

USERS

¡CONTIENE
EJEMPLOS
PRÁCTICOS DE
CASOS
REALES!

DESARROLLO PHP + MySQL

**POTENCIE SUS SITIOS
CON EL PODER DE
AMBAS HERRAMIENTAS**

DISEÑO DE BASES DE DATOS RELACIONALES

**TIPOS DE DATOS, TABLAS
Y FUNCIONES EN MYSQL**

**CAPAS DE ABSTRACCIÓN
PARA ACCESO A DATOS**

TRANSACCIONES EN MYSQL

**MYSQLI: GUÍA DE REFERENCIA
DE FUNCIONES**

**SEGURIDAD, RESPALDO
Y RECUPERACIÓN DE DATOS**

por Francisco Minera



ANUALES USERS MANUALES USERS MANUAL USERS ANUALES USERS MAN

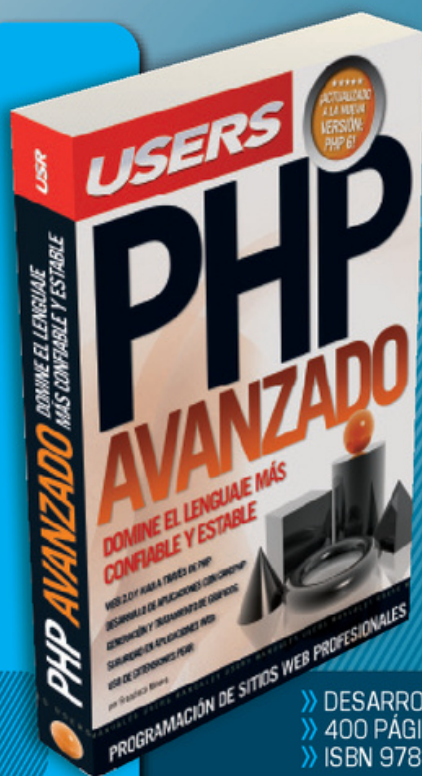
LA ROBUSTEZ DE PHP Y EL DINAMISMO DE MYSQL

CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN

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

 usershop.redusers.com
 usershop@redusers.com

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



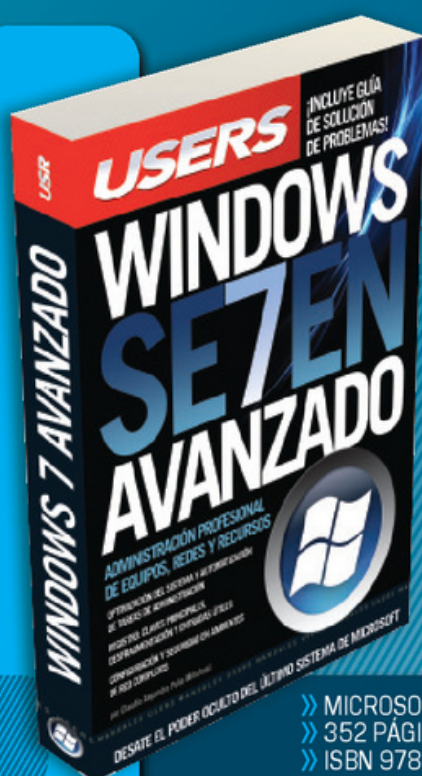
PROGRAMACIÓN
DE SITIOS WEB
PROFESIONALES

» DESARROLLO / PHP
» 400 PÁGINAS
» ISBN 978-987-1773-07-7



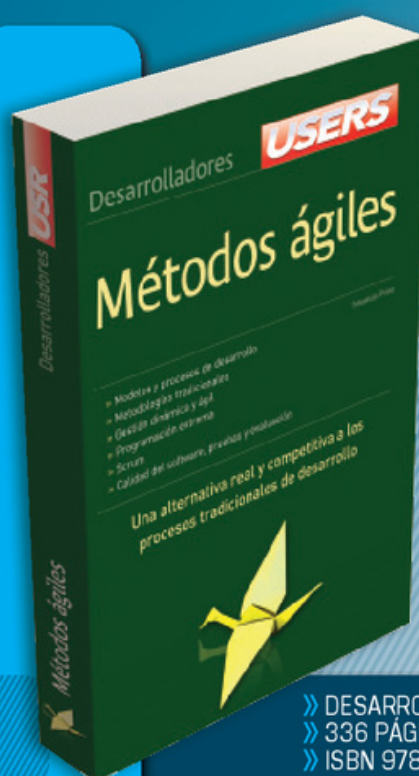
DISEÑO DE SITIOS
WEB COMPLEJOS
CON LAS HERRA-
MIENTAS DE ADOBE

» DISEÑO / DISEÑO WEB
» 320 PÁGINAS
» ISBN 978-987-663-022-1



DESCUBRA
EL PODER
OCULTO DEL
ÚLTIMO SISTEMA
DE MICROSOFT

» MICROSOFT / WINDOWS
» 352 PÁGINAS
» ISBN 978-987-1773-08-4



UNA ALTERNATIVA
A LOS MÉTODOS
TRADICIONALES
DE DESARROLLO

» DESARROLLO
» 336 PÁGINAS
» ISBN 978-987-1347-97-1

DESARROLLO PHP + MYSQL

**POTENCIE SUS SITIOS CON EL PODER
DE AMBAS HERRAMIENTAS**

por Francisco José Minera

RedUSERS



TÍTULO: Desarrollo PHP + MySQL
AUTOR: Francisco José Minera
COLECCIÓN: Manuales Users
FORMATO: 17 x 24 cm
PÁGINAS: 432

Copyright © MMXI. Es una publicación de Fox Andina en coedición con Gradi 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 III, MMXI.

ISBN 978-987-1773-16-9

Minera, Francisco
Desarrollo PHP + MySQL - 1a ed. - Buenos Aires : Fox Andina; Banfield -
Lomas de Zamora: Gradi, 2011.
432 p. ; 24x17 cm. - (Manual users; 207)

ISBN 978-987-1773-16-9

1. Informática. I. Título
CDD 005.3



LÉALO ANTES GRATIS

EN NUESTRO SITIO PUEDE OBTENER, DE FORMA GRATUITA, UN CAPÍTULO DE CADA UNO DE LOS LIBROS

RedUSERS
COMUNIDAD DE TECNOLOGIA

 **redusers.com**

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios, glosarios, atajos de teclado y todos los elementos necesarios para asegurar un aprendizaje exitoso y estar conectado con el mundo de la tecnología.



