto nos ayudará a desarrollar aplicaciones claras, legibles y entendibles no solo para nosotros.

Si se seleccionan todos los campos, se tendrá en cuenta el orden dado en la creación de la tabla. Por ejemplo:

```
<?php
//    estructura de la tabla alumnos:

//        CREATE TABLE t1 (
//            XXX INT NOT NULL,
//            FFF VARCHAR(12),
//            YYY TEXT)
//        ) ENGINE=MYISAM;

$sql = "select * from alumnos";

$res = mysql_query($sql);

while ($fila = mysql_fetch_array($res, MYSQL_NUM))    {
    echo $fila[0];    //muestra XXX
    echo "<br />";
    echo $fila[2];    //muestra YYY
    echo "<br />";
}
?>
```

Es lo mismo escribir **$fila[0]** que **$fila["0"]** o **$fila['0']**.

MYSQL_BOTH

Permite utilizar el número o el nombre del campo (columna) como índice. Es como utilizar **MYSQL_ASSOC** y **MYSQL_NUM** al mismo tiempo. Por ejemplo:

```
<?php
$sql = "select FFF, XXX, YYY from alumnos";
$res = mysql_query($sql);

while ($fila = mysql_fetch_array($res, MYSQL_BOTH))    {
    echo $fila[0];    //muestra FFF
    echo "<br />";
    echo $fila[2];    //muestra YYY
    echo "<br />";
    echo $fila[1];    //muestra XXX
    echo "<br />";

    echo $fila["FFF"];    //muestra FFF
    echo "<br />";
    echo $fila["YYY"];    //muestra YYY
    echo "<br />";
    echo $fila["XXX"];    //muestra XXX
    echo "<hr>";

}
?>
```

Los nombres de campos también son sensibles a mayúsculas. La forma correcta para escribir los nombres de los campos dependerá de cómo los hayamos definido en el momento de crear las tablas.

Si realizamos una consulta sobre más de una tabla, y por lo menos dos de los campos recuperados tienen el mismo nombre, solo tendrá validez la última columna. Ejemplo de esta situación y posibles soluciones pueden encontrarse en **mysql_fecth_assoc**, en este mismo capítulo.

mysql_fetch_row

Esta función es como utilizar **mysql_fetch_array** con el argumento **tipo_indice** igual a **MYSQL_NUM**, es decir, toma el número del campo (columna) como índice, según el orden dado en la consulta SQL. Por ejemplo:

```
<?php
$sql = "select a, b, c from tabla1";
$res = mysql_query($sql);

while ($f = mysql_fetch_row($res))    {
    echo $f[0];    //muestra a
    echo "<br />";
    echo $f[2];    //muestra c
    echo "<br />";
}
?>
```

La llamada a **mysql_fetch_row** devuelve la próxima fila del resultado (si es que hay una próxima) o **falso** si no quedan más filas. Para más información, ver **mysql_fetch_array** y **MYSQL_NUM**.

mysql_fetch_object

Esta función trata a cada fila como si fuera un objeto y a cada campo como si fuera una propiedad del objeto fila.

```
mysql_fetch_object(id_consulta, tipo_indice);
```

ÍNDICE DE LAS COLUMNAS

Es un error bastante común pensar que la función **mysql_num_rows** toma como índice el número de columna según el orden de creación en la instrucción **SQL create table**. Esto no es así, no importa el orden de creación de la columna, sino el orden en el que figura en la consulta de selección que se envía a MySQL –generalmente a través de **mysql_query**–, cuyo resultado es tomado por esta función.

Para recuperar los datos devueltos por la consulta (si es que los hay), una de las funciones que se puede utilizar es **mysql_fetch_object**, a la cual se le pasa como argumento el identificador de la consulta (en este caso **id_consulta**).

Las opciones para el argumento **tipo_indice** son otra vez **MYSQL_ASSOC**, **MYSQL_NUM** y **MYSQL_BOTH**. El argumento es opcional y, si se opta por no incluirlo, se asume por defecto **MYSQL_ASSOC**. Debe escribirse con letras mayúsculas. Por ejemplo, si tenemos la tabla:

NBA			
Código	**Nombre_y_Apellido**	**Nacionalidad**	**Estado_civil**
001	Allen Iverson	EE. UU.	Casado
004	Emanuel Ginóbili	Argentina	Casado
005	Eduardo Nájera	México	Soltero
006	Arvidas Sabonis	Lituania	Soltero

Podríamos acceder a los valores de cada campo de la siguiente forma:

fila1 - codigo = 001
fila2 - nombre_y_apellido = Emanuel Ginobili
fila3 - codigo = 005
fila2 - nacionalidad = Argentina

TOMEMOS EL CONTROL

Recomendamos modificar los ejemplos desarrollados en este libro, ampliarlos en funcionalidades. Esto nos ayudará a comprender y asimilar lo visto de manera rápida y segura.

En PHP sería:

```
<?php
$sql = "select * from NBA";
$res = mysql_query($sql);

while ($f = mysql_fetch_object($res))    {
    echo $f->codigo;
    echo "<br />";
    echo $f->nombre_y_apellido;
    echo "<br />";
    echo $f->nacionalidad;
    echo "<br />";
}
?>
```

mysql_fetch_assoc

Esta función es como utilizar **mysql_fetch_array** con el argumento **tipo_indice** igual a **MYSQL_ASSOC**, es decir que toma el nombre del campo (columna) como índice. Si realizamos una consulta sobre más de una tabla y por lo menos dos de los campos recuperados tienen el mismo nombre, solo tendrá validez la última columna. Por ejemplo, si tenemos las siguientes tablas:

Código1	Nombre	Edad
1	Homero	38
4	Marge	37
5	Lisa	7
6	Bart	8

Código2	Nombre	Habitantes
1	Filadelfia	1200000
4	Nueva York	4000000
5	Phoenix	2000000
6	Seattle	300000

En PHP sería:

```
<?php
$sql = "select Codigo1, TABLA1.Nombre,     Tabla2.Nombre,
Edad from TABLA1, TABLA2 where     Codigo1=Codigo2";

$res = mysql_query($sql);

while ($f = mysql_fetch_assoc($res))    {
    echo $f["Codigo1"]." - ";
    echo $f["Nombre"]."<br />";
}

// la salida sera:
//    1 - Philadelfia
//    4 - New York
//    5 - Phoenix
//    6 - Seattle
?>

<?php
$sql = "select Codigo1, TABLA2.Nombre,     Tabla1.Nombre,
Edad from TABLA1, TABLA2 where     Codigo1=Codigo2";

$res = mysql_query($sql);
```

```
while ($f = mysql_fetch_assoc($res))    {
    echo $f["Codigo1"]." - ";
    echo $f["Nombre"]."<br />";
}

// la salida será:
//    1 - Homero
//    4 - Marge
//    5 - Lisa
//    6 – Bart
?>
```

Para acceder a las dos columnas con el mismo nombre al mismo tiempo, tenemos al menos dos posibilidades:

- Usar índices numéricos, como se explicó en **mysql_fetch_row**.
- Usar alias en las consultas. Por ejemplo:

```
<?php
$sql = "select Codigo1, TABLA2.Nombre as nom2, Tabla1.Nombre
as nom1, Edad from TABLA1, TABLA2 where Codigo1=Codigo2";

$res = mysql_query($sql);

while ($f = mysql_fetch_assoc($res))    {
    echo $f["Codigo1"]." - ";
    echo $f["nom1"]." - ";
    echo $f["nom2"]."<br />";
}

// la salida será:
//    1 - Homero - Philadelfia
//    4 - Marge - New York
//    5 - Lisa - Phoenix
//    6 – Bart - Seattle
?>
```

Los nombres de campos son sensibles a mayúsculas y minúsculas. La forma correcta de escribir los nombres de los campos dependerá de cómo los hayamos definido en el momento de crear las tablas.

Si necesitamos trabajar solo con índices numéricos o con índices numéricos y asociativos, utilizaremos las funciones **mysql_fetch_row** o bien, **mysql_fetch_array**.

mysql_fetch_field

Esta función no recupera los datos contenidos en una columna, pero permite obtener información acerca de la estructura del campo y trabajar con él como si fuera un objeto, de la siguiente manera:

```
$campo->name; //devuelve el nombre de la columna
```

Cada vez que se llama a **mysql_fetch_field**, devuelve el campo siguiente (que figura en la instrucción SQL). **name** es solo una propiedad disponible, el listado completo es el siguiente:

- **name**: nombre de la columna.
- **table**: nombre de la tabla a la que pertenece la columna.

VALOR MÁXIMO DE ALMACENAMIENTO

La función **mysql_fetch_field** permite obtener, entre otros datos, la longitud máxima de una columna, como se explica en este capítulo. Esto puede ser particularmente interesante en el momento de validar el ingreso de datos a la tabla en cuestión y no sobrepasar el valor máximo permitido de una columna. Eventualmente, podríamos preparar mensajes de error personalizados a partir de estos datos.

- **max_length**: longitud máxima de la columna.
- **not_null**: 1 si la columna no contiene un valor nulo, si no 0.
- **primary_key**: 1 si la columna es clave primaria, si no 0.
- **unique_key**: 1 si la columna es clave única, si no 0.
- **multiple_key**: 1 si la columna es clave no única, si no 0.
- **numeric**: 1 si la columna es numérica, si no 0.
- **blob**: 1 si la columna es un BLOB, si no 0.
- **type**: el tipo de dato de la columna.
- **unsigned**: 1 si la columna es **unsigned**, si no 0.
- **zerofill**: 1 si la columna es **zero-filled**, si no 0.

Es muy importante escribir los nombres de las propiedades en minúsculas. Encontraremos más información sobre tipos de datos (numeric, blob, otros) en los **capítulos 2** y **3**, donde también encontraremos información sobre **primary key**, **unique key**, nombres de columnas, nombres de tablas y demás. La forma de uso es la siguiente:

```
<?php
$sql = "select a, b, c from alumnos";
$res = mysql_query($sql);
while ($campo = mysql_fetch_field($res))    {
    echo $campo->name."<br />";
    echo $campo->table."<br />";
    echo $campo->type."<br />";
}

// En la primera llamada a mysql_fetch_field se
// mostraran las propiedades de la columna a, en la
// segunda de b, y en la tercera de la c.
?>
```

Moverse entre registros

Cuando usamos las funciones que nos provee PHP para recorrer las filas devueltas por una instrucción **SELECT** (esto es **mysql_fetch_array**, **mysql_fetch_object**, **mysql_fetch_assoc**, **mysql_fetch_row**), vemos que, ante cada llamada a estas funciones, se avanza al registro siguiente (si es que lo hay). Para volver hacia atrás o para posicionarnos en algún registro específico, existe la función **mysql_data_seek**.

mysql_data_seek

La sintaxis de la función es:

```
mysql_data_seek(id_consulta, numero_fila);
```

Al enviar una consulta al servidor (con **mysql_query** o con **mysql_db_query**, por ejemplo) de bases de datos, el resultado devuelto por alguna de estas funciones se denomina **id_consulta**. El argumento **numero_fila** es el número de fila a la cual queremos acceder. A la primera fila corresponde el valor 0.