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**

Francisco José Minera



Es Analista de Sistemas y Programador Superior, recibido en el Instituto Superior de Formación Técnica N° 179 en el año 2003. Desde 2002 trabaja con PHP. Su experiencia como programador incluye numerosos trabajos realizados, en general, participando en diseños de sitios web dinámicos con bases de datos destinados a pequeñas y medianas empresas.

Sus tareas están relacionadas con el desarrollo de sistemas y plataformas para pequeñas y medianas empresas en diversos lenguajes y arquitecturas.

Es autor de los libros Proyectos con PHP, XML, PHP 5, Ajax, PHP Master, Desarrollo Web Profesional, Curso de programación PHP, PHP 6, PHP avanzado y Desarrollador Web.

PRÓLOGO

Ha transcurrido ya bastante tiempo desde el primer lanzamiento de este libro, lo que nos da una inmejorable oportunidad para realizar un análisis tanto acerca de lo que ha cambiado como de lo que sigue vigente hasta estos días.

En relación a las modificaciones, podríamos decir que han sido positivas para con los desarrolladores: las nuevas versiones tanto de PHP como de MySQL han sido aceptadas de muy buena manera y, aunque todavía hay un largo camino por recorrer, las mejoras incluidas han dado respuestas a los pedidos de la gran comunidad de desarrolladores. Otro punto a favor 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 rápidamente comenzar a desarrollar aplicaciones funcionales incluso durante el proceso de aprendizaje. En cuanto a los que provienen de otros lenguajes o tecnologías de desarrollo ven que tanto PHP como MySQL pueden brindarles la solución óptima para elaborar soluciones rápidas, estables, de bajo coste, y fácilmente actualizables.

Continuando con los ítems positivos podemos nombrar uno que tiene que ver justamente con el tiempo transcurrido, algo que habla por sí mismo tanto de la evolución como de la madurez alcanzadas: sin dudas esto no es una moda pasajera y el futuro es muy prometedor según los comentarios hechos por los responsables del lenguaje en diferentes entrevistas y opiniones realizadas en distintos blogs / redes sociales.

En relación a los puntos en contra, podemos nombrar a una de las virtudes del lenguaje (su facilidad de aprendizaje) que deriva en algo que podría eventualmente verse como algo 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 que los empleadores ponen tienen mayor preponderancia. Viendo el lado positivo, la competencia hace que el esfuerzo por parte de los programadores los haga mejores profesionales, y eso a su vez deriva en mejores condiciones laborales.

Otro punto que se puede marcar tiene que ver con la visión de algunas empresas para con el lenguaje: algunas, ya sea por publicidades, falta de conocimiento, o el tema del soporte técnico, prefieren apostar a alternativas como .Net o Java. Si bien se trata de opciones probadamente profesionales, quizás podríamos decir que algunas empresas todavía no cuentan con toda la información al momento de decidirse entre una opción u otra. Este hecho deriva en algo que tal vez sea una realidad aun hoy: PHP, para empresas de cierto nivel, no es visto como un lenguaje viable para grandes proyectos.

EL LIBRO DE UN VISTAZO

En la primera parte de este libro encontrará una introducción a la programación en el lenguaje PHP y el manejo de bases de datos relacionales, para luego profundizar en la sintaxis propia de PHP y MySQL. En cada caso, hallará ejemplos prácticos que le permitirán comprender con mayor facilidad los conceptos desarrollados. De forma particular se trata, además, una serie de temas de suma importancia, como son la administración de una base de datos y la implementación de medidas de seguridad.

Capítulo 1

PHP

En este capítulo encontraremos una serie de notas e información relevante relacionada con el lenguaje de programación PHP: como debemos adquirirlo, las versiones y sus principales características.

Capítulo 2

BASES DE DATOS RELACIONALES

A través de este capítulo haremos un interesante repaso acerca de qué son y para qué podemos utilizar las bases de datos relacionales: entre otras cosas analizaremos los modelos y diseño, normalización y operaciones relacionales básicas.

Capítulo 3

MYSQL

Para poner en práctica los conceptos teóricos 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.

Capítulo 4

PHP Y MYSQL

En este capítulo hallaremos información relevante sobre las distintas alternativas que nos ofrece el lenguaje de programación PHP

para acceder a e, incluso, definir estructuras de bases de datos utilizando el motor denominado MySQL: aprenderemos la sintaxis adecuada de las funciones, las diferencias y citaremos diversos ejemplos, además efectuaremos comentarios acerca del funcionamiento de cada una de ellas.

Capítulo 5

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.

Capítulo 6

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.

Capítulo 7

TRANSACCIONES

En este capítulo, nos ocuparemos de realizar un interesante recorrido a través de todas las ventajas que brinda

utilizar las transacciones, además podremos analizar su funcionamiento en los distintos tipos de tablas que utilicemos.

Capítulo 8

SEGURIDAD EN MYSQL

La información que almacenaremos en la base de datos deberá estar resguardada de cualquier problema o peligro. Veremos cuáles son las principales tareas de mantenimiento implicadas: backup, optimización, recuperación y reparación.

Capítulo 9

ADMINISTRACIÓN DE MYSQL

Una vez que la base de datos ha sido desarrollada, deberemos cumplir un nuevo rol que conlleva nuevos deberes y responsabilidades: el del administrador. Aquí entenderemos su naturaleza, conoceremos sus tareas principales y descubriremos cómo aplicar un sistema de privilegios.

Apéndice A

CAPAS DE ABSTRACCIÓN DE DATOS

Existen clases que nos permiten acceder a distintas bases de datos utilizando el mismo código, lo que es igual a contar con una

portabilidad de las aplicaciones que puede evitarnos un dolor de cabeza en el futuro. En este apartado veremos Pear::DB y ADOdb.

Apéndice B

EJEMPLO PRÁCTICO

La información presentada en éste apéndice intenta mostrar la aplicación práctica de los temas más importantes que han sido analizados y tratados en el libro, a través de la programación completa de un sistema de venta de discos de música.

Apéndice C

MYSQLI

Esta extensión ofrece a los programadores algunas nuevas funciones cuya sintaxis e implementación son diferentes de anteriores versiones. En esta sección del libro nos dedicaremos a presentar una completa guía de funciones que contiene la sintaxis adecuada para cada una de ellas.

Servicios al lector

Instalar y configurar PHP, Apache y MySQL para lograr un correcto funcionamiento es, desde ya, una tarea fundamental. Aquí veremos los pasos necesarios para lograrlo.



INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual encontrará una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados.

Cada recuadro está identificado con uno de los siguientes iconos:



CURIOSIDADES
E IDEAS



ATENCIÓN



DATOS ÚTILES Y
NOVEDADES



SITIOS WEB

RedUSERS

MEJORA TU PC



**Desarrollos temáticos
en profundidad**

Libros.

Coleccionables.

**Cursos intensivos
con multimedia**



**Capacitación
dinámica**

Revistas.

Sitios Web.

**Noticias al día,
downloads, comunidad**



**Información actualizada
al instante**

Newsletters.

*La red de productos sobre tecnología más
importante del mundo de habla hispana.*



redusers.com

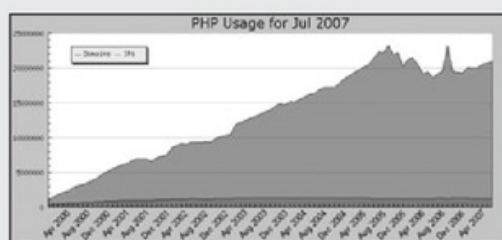
CONTENIDO

Sobre el autor	4
Prólogo	5
El libro de un vistazo	6
Información complementaria	7
Introducción	14

Capítulo 1

PHP

Obtención	16
Licencia de uso	17
Versiones	18
PHP 3	19
PHP 4	19
PHP 5	20
¿Qué versión utilizar?	22



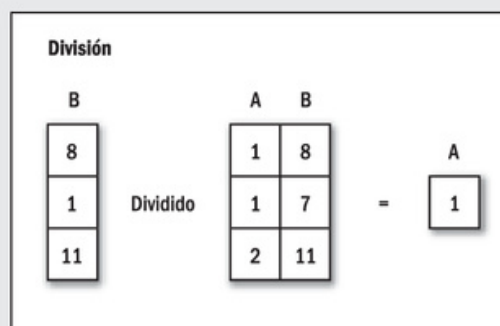
Extensiones en PHP	23
Bases de datos	25
Ventajas de trabajar con extensiones	26
Bibliotecas incorporadas	26
Facilidad de aprendizaje	28
Portabilidad	28
Resumen	29
Actividades	30

Capítulo 2

BASES DE DATOS RELACIONALES

¿Qué es y para qué sirve una base de datos?	32
Arquitectura de Bases de Datos	32
Modelo de datos	32
Modelo jerárquico	33

Modelo de red	34
Modelo relacional	35
Modelo orientado a objetos	36
Operaciones relacionales básicas	37
Diseño de la base de datos	42
Tabla	42



Relación	43
Atributo	43
Claves primarias	44
Claves candidatas	45
Vistas	46
Esquemas	46
Índices	47

Tipos de relaciones

entre tablas	52
Cardinalidad Uno a Uno	52
Cardinalidad Uno a Varios	53
Cardinalidad Varios a Varios	53

Normalización	56
Datos redundantes	57
Primera Forma Normal	59
Segunda Forma Normal	59
Tercera Forma Normal	61

MySQL y el soporte para

el modelo relacional actual	61
Integridad referencial	62
Subconsultas	67
Resumen	67
Actividades	68

Capítulo 3**MYSQL**

¿Por qué MySQL?	70
Obtención de MySQL	71
Licencia de uso	71
Diferencias entre versiones	73
Tipos de datos	73
Cadenas de caracteres	73
Numéricos	76
Fecha y hora	78



Tipos de tablas	80
ISAM	81
MyISAM	82
MERGE	83
HEAP	85
InnoDB	85
BerkeleyDB	86
¿En qué casos usar cada una?	86

Referencia de funciones **87**

Funciones para trabajar con cadenas de caracteres	87
Funciones para trabajar con campos numéricos	98
Funciones para trabajar con fecha y hora	104
Funciones de conversión	113
Funciones agregadas o estadísticas	114
Operadores de comparación	114
Operadores lógicos	115

Introducción a los procesos almacenados	117
--	------------

El uso del monitor de MySQL	123
Resumen	127
Actividades	128

Capítulo 4**PHP Y MYSQL**

Cómo conectar PHP con MySQL	130
mysql_connect	130
mysql_pconnect	132
Ejecución de sentencias	133
mysql_db_query	133
mysql_query	134

Creación de base de datos a través de PHP	135
--	------------

mysql_create_db	135
-----------------	-----

Selección de una base de datos	137
mysql_select_db	137

Creación de tablas a través de PHP	138
---	------------

Obtener listados de datos y tablas a través de PHP	140
---	------------

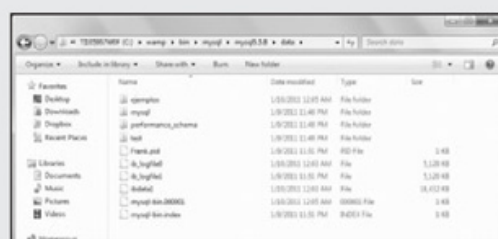
mysql_list_dbs	141
----------------	-----

mysql_list_tables	143
-------------------	-----

Utilización de múltiples bases de datos	144
--	------------

Cuándo hacerlo	144
----------------	-----

Cómo hacerlo	144
--------------	-----



Cerrar conexión con la base	146
mysql_close	146

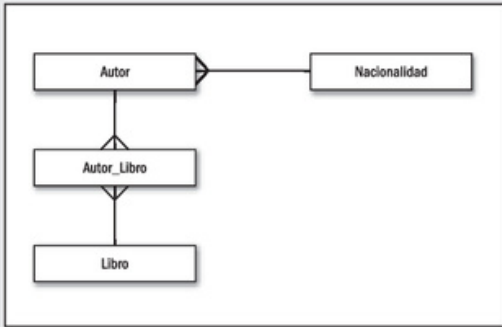
Ejemplo práctico: autos.php	147
------------------------------------	------------