Luego de llamar a la función, la primera llamada a cualquier función para recorrer las filas devueltas (**mysql_fetch_array**, **mysql_fetch_object**, **mysql_fetch_assoc**, **mysql_fetch_row**) devolverá la fila número **numero_fila**. Por ejemplo:

DEPURAR EL LENGUAJE

Recordemos verificar de manera detallada la sintaxis de las instrucciones SQL que utilizamos en las aplicaciones; es un error común referenciar en nuestras consultas nombres de tablas o campos inexistentes. Puede ser de ayuda probar las consultas en otros programas, como **PHPMyAdmin** o **MySQLFront**.

cod_s	nom_s
1	PHP
2	Perl
3	Ruby
4	Python

```
<?php
$sql = "select nom_s from tabla1";
$res = mysql_query($sql);

while ($fila = mysql_fetch_array($res)) {
    echo $fila["nom_s"]."<br />";
}

mysql_data_seek($res ,2);
$fila = mysql_fetch_array($res);
echo $fila["nom_s"]."<br />";

mysql_data_seek($res ,0);
$fila = mysql_fetch_array($res);
echo $fila["nom_s"]."<br />";

// imprime:
//    PHP
//    Perl
//    Ruby
//    Python
//    Ruby
//    PHP
?>
```

Devuelve verdadero si tiene éxito, falso si ocurrió algún error (por ejemplo, si el número de fila **numero_fila** no existe).

Número de registros y campos devueltos

PHP provee funciones para obtener el número de filas y campos devueltos luego de una consulta. Veremos una descripción de cada una de ellas.

mysql_num_rows

Esta función solo sirve para sentencias tipo **SELECT**. Devuelve el número de filas de un resultado. Su sintaxis es:

```
mysql_num_rows(id_consulta);
```

Por ejemplo:

```
<?php
$sql = "select * from tabla1";

$res = mysql_query($sql);

$numero_filas = mysql_num_rows($res);

echo "Numero de filas devueltas: ".$numero_filas;
?>
```

Si bien podríamos obtener el mismo resultado con la siguiente consulta:

```
select count(*) from tabla;
```

En muchos casos, haríamos el código más sencillo sin la necesidad de utilizar **mysql_num_rows**.

mysql_affected_rows

Esta función solo sirve para sentencias tipo **INSERT**, **UPDATE** o **DELETE** y devuelve el número de filas involucradas en la última instrucción SQL. Su sintaxis es la siguiente:

```
mysql_affected_rows(identificador);
```

El argumento **identificador** es un identificador de conexión. Es opcional y, si se opta por no incluirlo, la función intentará encontrar una conexión abierta al servidor MySQL. Por ejemplo, si tenemos la siguiente tabla:

cod_s	nom_s
1	Batman
2	Robin
3	Acertijo
4	Gatúbela
5	Alfred

MANTENER LOS DATOS EN MEMORIA

La función **mysql_data_seek** nos permite movernos entre registros. Además, puede ser útil para no llamar más de una vez a una de las funciones **mysql_fetch_array**, **mysql_fetch_object**, **mysql_fetch_assoc** o **mysql_fetch_row** con la misma consulta a la base de datos. Si mantiene la variable **id_consulta**, también mantiene los datos devueltos por ella, y se puede acceder a estos a través de **mysql_data_seek** o alguna de las funciones **mysql_fetch**.

```
<?php
$con = mysql_connect("168.22.22.3", "u", "p");

$sql = "update tabla1 set nom_s = '' where cod_s >  2";
$res = mysql_query($sql);

echo mysql_affected_rows($con);
//imprime 3
?>

<?php
$con = mysql_connect("168.22.22.3", "u", "p");

$sql = "delete from tabla1 where cod_s > 3";
$res = mysql_query($sql);

echo mysql_affected_rows($con);
//imprime 2
?>

<?php
$con = mysql_connect("168.22.22.3", "u", "p");

$sql = "insert into tabla1 values (6, 'Guason')";
$res = mysql_query($sql);

$sql = "insert into tabla1 values (7, 'Fierro')";
$res = mysql_query($sql);

echo mysql_affected_rows($con);
//imprime 1 (toma la última instrucción)
?>
```

Si la última sentencia fue un **DELETE** sin cláusula **WHERE**, esta función debería devolver el número total de registros que había en la tabla; sin embargo, devuelve cero.

mysql_num_fields

Esta función, en lugar de entregar el número de filas devueltas por la consulta, como hace **mysql_num_rows**, o modificadas, como hace **mysql_affected_rows**, nos permite obtener el número de campos (columnas) devueltos por la consulta. Su sintaxis es:

```
mysql_num_fields(id_consulta);
```

Por ejemplo:

```
<?php
//Devuelve el número de columnas de la tabla tabla1

$sql = "select * from tabla1";
$res = mysql_query($sql);

$numero_campos = mysql_num_fields($res);

echo "Numero de campos devueltos: ".$numero_campos;
?>

<?php
//Devuelve 3 (la consulta devuelve 3 campos: a, b, c)

$sql = "select a, b, c from tabla1";
$res = mysql_query($sql);

$numero_campos = mysql_num_fields($res);

echo "Numero de campos devueltos: ".$numero_campos;
?>
```

Insert

Podemos realizar ingresos en una tabla a través de las funciones **mysql_query** o **mysql_db_query**, pero PHP nos brinda otra función que puede ser muy útil cuando se trabaja con campos de autoincremento.

Campos autoincrementables

MySQL nos permite insertar registros en una tabla evitando de forma automática que un campo tome valores duplicados. Estos campos se llaman **autoincrementables** y se definen en el momento de crear una tabla.

Por ejemplo:

```
<?php
$sql  = "CREATE TABLE T1 (";
$sql .= "codigo INT NOT NULL AUTO_INCREMENT, ";
$sql .= "descripcion VARCHAR(100) NOT NULL, ";
$sql .= "PRIMARY KEY(codigo)";
$sql .= ")";

mysql_query($sql);

$sql = "INSERT INTO T1 (descripcion) VALUES ('positivo')";
mysql_query($sql);

$sql = "INSERT INTO T1 (descripcion) VALUES ('negativo)";
mysql_query($sql);

$sql = "INSERT INTO T1 (descripcion) VALUES ('neutro')";
mysql_query($sql);

//    la tabla T1 quedaría así:
//    1    positivo
//    2    negativo
//    3    neutro
?>
```

Los campos autoincrementables mantienen internamente el valor máximo, aunque lo borremos. Por ejemplo, teniendo en cuenta el ejemplo anterior:

```
<?php

$sql = "DELETE FROM T1 WHERE codigo = 3";
mysql_query($sql);

$sql = "INSERT INTO T1 (descripcion) VALUES ('no medible')";
mysql_query($sql);

//    la tabla T1 quedaría así:
//    1    lindo
//    2    feo
//    4    no medible
?>
```

mysql_insert_id

Esta función devuelve el valor (no el número de fila) del último campo autoincremental ingresado y debe llamarse inmediatamente después del ingreso. Su sintaxis es la siguiente:

¿CUÁL USAR?

Generalmente, la diferencia entre las funciones **mysql_fetch_array**, **mysql_fetch_object**, **mysql_fetch_assoc** y **mysql_fetch_row** es ínfima, por eso se recomienda usar la que más cómoda nos resulte, aunque **mysql_fetch_array** puede suplantar el funcionamiento de la mayoría de las demás funciones.

```
mysql_insert_id(identificador);
```

Por ejemplo, teniendo en cuenta el caso anterior:

```
<?php

$sql = "INSERT INTO T1 (descripcion) VALUES ('medible')";
mysql_query($sql);

//    la tabla T1 quedaria asi:
//    1    positivo
//    2    negativo
//    4    no medible
//    5    medible

echo "Ultimo valor de codigo : ".mysql_insert_id();

//devuelve 5
?>
```

Delete

No existen funciones específicas para borrar tablas o registros, o columnas de una tabla: todo esto se hace mediante las funciones **mysql_query** o la **mysql_db_query**. Observemos algunos ejemplos:

```
<?php
//borrar tablas
$sql = "DROP TABLE nombre_tabla";
mysql_query($sql);
```

```
//borrar registros de una tabla
$sql = "DELETE FROM nombre_tabla WHERE campo1 > 4";
mysql_query($sql);

//borrar columnas de una tabla
$sql = "ALTER TABLE nombre_tabla DROP COLUMN campo1";
mysql_query($sql);
?>
```

Aunque sí existe una función para borrar bases de datos completas, y es la función **mysql_drop_db**. Por ejemplo:

```
<?php
$cbas = mysql_create_db("mibase") or DIE("no se pudo crear la base");

echo "<br />base de datos creada";

$bbas = mysql_drop_db("mibase") or DIE("no se pudo eliminar la base");

echo "<br />base de datos eliminada";
?>
```

DEPURAR EL LENGUAJE

Entendamos la diferencia entre utilizar instrucciones para borrar tablas e instrucciones para borrar todos los datos –filas– de una tabla. En las primeras se borra la estructura completa y los datos que contiene, mientras que en la segunda se mantiene la estructura y se borran los datos.