Resumen	149
----------------	------------

Actividades	150
--------------------	------------

Capítulo 5**SQL EN PHP**

Consultas de selección	152
-------------------------------	------------

Selección simple	152
Consultas multitabla	152
Subconsultas	152
Recorrer las filas devueltas de una consulta	153
mysql_fetch_array	153
mysql_fetch_row	158
mysql_fetch_object	158
mysql_fetch_assoc	160
mysql_fetch_field	163
Moverse entre registros	164
mysql_data_seek	164
Número de registros y campos devueltos	166
mysql_num_rows	166
mysql_affected_rows	167
mysql_num_fields	169
Insert	170
Campos autoincrementables	170
mysql_insert_id	171
Delete	172
	
Update	173
Ejemplo práctico: libros.php	174
Resumen	187
Actividades	188
Capítulo 6	
ERRORES	
Reportes de error en PHP	190
error_reporting	190
display_errors	193
display_startup_errors	194

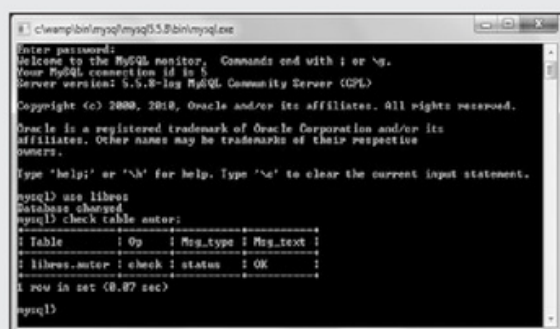
log_errors	194
error_log	196
log_errors_max_len	196
ignore_repeated_errors	197
ignore_repeated_source	199
track_errors	200
html_errors	201
error_prepend_string	203
error_append_string	204
warn_plus_overloading	205
mysql_error	206
mysql_errno	207
	
register_globals	208
Códigos de error en MySQL	212
perorr	212
Errores en INNodb	212
Manejo de excepciones	213
Resumen	213
Actividades	214
Capítulo 7	
TRANSACCIONES	
¿Qué es una transacción?	216
Casos de aplicación	216
Implementación de transacciones con MySQL	217
	

La variable autocommit	220
Tablas transaccionales	221
BDB	222
INNodb	222
Bloqueos	223
Bloqueos en INNODB: FOR UPDATE	224
Bloqueos en INNODB: LOCK IN SHARE MODE	227
Resumen	227
Actividades	228

Capítulo 8

SEGURIDAD EN MYSQL

Backup de Bases de Datos	230
BACKUP	230
MySQLDump	231



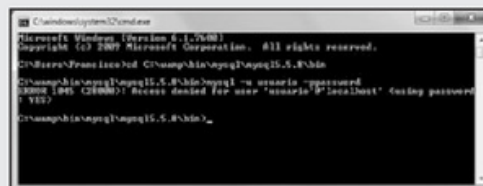
Recuperación ante fallos	235
Desde la línea de comandos	235
Desde el prompt de MySQL	236
Seguridad a nivel tabla	237
Reparación de tablas de una Base de Datos	237
Optimización de tablas	241
Introducción a la replicación	242
Seguridad física	243
Resumen	243
Actividades	244

Capítulo 9

ADMINISTRACIÓN DE MYSQL

El rol del administrador de Bases de Datos	246
---	-----

El rol del usuario: usuarios y administradores del sistema	247
Sistema de privilegios	247
Tabla user	248
Tabla host	249
Tabla db	250
Tablas tables_priv y columns_priv	251
Jerarquías de acceso	252



Comandos grant y revoke	253
Gestión de privilegios con INSERT	256
Gestión de privilegios con SELECT	257
Gestión de privilegios con UPDATE	257
Gestión de privilegios con DELETE	257
Baja de usuarios	258
mysql_change_user	258
Listar todos los usuarios mediante una consulta	258
Resumen	259
Actividades	260

Apéndice A

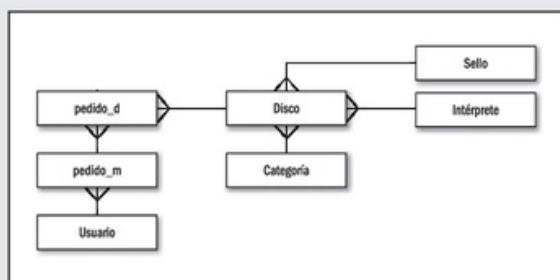
CAPAS DE ABSTRCCIÓN PARA ACCESO A DATOS

Arquitectura y funcionamiento	262
Introducción a MDB2	262
Introducción a ADOdb	274



Apéndice B**EJEMPLO PRÁCTICO**

Aplicación completa: discos 288



cnx.php	294
index.php	295
buscador.php	296
login.php	298
verdiscos.php	300
carrito.php	305
Menú de opciones	311
Dar de alta discos	312
Dar de baja discos	315



Modificar los datos de los discos	317
Dar de alta categorías	321
Dar de baja categorías	323
Modificar los datos de las categorías	325
Dar de alta intérpretes	327
Dar de baja intérpretes	329
Modificar los datos de los intérpretes	331
Dar de alta sellos	333
Dar de baja sellos	335
Modificar los datos de los sellos	337
Ventas por mes	339
Usuarios más activos	341

Apéndice C**MYSQLI**

Referencia de funciones 344

Servicios al lector**SERVICIOS AL LECTOR**

Apache	410
PHP	412
MySQL	412
Configuración	413
Instalación sobre Linux	415
Apache	415
PHP	416
MySQL	417
Configuración	419

PHP Version 4.3.4	
System	Windows 9x JAMACA 4.90
Build Date	Nov 2 2003 23:43:42
Server API	CGI/FastCGI
Virtual Directory Support	enabled
Configuration File (php.ini path)	C:\WINDOWS\php.ini
PHP API	20020918
PHP Extension	20020429
Zend Extension	20021010
Debug Debug	no
Thread Safety	enabled
Registered PHP Streams	php, http, ftp, compress.zlib

Información acerca del funcionamiento	420
Configuración de PHP a través del archivo php.ini	428
Configuración de MySQL a través del Archivo php.ini	425
Configuración del archivo my.cnf	428
Índice temático	429
Equivalencia de términos	431
Abreviaturas comúnmente utilizadas	433



INTRODUCCIÓN

Podríamos decir que uno de los objetivos de este libro es el de introducir al lector novel al desarrollo de aplicaciones web tomando como base a dos de las herramientas más populares que existen en la actualidad: el lenguaje de programación PHP y la base de datos relacional MySQL.

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

Pero más allá de lo circunstancial, 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 libro está dirigido a dar respuestas que a su vez lleven a nuevas preguntas, es decir disparadores que obliguen al lector a continuar permanentemente la búsqueda de nuevos niveles.

Los requisitos para con el lector son mínimos: alcanzara solo con tener experiencia en la utilización de algún sistema operativo y una idea básica acerca del funcionamiento de la arquitectura cliente servidor, esto es realizar una petición a un servidor web a través de un navegador y obtener una respuesta adecuada.

En definitiva, tanto el lenguaje PHP como la base de datos MySQL han continuado exitosamente un camino iniciado hace ya mucho tiempo y los resultados han sido realmente beneficiosos para todos los que de alguna manera están relacionados al mundo del desarrollo de aplicaciones web, y la idea es procurar que la mayor cantidad de desarrolladores se acerquen a esta vía.

PHP

En este capítulo encontraremos una serie de notas referentes al lenguaje de programación PHP: cómo adquirirlo, sus versiones, sus principales características y demás temas que servirán de base para los capítulos subsiguientes.

Obtención	16
Licencia de uso	17
Versiones	18
PHP 3	19
PHP 4	19
PHP 5	20
¿Qué versión utilizar?	22
Extensiones en PHP	23
Bases de datos	25
Ventajas de trabajar con extensiones	26
Bibliotecas incorporadas	26
Facilidad de aprendizaje	28
Portabilidad	28
Resumen	29
Actividades	30

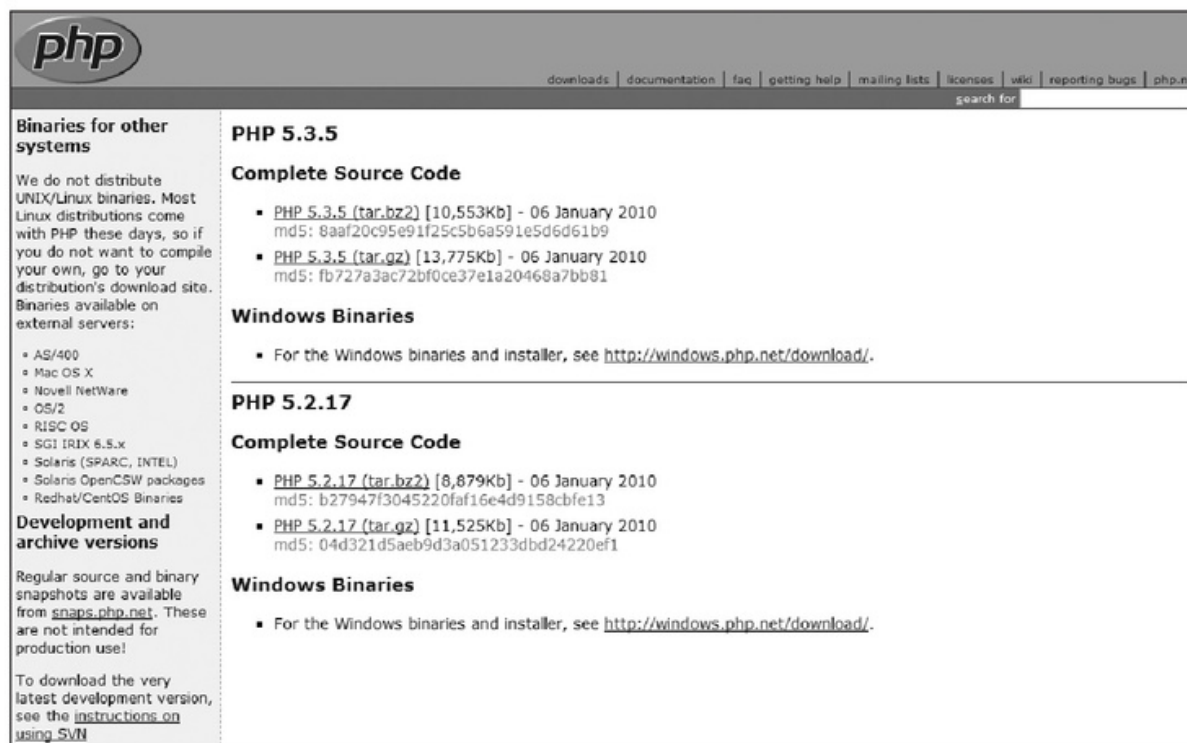
OBTENCIÓN

PHP, acrónimo de **PHP Hypertext Preprocessor**, es un lenguaje de programación que se utiliza en la mayoría de los casos 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 gran parte de las necesidades que posea 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 de PHP, deberemos ingresar a su sitio web, **www.php.net/downloads**, y seleccionar la opción que coincida con nuestro sistema operativo.



The screenshot shows the PHP website's download section. The page has a header with the PHP logo and navigation links: downloads, documentation, faq, getting help, mailing lists, licenses, wiki, reporting bugs, and php.net. A search bar is also present. The main content area is divided into two columns. The left column contains links for 'Binaries for other systems' and 'Development and archive versions'. The right column shows the download options for 'PHP 5.3.5' and 'PHP 5.2.17', including 'Complete Source Code' and 'Windows Binaries' with specific download links and MD5 hashes.

Binaries for other systems

We do not distribute UNIX/Linux binaries. Most Linux distributions come with PHP these days, so if you do not want to compile your own, go to your distribution's download site. Binaries available on external servers:

- AS/400
- Mac OS X
- Novell NetWare
- OS/2
- RISC OS
- SGI IRIX 6.5.x
- Solaris (SPARC, INTEL)
- Solaris OpenCSW packages
- Redhat/CentOS Binaries

Development and archive versions

Regular source and binary snapshots are available from snaps.php.net. These are not intended for production use!