Update

Al igual que para borrar datos o estructuras no existen funciones específicas, para modificar o actualizar estructuras de tablas o registros, se utilizan las funciones **mysql_query** o **mysql_db_query**. Por ejemplo:

```
<?php

$sql = "update tabla1 set campo = 10";
$mysql_query($sql);

$sql = "alter table tabla1 modify campo BIGINT NOT NULL";
$mysql_query($sql);

?>
```

Ejemplo práctico: libros.php

Para este ejemplo, definimos la siguiente estructura para la base de datos:

LIBRO		
#	COD_L	Código del libro
	NOM_L	Nombre del libro
FK	COD_A	Código del autor
	CANT_PAG_L	Cantidad de páginas del libro
	ISBN_L	ISBN del libro
	FOTO_L	Nombre de la imagen del libro
	PRECIO_L	Precio del libro

AUTOR_LIBRO		
#	COD_A	Código del autor
#	COD_L	Código del libro

AUTOR		
#	COD_A	Código del autor
	NOM_A	Nombre del autor
	APE_A	Apellido del autor
FK	COD_N	Código de la nacionalidad del autor
	FEC_NAC_A	Fecha de nacimiento del autor
	FEC_DEC_A	Fecha de deceso del autor

NACIONALIDAD		
#	COD_N	Código de nacionalidad
	DESC_N	Descripción de nacionalidad

CANTIDAD DE REGISTROS DEVUELTOS

En algunos ejemplos usamos la función **mysql_fetch_array** junto con la estructura de control **while** para recorrer las filas devueltas por una consulta. Si sabemos que la consulta devolverá solo un registro, no será necesario utilizar **while**; deberemos proceder del siguiente modo: **$fila = mysql_fetch_array($res)**; para luego trabajar sobre el array **$fila**.

Podemos visualizar las siguientes relaciones:

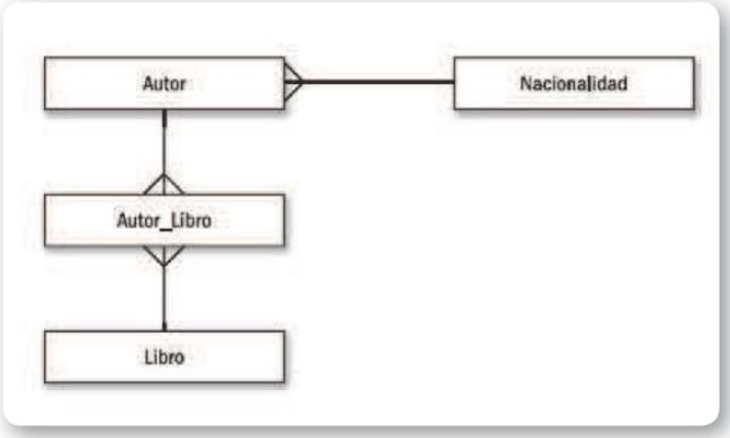


Figura 1. Relaciones entre tablas.

Como primer punto, vamos a definir el archivo **config.inc.php**, el cual contendrá las directivas para conectar a la base de datos:

```
<?php
//debe cambiar por los datos correctos de su
//equipo/configuración!

$servidor = "localhost";
$usuario = "root";
$password = "";
$nombre_base = "libros";

//se establece una conexión con MySQL
$link = mysql_connect($servidor, $usuario,
$password) or die("Error en la conexion");

//se selecciona la base
mysql_select_db($nombre_base, $link);
?>
```

Luego, vamos a definir el archivo **crear_estructura.php**, que nos permitirá trasladar a una base de datos el diseño visto anteriormente. Si no existe una base de datos llamada **libros** en el servidor, este archivo la creará. Si ya existe –o si se ejecuta este archivo dos veces– se le pedirá que modifique la variable **$nombre_base**, es decir, se creará otra base de datos (si optamos por esto último, se deberá actualizar en el archivo **config.inc** el nuevo nombre de la base).

Podríamos modificar el archivo **crear_estructura.php** para que, cada vez que se ejecute, borre la base de datos llamada **libros** –si es que existía– y, luego, la cree otra vez. Intentémoslo.

```
<?php

//cambie por los datos correctos !
$servidor = "localhost";
$usuario = "root";
$password = "";
$nombre_base = "libros";

//se establece una conexion con MySQL
$link = mysql_connect($servidor, $usuario,
$password) or die("Error en la conexion");

//se crea la base de datos
mysql_query("CREATE DATABASE $nombre_base") or die("Error al crear la
base $nombre_base (puede que ya exista). Modifique la variable \$nombre_base.");

//se selecciona la base
mysql_select_db($nombre_base, $link);

//se crean a continuacion las cuatro tablas
$sql = "CREATE TABLE libro (";
$sql .= "cod_l INT UNSIGNED NOT NULL,";
$sql .= "nom_l VARCHAR(100),";
$sql .= "cod_a INT UNSIGNED NOT NULL,";
```

```
$sql .= "cant_pag_l INT UNSIGNED,";
$sql .= "isbn_l VARCHAR(20),";
$sql .= "foto_l VARCHAR(100),";
$sql .= "precio_l FLOAT (5, 2),";
$sql .= "PRIMARY KEY (cod_l)";
$sql .= ") ENGINE=MYISAM";
mysql_query($sql);

$sql = "CREATE TABLE autor_libro (";
$sql .= "cod_a INT UNSIGNED NOT NULL,";
$sql .= "cod_l INT UNSIGNED NOT NULL,";
$sql .= "PRIMARY KEY (cod_a, cod_l)";
$sql .= ") ENGINE=MYISAM";
mysql_query($sql);

$sql = "CREATE TABLE autor (";
$sql .= "cod_a INT UNSIGNED NOT NULL AUTO_INCREMENT,";
$sql .= "nom_a VARCHAR(100),";
$sql .= "ape_a VARCHAR(100),";
$sql .= "cod_n TINYINT UNSIGNED NOT NULL,";
$sql .= "fec_nac_a DATE,";
$sql .= "fec_dec_a DATE,";
$sql .= "PRIMARY KEY (cod_a)";
$sql .= ") ENGINE=MYISAM";
mysql_query($sql);

$sql = "CREATE TABLE nacionalidad (";
$sql .= "cod_n TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,";
$sql .= "desc_n VARCHAR(50),";
$sql .= "PRIMARY KEY (cod_n)";
$sql .= ") ENGINE=MYISAM";
mysql_query($sql);

//insertamos algunos datos
mysql_query("INSERT INTO nacionalidad (desc_n) VALUES ('argentina')");
mysql_query("INSERT INTO nacionalidad (desc_n) VALUES ('belgica')");
mysql_query("INSERT INTO autor VALUES
```

```
(1, 'julio', 'cortazar', 2, '1912-11-12', '1984-1-10')");
mysql_query("INSERT INTO autor VALUES
(2, 'jorge luis', 'borges', 1, '1912-11-12', '1912-11-12')");
mysql_query("INSERT INTO autor VALUES (3, 'ernesto',
'sabato', 1, '1912-11-12', '1912-11-12')");

mysql_query("INSERT INTO libro VALUES (1,
'uno y el universo', 3, 320, 'isbn', '1.png', 23.90)");
mysql_query("INSERT INTO libro VALUES
(2, 'bestiario', 1, 150, 'isbn', '3.png', 13.90)");
mysql_query("INSERT INTO libro VALUES
(3, 'octaedro', 1, 120, 'isbn', '4.png', 16)");

mysql_query("INSERT INTO autor_libro VALUES (1, 2)");
mysql_query("INSERT INTO autor_libro VALUES (1, 3)");
mysql_query("INSERT INTO autor_libro VALUES (3, 1)");

echo 'Fin del script';

?>
```

En el archivo anterior, insertamos datos en las tablas. Lo hicimos desde PHP, pero también podemos hacerlo por otros medios (el monitor de MySQL, por ejemplo). Si todo funcionó bien, luego de ejecutar esta página en nuestro navegador, deberíamos obtener los siguientes resultados en nuestro monitor:

ACCESO A DATOS EN MEMORIA

La función **mysql_data_seek** permite posicionarnos sobre algún registro particular de los almacenados en un array de datos. Cada vez que llamamos a esta función, no hacemos una consulta al servidor, sino que trabajamos con los datos almacenados en memoria de la primera consulta, con lo que se obtiene una velocidad mayor de respuesta.

```
c:\wamp\bin\mysql\mysql5.5.8\bin\mysql.exe
Your MySQL connection id is 14
Server version: 5.5.8-log MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> use libros;
Database changed
mysql> show tables;
+-----------------+
| Tables_in_libros |
+-----------------+
| autor           |
| autor_libro     |
| libro           |
| nacionalidad    |
+-----------------+
4 rows in set (0.00 sec)

mysql> _
```

Figura 2. Estructura de la base de datos a través de MySQL.

Y tendríamos que poder ver el archivo de la nueva base creada:

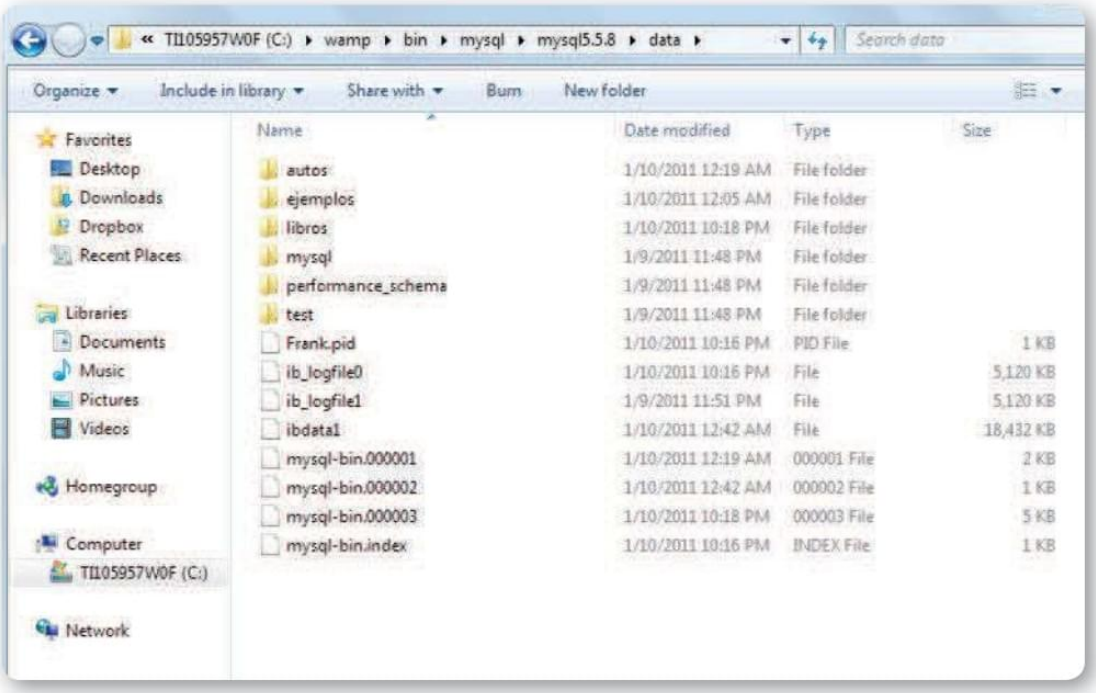


Figura 3. La base de datos creada, en el sistema de archivos.

A continuación, definimos el archivo **buscar.php**, que nos dará una interfaz para buscar libros de nuestra base:

```
<?php

if (!include("./config.inc.php"))        {
    echo "no encuentro el archivo config.inc.php !!!!";
    exit;
}

// select autores
$res = mysql_query("select cod_a, concat_ws(', ', ape_a, nom_a)
as nombre from autor order by ape_a asc, nom_a asc");
$select_autores = '<select id="cbo_autor" name="cbo_autor">';
$select_autores .= '<option value="0">Todos los autores</option>';
while ($fila = mysql_fetch_array($res)) {
    $select_autores .= '<option value = "'.$fila['cod_a'].'">'.
ucwords(strtolower($fila['nombre']).'</option>';
}
$select_autores .= '</select>';

// select libros
$res = mysql_query("select cod_l, nom_l from libro order by nom_l asc");
$select_libros = '<select id="cbo_libro" name="cbo_libro">';
$select_libros .= '<option value="0">Todos los Libros</option>';
while ($fila = mysql_fetch_array($res)) {
      $select_libros .= '<option value="'.$fila['cod_l'].'">'.
ucwords(strtolower($fila['nom_l']).'</option>';
}
$select_libros .= '</select>';

// select nacionalidad
$res = mysql_query("select cod_n, desc_n from na-
cionalidad order by desc_n asc");
$select_nacionalidad = '<select id="cbo_nacio-
nalidad" name="cbo_nacionalidad">';
$select_nacionalidad .= '<option value="0">Todas las Nacionalidades </option>';
while ($fila = mysql_fetch_array($res)) {
    $select_nacionalidad .= '<option value="'.$fila['cod_n'].'">'.
```

```
ucwords(strtolower($fila['desc_n'])).'</option>';
   }
   $select_nacionalidad .= '</select>';

   ?>
   <html>
   <head>
        <title>buscar.php</title>
            <style>
                    * {
                    font-family: Arial, Helvetica, sans-serif;
                    font-size: 13px;
                    font-weight: bold;
                    color: #57534c;
                    text-decoration: none;
                    line-height: 24px;
                    }

                    table {
                    border-top: 1px solid #57534c;
                    border-right: 1px solid #57534c;
                    }

                    td {
                    border-bottom: 1px solid #57534c;
                    border-left: 1px solid #57534c;
                    }
            </style>
   </head>

   <body>

   <table cellpadding="4" cellspacing="0">
     <tr>
       <td colspan="2">Buscador de Libros:</td>
     </tr>
```

```
  <tr>
    <td> </td>
    <td>
    <form action="mostrar_resultados.php" method="POST">
      <table cellpadding="4" cellspacing="0">
        <tr>
          <td>Seleccione Autor</td>
          <td>
          <?php echo $select_autores; ?>
          </td>
        </tr>
        <tr>
          <td>Seleccione Libro</td>
          <td>
          <?php echo $select_libros; ?>
          </td>
        </tr>
        <tr>
          <td>Ingrese I.S.B.N.</td>
          <td><input type="text" id="txt_isbn" name="txt_isbn"></td>
        </tr>
        <tr>
          <td>Seleccione Nacionalidad del Autor</td>
          <td>
          <?php echo $select_nacionalidad; ?>
          </td>
        </tr>
        <tr>
          <td>Ingrese Palabra Clave</td>
          <td><input type="text" id="txt_palabra_clave"
name="txt_palabra_clave"></td>
        </tr>
        <tr>
          <td> </td>
```

```
        <td><input type="submit" value="Buscar" id="submitBuscar"
name="submitBuscar"><input type="reset" value="Borrar"></td>
      </tr>
     </table>
   </form>
   </td>
  </tr>
  <tr>
   <td colspan="2">Complete los datos que conozca y presione
el botón &quot;Buscar&quot;</td>
  </tr>
</table>

</body>
</html>
```

Por último, el archivo **mostrar_resultados.php**, que nos permite mostrar los resultados de la búsqueda. Prestemos atención a la primera parte del script, en donde se crea la consulta para recuperar los libros devueltos y se guarda en la variable **$sql**.

Las variables recibidas mediante el array **$_POST** son los controles del formulario de búsquedas y una instrucción como la que sigue:

```
if ($_POST['nombre_control'])          {
    ......
    instrucciones
    ......
}
```

Se ejecutará, si **$_POST['nombre_control']** es distinto de 0 (distinto de vacío). Recordemos que, en el caso de los controles tipo **<select> </select>**, se tomará como valor el atributo **value** de la opción (**option**) elegida.

Aquí se ve el código de **mostrar_resultados.php**:

```
<?php

if (!include("./config.inc.php"))        {
    echo "no encuentro el archivo config.inc.php !!!!";
    exit;
}

$sql = " select * from autor, autor_libro, libro, nacionalidad";
$sql .= " where autor.cod_a = autor_libro.cod_a";
$sql .= " and libro.cod_l = autor_libro.cod_l";
$sql .= " and autor.cod_n = nacionalidad.cod_n";

if (count($_POST)) {
        foreach ($_POST as $c => $v)
                $_POST[$c] = mysql_real_escape_string($v);
}

if ($_POST['cbo_autor'])
    $sql .= " and autor.cod_a = ".$_POST['cbo_autor'];

if ($_POST['cbo_libro'])
    $sql .= " and libro.cod_l = ".$_POST['cbo_libro'];

if ($_POST['cbo_nacionalidad'])
    $sql .= " and nacionalidad.cod_n = ".$_POST['cbo_nacionalidad'];

if ($_POST['txt_isbn'])
    $sql .= " and isbn_l = '".$_POST['txt_isbn']."'";

if ($_POST['txt_palabra_clave'] != '')    {
    $sql .= " and (";
    $sql .= " nom_l like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or cant_pag_l like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or isbn_l like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or precio_l like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or nom_a like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or ape_a like '%".$_POST['txt_palabra_clave']."%'";
```

```
    $sql .= " or fec_nac_a like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or fec_dec_a like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= " or desc_n like '%".$_POST['txt_palabra_clave']."%'";
    $sql .= ")";
}

$sql .= " group by libro.cod_l";
$sql .= " order by nom_l";
$res = mysql_query($sql);

?>
<html>
<head>
    <title>mostrar_resultados.php</title>
        <style>
                * {
                font-family: Arial, Helvetica, sans-serif;
                font-size: 13px;
                font-weight: bold;
                color: #57534c;
                text-decoration: none;
                line-height: 24px;
                }

                table {
                border-top: 1px solid #57534c;
                border-right: 1px solid #57534c;
                }

                td {
                border-bottom: 1px solid #57534c;
                border-left: 1px solid #57534c;
                }
        </style>
</head>
<body>
```

```
<table cellpadding="4" cellspacing="0">
 <tr>
  <td colspan="2">

                <?php

                $temp = mysql_num_rows($res)==1 ? '1 libro encontrado' :
mysql_num_rows($res).' libros encontrados';
                echo 'Resultado de la Busqueda:'.$temp.':';

                ?>

   </td>
 </tr>
 <tr>
  <td>

      <table cellpadding="4" cellspacing="0">

      <?php

      if (!mysql_num_rows($res)) {
                echo '<td bgcolor="#C0C0C0"> 
No se encontraron resultados.</td>';
      } else {

                $c=0;
                while ($f = mysql_fetch_array($res)) {

                        $col = $c % 2 ? "#F0F3F8" : "#C0C0C0";
                        echo '<tr bgcolor='.$col.'>';
                        echo '<td>'.ucwords(strtolower($f['nom_l'])).'</td>';
                        echo '<td>'.ucwords
```

```
(strtolower($f['ape_a'].', '.$f['nom_a'])).'</td>';
                                echo '<td>'.$f['cant_pag_l'].'</td>';
                                echo '<td>'.$f['isbn_l'].'</td>';
                                echo '<td>u$s '.$f['precio_l'].'</td>';
                                echo '<tr>';

                                $c++;
                        }
            }

            ?>

            </table>

        </td>
      </tr>
      <tr>
        <td colspan="2"><a href="buscar.php">Volver al buscador</a></td>
      </tr>
    </table>

    </body>

    </html>
```

Ahora guardaremos los archivos en una carpeta accesible en el servidor. Debería quedar algo similar a lo que muestra la **figura 4**.

Recordemos que, si estamos trabajando de forma local, para acceder a una de estas páginas debemos ingresar en nuestro navegador: **http://127.0.0.1/nombre_pagina.php**, siempre y cuando hayamos guardado las páginas en el directorio raíz del servidor (normalmente **HTDOCS)**. De lo contrario, debemos incluir en la dirección la ruta hasta **nombre_pagina.php**, por ejemplo: **http://127.0.0.1/ejemplo1/paginas/nombre_pagina.php**. Recordemos que, antes de acceder a la página **buscar.php**, debemos ejecutar una vez la página **crear_estructura.php**.