To download the very latest development version, see the [instructions on using SVN](#)

PHP 5.3.5

Complete Source Code

- [PHP 5.3.5 \(tar.bz2\)](#) [10,553Kb] - 06 January 2010
md5: 8aaf20c95e91f25c5b6a591e5d6d61b9
- [PHP 5.3.5 \(tar.gz\)](#) [13,775Kb] - 06 January 2010
md5: fb727a3ac72bf0ce37e1a20468a7bb81

Windows Binaries

- For the Windows binaries and installer, see <http://windows.php.net/download/>.

PHP 5.2.17

Complete Source Code

- [PHP 5.2.17 \(tar.bz2\)](#) [8,679Kb] - 06 January 2010
md5: b27947f3045220faf16e4d9158cbfe13
- [PHP 5.2.17 \(tar.gz\)](#) [11,525Kb] - 06 January 2010
md5: 04d321d5aeb9d3a051233dbd24220ef1

Windows Binaries

- For the Windows binaries and installer, see <http://windows.php.net/download/>.

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

Según Netcraft (www.netcraft.com), compañía que se dedica entre otras cosas a brindar estadísticas acerca del uso de tecnologías en Internet desde 1995, la utiliza-

ción de PHP en servidores viene creciendo en forma sostenida y se hace cada vez más popular, como se ve en la siguiente figura.

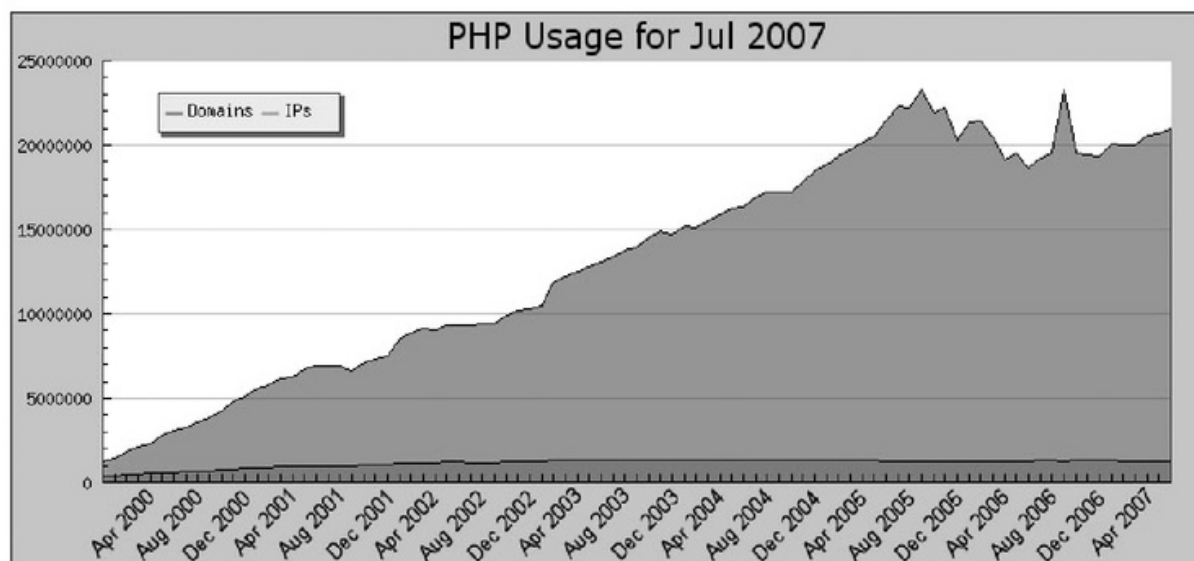


Figura 2. Uso de PHP en servidores a través del tiempo. (Fuente: <http://news.netcraft.com>).

LICENCIA DE USO

Se mencionó con anterioridad que PHP es un lenguaje “libre”. Este término se refiere al tipo de licencia que tiene, y consiste, básicamente, en tres puntos o “libertades”:

- La **primera** libertad es la de usar el programa (PHP).
- La **segunda** es la de poder modificar el programa accediendo a su código fuente.
- La **tercera** es la de distribuir el programa modificado o no.

La licencia de PHP está disponible en **www.php.net/license** o, también, viene junto con el programa en el archivo **license.txt**.

La redistribución, modificación y uso del lenguaje PHP están permitidos bajo las siguientes normas (licencia versión 3.01):

{ } MI NOMBRE ES MI NOMBRE

PHP es un acrónimo recursivo de **PHP Hypertext Preprocessor**, ya que utiliza su propio acrónimo como parte de su nombre. También ocurre esto con **GNU**, acrónimo recursivo de **GNU's Not Unix**. En el caso de PHP se eligió la primera letra P (que podría haber sido cualquier otra) porque en algún momento (en los inicios del lenguaje) éste se llamó **Personal Home Page Tools**.

- La redistribución del código fuente de PHP debe estar siempre acompañada de la correspondiente licencia y copyright de PHP.
- No puede usar el nombre **PHP** para promocionar sus productos, a menos que tenga permiso por escrito del **PHP Group**.

No hay una empresa comercial detrás de PHP; las mejoras y avances dentro del lenguaje resultan de una gran comunidad de desarrolladores que contribuyen, sin obtener réditos comerciales, con: código fuente, soporte a través de listas de correo, revisión del programa en busca de errores, notificación de fallas de seguridad, y más. Sobre esta base se sostiene una licencia que, justamente, asegura la libertad del lenguaje y no permite bajo concepto alguno que alguien obtenga beneficios comerciales de PHP y sea el dueño del lenguaje: éste es el espíritu de la licencia.

Cuando se desarrolla una aplicación y se la vende a terceros el importe que se cobra no es el lenguaje de programación sino la solución a un problema, el tiempo invertido en el desarrollo, el soporte, u otro particular.

VERSIONES

Esta sección no pretende repasar la historia de PHP, sino que, por el contrario, tiene como objetivo realizar una breve reseña de las características principales y los cambios que sufrió el lenguaje desde la versión **3** hasta la **5** inclusive.

PHP tuvo versiones anteriores a la 3, pero elegimos ésta como punto de partida porque es la versión más antigua que puede verse hoy en día en algunos sitios.