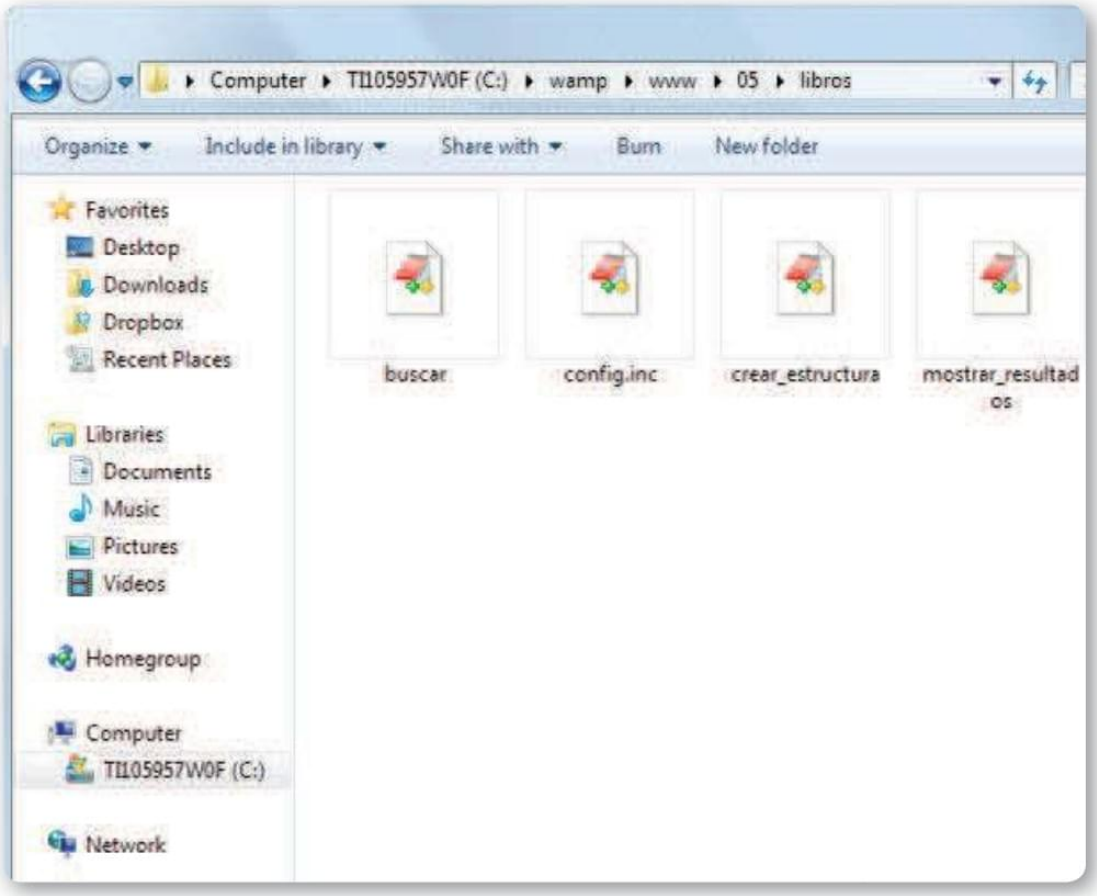


Figura 4. Listado de archivos del proyecto.

RESUMEN

Los datos que se guardan en una base están, por lo general, en continuo cambio. En este capítulo, aprendimos a recuperar filas a través de consultas de selección, insertar datos, modificarlos y borrarlos. Además consideramos las distintas funciones para recorrer el conjunto de filas devueltas por una consulta y la manera de acceder a cada campo en particular; por último, vimos un ejemplo completo de todo lo explicado.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué diferencia hay entre las funciones **mysql_fetch_array** y **mysql_fetch_row**?
2. La función **mysql_num_rows** es equivalente a una función del lenguaje SQL, ¿cuál es esa función?
3. ¿Qué diferencia hay entre las funciones **mysql_num_rows** y **mysql_affected_rows**?
4. ¿Qué aplicación tienen las constantes **MYSQL_ASSOC**, **MYSQL_NUM** y **MYSQL_BOTH** en la función **mysql_fetch_array**?
5. ¿Qué función deberíamos usar si quisiéramos saber cuál es el nombre de columna de una tabla?

EJERCICIOS PRÁCTICOS

1. Abra la tabla y el código del archivo **Cap5_practica1** y responda, ¿cuál es la salida?
2. Suponiendo la tabla del ejercicio 1, ¿qué devuelve el siguiente código del archivo **Cap5_practica2**?
3. Cree una tabla y verifique a través del ingreso de registros que efectivamente esta contiene un campo autoincremental.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: **profesor@redusers.com**.

*06

Manejo de errores

Para lograr desarrollos robustos y estables, nos ayudará conocer los errores frecuentes que podemos cometer o que suelen aparecer, y aprender de ellos. Tanto PHP como MySQL nos brindan funciones y opciones de configuración que nos permitirán tener un acceso controlado y un entendimiento de los errores que puedan surgir en nuestros futuros desarrollos.

Reportes de error en PHP.....170
- error_reporting.....170
- display_errors.....173
- display_startup_errors.....174
- log_errors.....175
- error_log.....176
- log_errors_max_len.....177
- ignore_repeated_errors.....177
- ignore_repeated_source.....180
- track_errors.....180
- html_errors.....181
- error_prepend_string.....183
- error_append_string.....184
- warn_plus_overloading.....184
- mysql_error.....184
- mysql_errno.....186
- register globals.....186
- perror.....191
- Manejo de excepciones.....191

Resumen.....191

Actividades.....192

Reportes de error en PHP

PHP nos ofrece distintas funciones y directivas de configuración para obtener diferentes niveles de reportes de error. A continuación, presentaremos un listado de ellas y algunos ejemplos.

error_reporting

Esta es una función muy útil para detallar los tipos de errores que es posible mostrar en las páginas. Las opciones que nos brinda son:

- **E_ALL**: todos los errores y advertencias, excepto **E_STRICT**.
- **E_ERROR**: errores fatales en tiempo de ejecución.
- **E_WARNING**: advertencias en tiempo de ejecución y errores no fatales.
- **E_PARSE**: errores de compilación internos.
- **E_NOTICE**: errores no críticos, normalmente producto de fallas en el código, como variables no inicializadas, por ejemplo.
- **E_CORE_ERROR**: errores fatales producidos durante el inicio de PHP.
- **E_CORE_WARNING**: advertencias –no errores fatales– de posibles errores producidos durante el inicio de PHP.
- **E_COMPILE_ERROR**: errores fatales en tiempo de compilación.
- **E_COMPILE_WARNING**: advertencias –no errores fatales– de posibles errores en tiempo de compilación.
- **E_USER_ERROR**: mensaje de error generado por el usuario que se asemeja a **E_ERROR**.
- **E_USER_WARNING**: mensaje de error generado por el usuario que se asemeja a **E_WARNING**.
- **E_USER_NOTICE**: mensaje de error generado por el usuario que se asemeja a **E_NOTICE**.
- **E_STRICT**: noticias en tiempo de ejecución para hacer que PHP sugiera cambios para mantener la compatibilidad de su código.
- **E_RECOVERABLE_ERROR**: el error puede ser capturado y tratado a partir de excepciones.

- **E_DEPRECATED**: mensajes acerca de código que no será soportado por PHP en futuras versiones.
- **E_USER_DEPRECATED**: similar al anterior solo que los mensajes de error pueden ser generados por el usuario.

Si estamos desarrollando sitios, es conveniente tener habilitadas todas las opciones referidas al informe de errores. Esto no es aconsejado cuando estamos en etapa de producción, es decir, si el sitio está en funcionamiento y es visitado por los usuarios.

A partir de versiones posteriores a PHP 5, se incluirá el nivel de errores denominado **E_RECOVERABLE_ERROR**, que podremos utilizar en lugar de **E_ERROR** para identificar fallas que requieren un control y un manejo específicos, pero de los que el sistema puede recuperarse, es decir, que producen una inestabilidad no definitiva. Además, el nivel de errores **E_ALL** contendrá al nivel **E_STRICT**.

Se puede configurar el reporte de errores a través de constantes simbólicas, que deben ser deshabilitadas explícitamente mediante el uso de algunos operadores. Los operadores son:

- **~**: negación
- **|**: or (O)
- **&**: and (Y)

ERRORES CONOCIDOS

En las distribuciones no solo de PHP, MySQL y Apache sino de muchas otras aplicaciones, suelen incluirse archivos que contienen información acerca de los errores encontrados o reportados por usuarios y todavía no solucionados. Incluso, si somos nosotros los que encontramos algún error desconocido hasta el momento, podemos reportarlo a través de un formulario o a una dirección de correo electrónico que figura en la misma distribución.

Veamos algunos ejemplos:

Mostrar todos los errores menos los errores no críticos:

```
error_reporting = E_ALL & ~E_NOTICE
```

Mostrar todos los errores menos los errores no críticos y las advertencias:

```
error_reporting = E_ALL & ~(E_NOTICE | E_WARNING)
```

Mostrar todos los errores:

```
error_reporting  =  E_ALL
```

En PHP existe una función con el mismo nombre que la directiva vista anteriormente (**error_reporting**), que permite pasar por encima de lo que está escrito en el archivo **php.ini** y utilizar el nivel de reporte de errores que le demos a la función. Este valor permanecerá solo durante nuestra sesión, luego se perderá y se seguirá usando el valor que figura en el **php.ini**. La sintaxis de la función es la siguiente:

AYUDA EN PHP

Algunos lenguajes de programación traen junto con la ayuda del lenguaje sus entornos de desarrollo. Como PHP no posee un entorno de desarrollo único, es posible que deba descargar la ayuda que proviene del sitio web de PHP, ya que tampoco viene junto con la versión oficial. La ayuda está disponible en muchos formatos y para diversos sistemas operativos.

```
error_reporting(nivel);
```

El argumento **nivel** puede ser uno de los siguientes valores enteros:

- 1: **E_ERROR**
- 2: **E_WARNING**
- 4: **E_PARSE**
- 8: **E_NOTICE**
- 16: **E_CORE_ERROR**
- 32: **E_CORE_WARNING**

Para seleccionar varias opciones a la vez, debemos sumar los valores enteros de las opciones. Por ejemplo, si quisiéramos utilizar las opciones **E_ERROR**, **E_WARNING** y **E_PARSE**, deberíamos escribir:

```
error_reporting(7);
```

La función **error_reporting** devuelve el valor anterior, en forma de valor entero, que contenía la directiva. Se puede utilizar la función **ini_set** para configurar el nivel de error, pero tendrá validez solo en tiempo de ejecución:

```
ini_set ("error_reporting", opcion);
```

El argumento **opcion** puede ser **E_ALL**, **E_WARNING**, etc. Con la función **ini_set**, podemos modificar valores de otras directivas en tiempo de ejecución.

display_errors

Para esta función obtenemos dos valores: **1** o **0**. Si se toma el valor verdadero (**1**), al ocurrir un error se mostrará el mensaje correspondiente en pantalla.

Por ejemplo, si intentamos conectarnos a una base de datos inexistente, obtendremos un mensaje a través del navegador como muestra la **figura 1**.

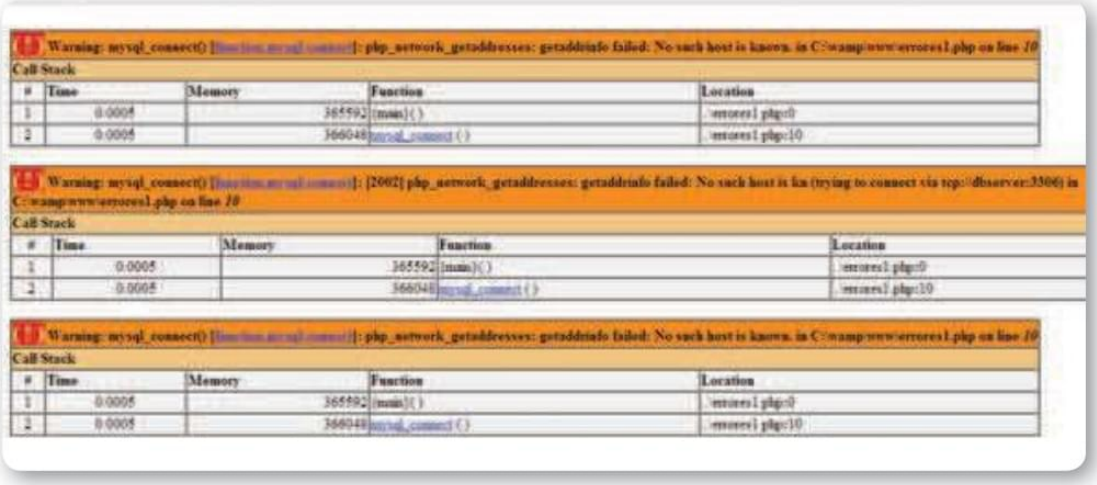


Figura 1. Directiva `display_errors` habilitada.

El valor por defecto de esta directiva es **1**. Sin dudas, tenerla activada es muy útil durante el período de desarrollo de un sitio web, ya que notificará los errores que cometamos. Si, en cambio, el sitio está publicado y en etapa de producción, es conveniente no mostrar los mensajes de error, por más que estos ocurran, a los usuarios. Esto tiene que ver tal vez con una cuestión relacionada con el diseño, con lo amigable que necesita ser siempre un sitio web.

Incluso, en ocasiones, se producen errores que no modifican el buen funcionamiento del sistema, pero que de todas formas emiten mensajes de error al producirse. De hecho, formalmente, podríamos decir que los errores deben solucionarse en la etapa de desarrollo del sitio, y solo cuando no se produzcan deberá pasar a la publicación definitiva.

display_startup_errors

Es posible que se produzcan errores durante el arranque de PHP, que no forman parte de los que se muestran o se ocultan mediante la configuración de la directiva **display_errors**. Para mostrar u ocultar estos errores, debemos configurar la directiva **display_startup_errors**, que puede tomar los

valores **1** (verdadero) o **0** (falso). El valor por defecto es **0** (oculta los errores). Nuevamente, recomendamos no tener habilitada esta directiva durante la etapa de producción de un sitio web, pero sí durante la etapa de desarrollo.

Un error típico dentro de la categoría de errores de arranque se da cuando PHP no encuentra directorios especificados en el archivo **php.ini**, como es el caso de las extensiones o los definidos en la directiva **include_path**.

log_errors

Cada servidor web, generalmente, mantiene en un archivo el registro de todos los errores ocurridos. Estos errores pueden ser de cualquier tipo, desde problemas al cargar módulos hasta de configuración. Si queremos agregar a este archivo los errores que se producen en las páginas PHP, deberemos configurar esta directiva para que lo haga, esta admite dos valores: **1** (verdadero) para activarla y **0** (falso) para desactivarla. Su valor por defecto es **0**.

Tengamos en cuenta que habilitar o deshabilitar esta directiva no modifica la conducta de **display_errors** ni viceversa. Puede mostrar u ocultar los errores producidos al mismo tiempo que los guarda en el archivo de errores del servidor, es decir, ambas directivas no son privativas ni están relacionadas de ninguna manera.

Si tenemos esta directiva habilitada, pronto –al cometer un error– podremos ver en el archivo de errores líneas como las siguientes:

SISTEMAS OPERATIVOS

Los mensajes de error pueden variar de un sistema operativo a otro, por lo que se recomienda tener cuidados especiales en este aspecto y tratar de conocer en profundidad las características técnicas de cada uno de los sistema disponibles para no ser sorprendidos por las posibles incompatibilidades.

```
[Thu Mar 11 22:19:55 2014] [error] [client 127.0.0.1] PHP Warning:
mysql_query(): Access denied for user: 'usuario@localhost' (Using password: NO) in
c:\archivos de programa\apache group\apache\htdocs\web1\index.php on line 33
```

Estas marcan errores que tienen que ver con PHP. En el ejemplo anterior, se puede ver que un usuario se intentó conectar a una base de datos y no tenía permisos suficientes para hacerlo.

También registra errores de sintaxis, los **parse errors**, que ocurren cuando escribimos de manera incorrecta líneas de código, cuando no cerramos correctamente una cadena o cuando nos sobra o falta alguna llave para cerrar o abrir módulos de código. Por ejemplo:

```
[Sun Apr 17 21:02:28 2005] [error] [client 127.0.0.1] PHP
Parse error:  parse error, unexpected ',' in c:\archivos de programa\
apache group\apache\htdocs\web1\index.php on line 49
```

Si utilizamos el servidor web Apache, podremos ubicar este archivo en la carpeta **LOGS**, dentro del directorio **Apache**. El archivo de errores normalmente se llama **error.log** y, al ser de texto plano, se puede abrir con cualquier editor de textos. Si el servidor web Apache está activo, no podremos tener acceso al archivo. Esto es lógico porque Apache necesita acceso permanente a este archivo, ya que lo modifica –en realidad solo agrega líneas– de manera permanente. Para leer este archivo, es posible realizar una copia y trabajar sobre ella o apagar el servidor web.

error_log

Desde esta directiva es posible configurar el nombre bajo el cual se guardará el archivo que almacena los errores ocurridos. Podemos especificar también la ruta hacia el archivo, de manera de poder ubicarlo en el directorio que nos resulte adecuado.

log_errors_max_len

Esta directiva está estrechamente ligada a las anteriores, ya que nos permite definir la capacidad máxima que podrá tener el archivo de errores. Esta capacidad será definida en bytes, y el valor por defecto de la directiva es **1024**. Si no queremos imponer un límite al tamaño del archivo, le asignamos **0**. Recordemos este hecho que por lo general se presta a confusión y que sucede también en otras directivas: puede que el valor **0** indique valores ilimitados, es decir que represente un valor infinito, no acotado. Si quisiéramos evitar utilizar el archivo de errores del servidor web para almacenar las fallas relacionados con PHP, deberíamos configurar la directiva **log_errors**, vista antes.

> LOS ERRORES, PARA SER REITERADOS, DEBEN OCURRIR EN LA MISMA LÍNEA Y ARCHIVO

ignore_repeated_errors

Si configuramos esta directiva en el valor **1**, PHP mostrará los errores que se repiten solo una vez. Se asume que los errores, para considerarse reiterados, deben ocurrir en la misma línea y en el mismo archivo. Veamos un ejemplo:

AYUDA EN MYSQL

MySQL trae, junto con su distribución, una ayuda en formato HTML, que es realmente muy extensa y completa. No solo proporciona una referencia de las funciones y las características de la base de datos, sino también una serie de ejemplos y recomendaciones de mucha utilidad. Se puede descargar la ayuda en otros formatos desde el sitio web de MySQL (**www.mysql.com**).

:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> ejemplo de ignore_repeated_errors </TITLE>
</HEAD>

<BODY>

<?php
$cnx = mysql_pconnect('localhost', 'root', '') or DIE('Error en la conexion');
$db = mysql_select_db('base1');

for ($x=0; $x<3; $x++) {
    $res = mysql_query();
}
?>
</BODY>
</HTML>
```

Asumiendo que la conexión se haya establecido correctamente, con **ignore_repeated_errors** asignado a **0**, la salida será como lo muestra la **figura 2**. Y con **ignore_repeated_errors** asignado a **1**, la salida será como podemos observarla en la **figura 3**.

RECURRIR AL MANUAL

Es una buena costumbre tratar de resolver por nuestra cuenta los errores que cometemos, pero en ocasiones nos será más beneficioso consultar la ayuda de PHP o de MySQL y no quedarnos estancados perdiendo tiempo para tratar de solucionar algún inconveniente que se presente en nuestras aplicaciones.

Warning: mysql_query() expects at least 1 parameter, 0 given in C:\wamp\www\errores1.php on line *15*

Call Stack

#	Time	Memory	Function	Location
1	0.0004	366776	{main}()	..\errores1.php:0
2	0.0011	367120	mysql_query ()	..\errores1.php:15

Warning: mysql_query() expects at least 1 parameter, 0 given in C:\wamp\www\errores1.php on line *15*

Call Stack

#	Time	Memory	Function	Location
1	0.0004	366776	{main}()	..\errores1.php:0
2	0.0014	367352	mysql_query ()	..\errores1.php:15

Warning: mysql_query() expects at least 1 parameter, 0 given in C:\wamp\www\errores1.php on line *15*

Call Stack

#	Time	Memory	Function	Location
1	0.0004	366776	{main}()	..\errores1.php:0
2	0.0016	367352	mysql_query ()	..\errores1.php:15

Figura 2. `ignore_repeated_errors` deshabilitada.

Warning: mysql_query() expects at least 1 parameter, 0 given in C:\wamp\www\errores1.php on line *15*

Call Stack

#	Time	Memory	Function	Location
1	0.0003	366920	{main}()	..\errores1.php:0
2	0.0013	367264	mysql_query ()	..\errores1.php:15

Figura 3. `ignore_repeated_errors` habilitada.

Dadas las condiciones que debe cumplir un error para considerarse repetido, está claro que, como en el ejemplo, las líneas de error deberán estar dentro de ciclos o bucles (**for**, **while**, **do while**, **foreach**, etc.). Puede resultar cómodo tener esta directiva habilitada, porque nos facilitaría y aclararía de manera considerable la visión de los errores cometidos en el momento de tenerlos en la pantalla. Al acudir al código fuente de la aplicación y resolverlo, estaríamos solucionando todas sus instancias. El comportamiento de **ignore_repeated_errors** cambia según el valor de otra directiva llamada **ignore_repeated_source**.

ignore_repeated_source

En apartados anteriores, cuando explicamos el funcionamiento de la directiva **ignore_repeated_errors**, dijimos que, para considerar un conjunto de errores como repetidos, estos debían ubicarse en la misma línea del mismo archivo. La directiva **ignore_repeated_source** modifica la definición de lo que es en sí un error repetido con respecto a como lo hace la directiva **ignore_repeated_errors**. Cuando está activada, no tiene en cuenta el origen de los errores, es decir, el archivo en el que se produjeron. Así, si cometemos un mismo error repetido que se reitera a su vez en más de un archivo, solo veremos un mensaje de error.

Si mantenemos **ignore_repeated_source** en **0** e **ignore_repeated_errors** en 1, PHP se comportará como se expuso en la sección **ignore_repeated_errors**. Si mantenemos **ignore_repeated_source** en **1** e **ignore_repeated_errors** en **1**, PHP se comportará como se expuso en la sección **ignore_repeated_source**.

track_errors

En PHP existe una variable cuyo nombre es **php_errormsg** (**$php_errormsg**), que nos permite acceder al mensaje correspondiente al último error cometido, siempre y cuando la directiva **track_errors** esté habilitada. Suponiendo esto último, veamos el siguiente código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> ejemplo de track_errors </TITLE>
</HEAD>

<BODY>

<?php
echo $matriz_no_definida[34];
echo "<br>---------------------------";
```

```
echo '<br><br>Mensaje del ultimo error cometido: '.$php_errormsg;

echo "<br><br>---------------------------";
?>

</BODY>
</HTML>
```

Su salida será lo que muestra la **figura 4**. Esta directiva está deshabilitada por defecto; para habilitarla debemos asignarle el valor **On**.