Esto sucede, mayormente, porque los sitios en cuestión –que fueron desarrollados en origen cuando esta versión era la última– cumplen con un objetivo preciso y no han necesitado actualización a versiones posteriores.

Hoy, la mayoría de las aplicaciones están programadas con las versiones **4** o **5** del lenguaje PHP. Al momento de escribir esta reedición, PHP versión **6** no ha sido lanzado a producción de manera oficial.

EL PRINCIPIO

PHP nació originalmente como un contador de visitas al sitio web que contenía el currículum de su creador, Rasmus Lerdorf, en 1995. Estaba programado en PERL. Lerdorf realizó algunas mejoras y agregó nuevas funcionalidades para luego liberar el código fuente al público. En 1997 Andi Gutmans, Zeev Suraski y Lerdorf describieron el código y anunciaron la versión 3 de PHP.

PHP 3

PHP versión 3 fue creado en 1997 y se lo considera una continuación de una versión anterior de PHP llamada **PHP/FI 2.0**, aunque llamarla “continuación” es relativo porque el código se rescribió completamente, y sólo se mantuvieron su forma de trabajar y algunas funciones implementadas: la idea era mantener a los usuarios de **PHP/FI** y seguir trabajando en conjunto con ellos.

Ya en esta versión, PHP tenía características que perduraron en el tiempo y aún hoy son un punto fuerte del lenguaje: soporte para una gran cantidad de bases de datos, interacción con protocolos de red y uso de extensiones.

En cuanto a la orientación a objetos (*POO*), la versión 3 le daba soporte a medias, sin implementar todas las posibilidades de este paradigma.

Se vislumbraba ya por estos tiempos a una gran cantidad de personas (decenas de millares de usuarios y cientos de miles de sitios web) que se encontraban interesadas no únicamente en utilizar sino, también, en colaborar con el lenguaje. PHP 3.0 se lanzó de manera oficial en el mes de junio de 1998.

PHP 4

Tomando como punto de referencia la versión anterior, el núcleo (parte del programa que se encarga de administrar los procesos) de PHP fue rescrito para la versión 4. Esto se dio porque al ser cada vez más popular, las aplicaciones existentes en el mercado fueron haciéndose más complejas y requerían mayor velocidad en la ejecución que la que podía ofrecer PHP versión 3.

El nuevo núcleo se denominó **Motor Zend** (en referencia a los nombres de sus principales desarrolladores, **Zeev Suraski** y **Andi Gutmans**).

Otras mejoras importantes son: el soporte para la mayoría de los servidores web, las funciones para el manejo de sesiones HTTP, los buffer de salida y la inclusión de gran cantidad de funciones de propósitos diversos.

La versión 4 llegó a estar instalada en más del 20% de los dominios en Internet. En cuanto a la POO, el soporte que PHP ofrecía, prácticamente, no se llegó a modificar con relación a la versión 3. Lo que sí se modificó fue su uso, ya que se volvió habitual para gran cantidad de usuarios, generalmente, en aplicaciones de gran tamaño. Este



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. Ésta es, quizás, otra de las ventajas de este lenguaje: aprender a programar en él da la posibilidad de hacerlo luego en otros.

requerimiento casi explícito por parte de los usuarios tendría su respuesta en la versión 5. Un caso similar se dio con XML: PHP daba soporte para manejar esta clase de archivos pero era bastante limitado en cuanto a las opciones que ofrecía. Un punto importante es que las extensiones escritas para PHP 3 no trabajan con PHP 4 (aunque es posible portar extensiones a PHP 4 si se tiene acceso a las fuentes originales). PHP 4 se introdujo por primera vez en el mercado en el año 1999, y la versión oficial fue lanzada en el mes de mayo de 2000.

PHP 5

Los cambios que experimenta PHP en esta versión son verdaderamente importantes, y se podría decir que revolucionaron el lenguaje.

La primera versión de PHP 5 se liberó en junio de 2003 (la primera versión considerada estable de PHP 5 se liberó en julio de 2004). Entre las nuevas características figuran:

The screenshot shows the official PHP website with the following content:

- What is PHP?** PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. If you are new to PHP and want to get some idea of how it works, try the introductory tutorial. After that, check out the online manual, and the example archive sites and some of the other resources available in the links section.
- Ever wondered how popular PHP is?** see the Netcraft Survey.
- Thanks To**
 - allaydigi
 - DeSci
 - par Networks
 - Server Central
 - Hosted Solutions
 - Spry VPS Hosting
 - SC Systems / H&I
 - OSU Open Source Lab
 - Yahoo! Inc.
 - BracySEC
 - NEXCESS.NET
 - Reduxion
- PHP 5.3.5 and 5.2.17 Released!**

[06-Jan-2011] The PHP development team would like to announce the immediate availability of PHP 5.3.5 and 5.2.17.

This release resolves a critical issue, reported as PHP bug #53632 and CVE-2010-4645, where conversions from string to double might cause the PHP interpreter to hang on systems using x87 FPU registers.

The problem is known to only affect x86 32-bit PHP processes, regardless of whether the system hosting PHP is 32-bit or 64-bit. You can test whether your system is affected by running this script from the command line.

All users of PHP are strongly advised to update to these versions immediately.
- PHP 5.2.16 Released!**

[16-Dec-2010] The PHP development team would like to announce the immediate availability of PHP 5.2.16. This release marks the end of support for PHP 5.2. All users of PHP 5.2 are encouraged to upgrade to PHP 5.3.

This release focuses on addressing a regression in open_basedir implementation introduced in 5.2.15 in addition to fixing a crash inside PDO::pgsql on data retrieval when the server is down. All users who have upgraded to 5.2.15 and are utilizing open_basedir are strongly encouraged to upgrade to 5.2.16 or 5.3.4.

To prepare for upgrading to PHP 5.3, now that PHP 5.2's support ended, a migration guide available on <http://php.net/migration53> details the changes between PHP 5.2 and PHP 5.3.