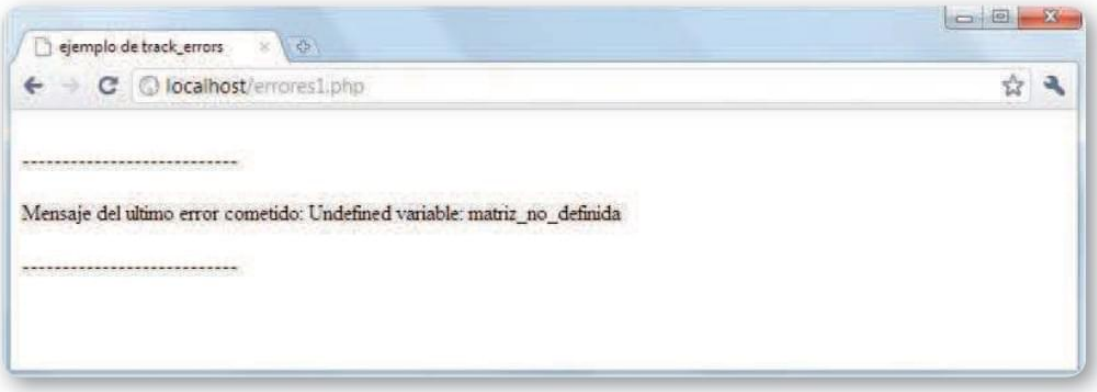


Figura 4. Directiva **track_errors** habilitada.

html_errors

Cuando esta directiva se encuentra deshabilitada, los errores se muestran como mensajes en formato HTML. Supongamos el siguiente código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> ejemplo de html_errors </TITLE>
</HEAD>

<BODY>
```

```
<?php
mysql_query($var);
?>
</BODY>
</HTML>
```

Al habilitar esta directiva, asignándole el valor **On**, a los mensajes de error se les agregará un enlace que derivará en una descripción del error cometido, que en general estará dentro del manual de PHP. Si no tenemos el manual de PHP, podemos bajarlo desde **www.php.net/docs.php**. Una vez descargado, lo copiamos a algún directorio; en nuestro ejemplo, lo llamaremos **manualPHP**, dentro del directorio de instalación de PHP. Luego, abrimos el archivo **php.ini**, vamos a la línea que comienza con **html_errors** y cambiamos su valor a **On**:

```
html_errors = On
```

Vamos a la línea que comienza con **docref_root** y escribimos la dirección en donde se encuentra el manual:

```
docref_root = "/manualPHP/"
```

Podemos usar también referencias a sitios de Internet. Por ejemplo:

```
docref_root = "http://www.direccion.com/manual"
```

Por último, indicamos la extensión que tienen los archivos del manual que descargamos (**HTML**, **TXT**, etc.)

```
docref_ext = .html
```

error_prepend_string

Esta directiva hace posible mostrar contenidos antes de los mensajes de error. Si escribimos textos en formato **HTML**, la directiva los soportará y mostrará la salida correspondiente. Además, nos permite personalizar el formato o la presentación de los mensajes de error y adecuarlos a nuestras necesidades. No hace falta habilitar o deshabilitar expresamente la directiva: bastará con asignarle valores o no. En el caso de asignarle algún texto, se considerará habilitada, de lo contrario, se considerará deshabilitada. Un ejemplo es:

```
error_prepend_string = "<b>Error!</b><br><font color=FFCC00>"
```

En este caso, cada vez que se produzca un error, antes de mostrarlo por pantalla, se imprimirá el texto **Error!** en negrita y, como se puede observar, el mensaje de error en sí se imprimirá con sus caracteres en color naranja, puesto que abrimos la etiqueta **<font color=FFCC00>** en la directiva **error_prepend_string**.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> ejemplo de error_prepend_string </TITLE>
</HEAD>

<BODY>

<?php

echo $matriz[34;

?>

</BODY>
</HTML>
```

error_append_string

Esta directiva es similar a **error_prepend_string**, pero con la diferencia de que su contenido se imprimirá luego del mensaje de error.

```
error_append_string = "</font><br>Final del mensaje de error."

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> ejemplo de error_append_string </TITLE>
</HEAD>

<BODY>

<?php

echo $matriz[34;

?>

</BODY>
</HTML>
```

warn_plus_overloading

Esta directiva nos informa, si es que está habilitada, a través de una advertencia, cuando usamos el signo de suma (**+**) para concatenar cadenas en lugar del utilizado por PHP con el mismo fin, que es el punto (**.**). Hay lenguajes cuya sintaxis es muy parecida a la de PHP que utilizan el signo más (**+**) en lugar del punto (**.**) para unir cadenas de caracteres o variables dentro de un script.

mysql_error

Esta función devuelve una descripción del último error cometido, relacionado con MySQL. Su sintaxis es la siguiente:

```
mysql_error(identificador);
```

El argumento **identificador** es un identificador de conexión opcional. Si no se incluye, la función intentará hallar la última conexión abierta activa al servidor MySQL. Si no la encuentra, tratará de crear una como si se llamara a **mysql_connect()** sin argumentos. La función no presenta mensajes de alerta (**WARNINGS**) sobre posibles errores, solo trabaja con los errores ya consumados. A continuación, damos algunos ejemplos acerca de cómo utilizarla.

```
<?php
$id = mysql_connect("168.22.22.3", "u", "p");
echo mysql_error();

$sql = "insert into t1 values (1, 2)";

mysql_db_query("base", $sql, $id);
echo mysql_error();
?>
```

En otros ejemplos de este mismo libro (**capítulo 4** y **5**), hemos utilizado la construcción **DIE**. La diferencia entre usar **DIE** y **mysql_error** es que, con la primera, podemos personalizar los mensajes de error (además **DIE** finaliza la ejecución del script) y, en cambio, con la segunda obtenemos mensajes de error **oficiales** de MySQL. Además, **DIE** puede usarse en cualquier ámbito, no solo para interceptar errores relacionados con el servidor de bases de datos MySQL.

Quizás en pequeñas y medianas aplicaciones se tienda a no usar este tipo de funciones, ya que, normalmente, conociendo en qué línea del script se produjo el error, podremos solucionarlo sin mayores inconvenientes. Sin embargo, es una buena práctica utilizar esta clase de funciones, que en muchos casos pueden evitarnos perder tiempo.

mysql_errno

En MySQL y en cualquier lenguaje de programación o aplicación, los errores están catalogados y es posible identificarlos a través de un código. Podemos conocer este código mediante la función **mysql_errno**. Su sintaxis es similar a **mysql_error**:

```
mysql_errno(identificador);
```

Un ejemplo de esta función es:

```
<?php
$id = mysql_connect("168.22.22.3", "u", "p");
echo mysql_errno();

$sql = "insert into t1 values (1, 2)";

mysql_db_query("base", $ sql, $id);
echo mysql_errno();
?>
```

register globals

Utilizando **register_globals = On**, PHP asume todas las variables como globales y no podrá diferenciar entre los diferentes tipos de ellas (si usa una variable en su script que tiene el mismo nombre de una variable de sesión, PHP no podrá distinguirlas y las tratará como a una sola).

Para solucionar este problema y trabajar con **register_globals = Off**, PHP nos ofrece los siguientes arrays:

- **$_SERVER**
- **$_GET**
- **$_POST**

- **$_COOKIE**
- **$_FILES**
- **$_ENV**
- **$_REQUEST**
- **$_SESSION**

La idea es identificar el método por el cual recibimos las variables y acceder a ellas mediante el array correspondiente. Por ejemplo, si enviamos variables a través de un formulario a una página, debemos recuperarlas a través del array correspondiente al método usado en el formulario.

```
<!--pagina1.php          -->

<form action=pagina2.php method=POST>
<input type=text name=nombre>
<input type=submit>
</form>


<?php

//pagina2.php

echo "la variable nombre contiene el valor : ".$_POST[nombre];

?>
```

REVISEMOS EL ARCHIVO DE ERRORES

Los servidores web guardan un archivo con los errores producidos durante un período. Es recomendable revisar este archivo para observar comportamientos irregulares.

A continuación, haremos una breve reseña de los distintos arrays que provee PHP para almacenar variables:

- **$_SERVER**: contiene una serie de variables generadas por el servidor web. Algunas de ellas son:
  - **REMOTE_ADDR**: dirección IP del usuario.
  - **SERVER_NAME**: nombre del servidor web.
  - **PHP_SELF**: nombre del archivo que se está ejecutando.

  Accedemos a ellas de la siguiente manera:

```
echo $_SERVER[PHP_SELF];
```

- **$_GET**: almacena las variables pasadas por URL (por ejemplo, a través de links o formularios con **method=get**). Quizás sea aquí donde se vea de manera clara cuál es el problema de seguridad que supone usar **register_globals = On**. Supongamos que tenemos un script para autentificar usuarios. Luego de un determinado proceso, si el usuario está registrado, almacenamos en la variable **usuario** el valor **registrado**. Al pasar la variable por URL, de acuerdo con la siguiente estructura: **http://www.misitioweb.com?usuario=registrado**, utilizando **register_globals = On** cualquiera estaría registrado, evitando el proceso de registro (recordemos que PHP no necesita declarar las variables).

REGISTER_GLOBALS

Habilitar la directiva **register_globals** puede convertirse en una tentación, ya que la manera de programar aplicaciones se vuelve más fácil; pero, tanto por cuestiones de seguridad como porque casi nadie la habilita, se recomienda tenerla deshabilitada.

Con **register_globals = Off** podríamos especificar el modo en el que recibiríamos la variable. Si definimos que la variable sea de tipo **SESSION**, no nos importa qué valor tenga **$_GET[usuario]**, lo que nos importa es el valor de **$_SESSION[usuario]**.

- **$_POST**: tiene un uso similar a **$_GET**, almacena las variables recibidas por el método **POST**; por ejemplo, a través de formularios.
- **$_COOKIE**: una matriz asociativa de variables pasadas al script actual a través de cookies HTTP.
- **$_FILES**: aquí se guardan algunas informaciones acerca de los archivos enviados a través de formularios.
- **$_ENV**: las variables que se incluyen en este array realmente dependen del sistema operativo sobre el que se está trabajando.

Otras variables de entorno incluyen las variables CGI:

- **$_REQUEST**: una matriz asociativa que guarda los contenidos de **$_GET**, **$_POST** y **$_COOKIE**.
- **$_SESSION**: aquí se guardan las variables de tipo sesión.

Como dato interesante, si trabajamos con los arrays que nos provee PHP, nuestros scripts funcionarán tanto con **register_globals = On** como con **register_globals = Off**. Podemos observar qué valor tiene la directiva **register_globals** en su sistema de dos maneras: accediendo al archivo **php.ini** y buscando la línea que contenga algo parecido a lo que muestra la **figura 10**.

ERRORES EN TABLAS

Las innovaciones sobre las funcionalidades de las tablas incluyen avances respecto del manejo y la descripción de los errores cometidos por los usuarios. Sin embargo, el tratamiento de errores es una cuestión poco considerada por un alto porcentaje de usuarios.

O a través de la función **phpinfo**, como muestra la **figura 5**.

max_input_time	60	60
memory_limit	128M	128M
open_basedir	no value	no value
output_buffering	1	1
output_handler	no value	no value
post_max_size	8M	8M
precision	14	14
realpath_cache_size	16K	16K
realpath_cache_ttl	120	120
register_argc_argv	Off	Off
register_globals	Off	Off
register_long_arrays	Off	Off
report_memleaks	On	On
report_zend_debug	On	On
request_order	GP	GP
safe_mode	Off	Off
safe_mode_exec_dir	no value	no value

register_globals 1 of 1

Figura 5. Directiva **register_globals** en la salida de la función **phpinfo**.

Suelen circular algunas **soluciones** para los códigos programados con **register_globals = On**, que en resumen recorren todas las variables contenidas en los arrays vistos antes y copian sus valores a variables:

```
$nombre = $_GET[nombre];
```

O con la función **ini_set**:

```
ini_set("register_globals","1");
```

Pero debe quedar claro que se trata de un tema de seguridad y que es necesario trabajar con los arrays provistos por PHP. Las dos líneas de código anteriores solo mantienen el funcionamiento de las aplicaciones que están programadas con **register_globals = On** y que desean pasar a **register_globals = Off** sin modificar casi nada del código, pero el problema de seguridad alertado por PHP seguirá latente.

perror

MySQL exhibe mensajes y códigos de error en muchos sistemas. Para obtener más información o descripciones acerca de los errores podemos utilizar el programa perror que viene junto con la distribución de MySQL.

Las descripciones de los errores pueden variar de un sistema operativo a otro. Su sintaxis es la siguiente:

```
C:\> perror codigo [codigo] [codigo] [...]
```

Manejo de excepciones

Una **excepción** es un suceso o evento que ocurre y que interrumpe la ejecución normal de un programa. Puede ser por problemas en el hardware de un equipo, errores de programación u otros.

Las excepciones están directamente ligadas a la programación orientada a objetos. Cuando una excepción ocurre dentro de un método, este crea un objeto que guarda información acerca de la excepción ocurrida. PHP puede lanzar, intentar o capturar excepciones a través de los bloques **try**, **catch** y **throw**. Si la excepción no es capturada, se producirá un error fatal, y PHP mostrará un mensaje para referir esta situación. Las excepciones son comunes en lenguajes como Java o C++ y se implementaron en PHP versión 5.

RESUMEN

Analizamos una serie de funciones, programas y opciones de configuración de PHP y MySQL para conocer los errores y administrarlos. Vimos cómo programar las aplicaciones ante la disposición del nuevo valor por defecto que se le da a la directiva de **register globals**.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué tipo de variables se guardan en los arrays **$_POST**, **$_GET** y **$_SESSION**?
2. ¿Qué valor tiene la directiva **error_reporting** en su sistema?
   Para responder verifique el valor en el archivo **php.ini** o utilice las funciones **phpinfo** o **error_reporting**.
3. ¿Cómo configuraría la directiva **error_reporting** si su sitio está en etapa de desarrollo? ¿Y si está en etapa de producción? Justifique su respuesta.
4. ¿Qué diferencia hay entre **E_ALL** y **E_WARNING**?
5. Habilite la directiva **log_errors** en su sistema.
   ¿Qué valores contiene su archivo **error.log**?

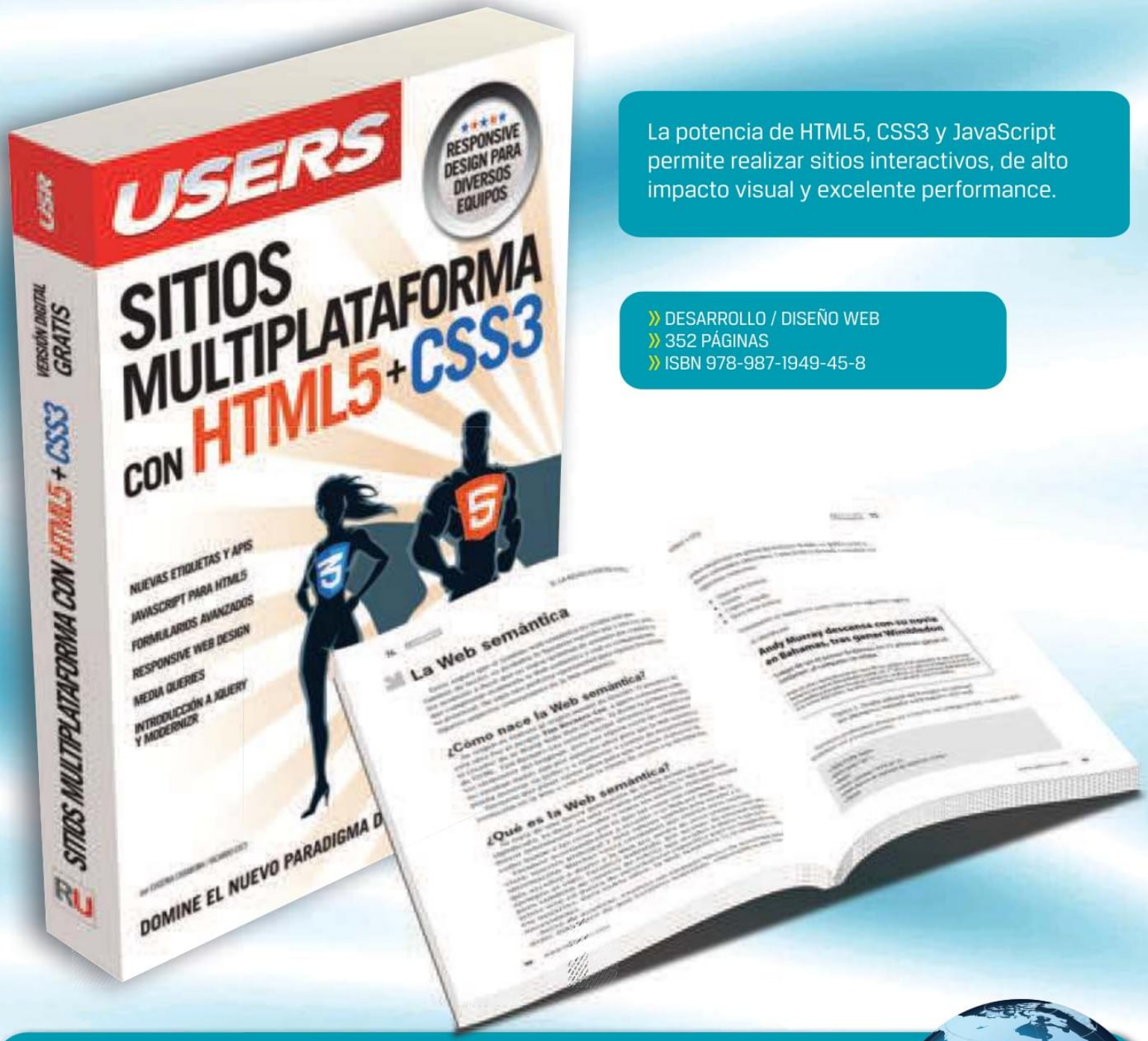
EJERCICIOS PRÁCTICOS

1. ¿Cuál es el error número 3 en MySQL?
2. Abra el archivo **Cap6_práctica2**, ¿cuál es la salida del código si usamos **register_globals = Off**? ¿Y con **register_globals = On**?

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: **profesor@redusers.com**.

CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN



La potencia de HTML5, CSS3 y JavaScript permite realizar sitios interactivos, de alto impacto visual y excelente performance.

» DESARROLLO / DISEÑO WEB
» 352 PÁGINAS
» ISBN 978-987-1949-45-8

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

MÁS INFORMACIÓN / CONTÁCTENOS

usershop.redusers.com +54 (011) 4110-8700 usershop@redusers.com

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

PHP+MySQL desde cero

En este libro nos centraremos en el uso del lenguaje de programación PHP en conjunto con una potente herramienta: el gestor de bases de datos MySQL. Desde el inicio veremos la teoría y puesta en práctica de bases de datos, continuaremos con el aprendizaje de las facilidades que nos provee el lenguaje y aprenderemos cómo potenciar nuestras aplicaciones con el gestor.

Dentro del libro encontrará*:

Teoría y práctica de la creación de bases de datos en PHP y MySQL / Utilización de sentencias SQL / Funciones de acceso a datos desde PHP / Administración y gestión de errores en MySQL

* Parte del contenido de este libro fue publicado previamente en *Desarrollo PHP+MySQL*, de esta misma editorial.

Otros títulos de la colección:

ISBN 978-987-1949-66-3

RU

REDUSERS.com
En nuestro sitio podrá encontrar noticias relacionadas y también participar de la comunidad de tecnología más importante de América Latina.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.