For a full list of changes in PHP 5.2.16 see the Changelog at <http://www.php.net/Changelog-5.php#5.2.16>.
- PHP 5.3.4 Released!**

[10-Dec-2010] The PHP development team is proud to announce the immediate release of PHP 5.3.4. This is a maintenance release in the 5.3 series, which includes a large number of bug fixes.
- Security Enhancements and Fixes in PHP 5.3.4:**
 - Fixed crash in zip extract method (possible CWE-170).
- Stable Releases**
 - Current PHP 5.3 Stable: 5.3.5
 - Current PHP 5.2 Stable: 5.2.17
- Upcoming Events [add]**
 - January**
 - Conferences**
 - 28. PHPBentux Conference 2011
 - User Group Events**
 - 09. Los Angeles PHP Developers Group
 - 10. Queen City (Charlotte) PHP
 - 11. Hamburg
 - 11. Dallas PHP/MySQL Users Group
 - 11. Dallas PHP Users Group (DPUG)
 - 11. Austin PHP Meetup
 - 11. OKC PHP Meetup
 - 11. Nashville PHP User Group
 - 11. Oklahoma City PHP User Group
 - 11. Buffalo PHP Meetup
 - 11. South Florida PHP Users

Figura 3. Es posible encontrar información y características de la última versión de PHP en su sitio oficial.

- Mejoras sobre el motor Zend (**Zend Engine II**).
- En cuanto a la programación orientada a objetos, PHP ahora 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 (es importante saber que el código fue rescrito en comparación con la versión inmediatamente anterior).
- Manejo de excepciones (similar a como lo hace Java).

SQLite

SQLite es una librería escrita en lenguaje C que implementa un motor de base de datos accesible por varios lenguajes (PHP, Python, C, etcétera).

No permite múltiples usuarios accediendo a la vez en modo escritura a la base de datos, debido a que el mecanismo de bloqueo que utiliza consiste en bloquear toda la base completa. Por eso, esta librería está especialmente indicada cuando se requiera de una gran rapidez en las consultas y nos baste que un único usuario pueda realizar modificaciones (cabe mencionar que SQLite es entre dos y tres veces más rápido que bases 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.

SQLite es software libre y podremos obtener más información en su sitio oficial, el cual encontramos en la dirección **www.hwaci.com/sw/sqlite**.

MySQLi

Pero a pesar de estos importantes avances, quizás nos interese, particularmente, la inclusión de **MySQLi** (con **i** de “**improved**”, *mejorado*), una extensión que permite acceder a las funcionalidades provistas por MySQL a partir de la versión 4.1.2.

Se ha probado que esta extensión ofrece mayor velocidad (en algunas operaciones es hasta cuarenta veces más rápida que la extensión MySQL anterior).

Se está trabajando para que la migración desde la extensión de MySQL a MySQLi sea lo menos trabajosa posible. Esto se logra, entre otras cosas, manteniendo las funciones –y sus nombres– lo más parecidas que se pueda en ambas extensiones. Si vemos códigos programados con ambas extensiones, no podremos ser capaces de notar grandes diferencias en general.

Podemos encontrar una completa reseña acerca de cómo instalar la extensión MySQLi en la dirección **www.php.net/mysqli**.

Veremos dos características: las consultas preparadas y la seguridad en las conexiones, las cuales analizamos a continuación.

Consultas preparadas

Cuando ejecutamos una instrucción SQL, lo que sucede es, en resumidas cuentas, que se crea la instrucción en PHP, se la envía al servidor de bases de datos (por caso, MySQL), éste controla que la instrucción sea válida y devuelve un resultado (puede ser un conjunto de registros, por ejemplo). Luego PHP trabaja con esos datos.

Debemos saber que si ejecutamos una instrucción **N** veces (por ejemplo, insertar 100 registros), hay que enviar **N** instrucciones a MySQL.

La posibilidad que da MySQL desde la versión 4.1.2 es que, en vez de recibir **N** instrucciones, permite aceptar consultas preparadas: PHP le envía un **molde** de la instrucción, MySQL lo valida y lo guarda; luego PHP no tiene que enviarle la instrucción completa sino que puede mandarle sólo los datos que podrían cambiar de una instrucción a otra. Por ejemplo, trabajando sin consultas preparadas se haría lo siguiente:


```
INSERT INTO tabla1 VALUES (1, "argentina", 100.000);  
INSERT INTO tabla1 VALUES (2, "brasil", 500.000);  
INSERT INTO tabla1 VALUES (3, "uruguay", 300.000);  
INSERT INTO tabla1 VALUES (4, "paraguay", 200.000);  
INSERT INTO tabla1 VALUES (5, "chile", 600.000);
```

En este caso, MySQL hace cinco veces un trabajo similar, esto sin contar el problema que se genera por el tráfico en la red.

En cambio, con consultas preparadas sería:

```
INSERT INTO tabla1 VALUES (?, ?, ?);
```

Así, se envía y se valida una sola vez. Luego enviamos solo los datos que varían:

```
(1, "argentina", 100.000);  
(2, "brasil", 500.000);  
(3, "uruguay", 300.000);  
(4, "paraguay", 200.000);  
(5, "chile", 600.000);
```

Vemos que se envían menos datos.

Lo anterior no está codificado en PHP. Simplemente sirve para ilustrar el concepto de las consultas preparadas (*Prepared Statements* en inglés).

Seguridad en las conexiones

Utilizando las funciones provistas por PHP para acceder a bases de datos MySQL, existía la posibilidad de tener una conexión por defecto: se establecía una conexión y, si en las siguientes instrucciones SQL no se la indicaba expresamente, se asumía la última abierta. Esto puede traer ciertos problemas relacionados a la seguridad, por eso MySQLi requiere que se especifique explícitamente en cada **script** PHP la conexión que se utiliza.

¿Qué versión utilizar?

Si la aplicación ya está funcionando y cumple con sus propósitos, evidentemente 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 más reciente del lenguaje: quizás no utilicemos todas las mejoras ni todas sus características; pero si quisiéramos hacerlo no tendríamos que rescribir el código, ni actualizar a la versión más reciente, ni entrar en temas en ocasiones engorrosos como es la compatibilidad del lenguaje entre dos versiones distintas del mismo. Aunque a veces se diga que esta decisión depende de los requerimientos del desarrollador, PHP versión 5 es un tema aparte porque, como se dijo, ha sido mejorado y no de forma cosmética: el soporte mejorado para la programación orientada a objetos es algo que ya es 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 PHP, más en carrera estará a la hora de actualizar sus conocimientos e incluso al momento de buscar empleo.

Si tenemos instalada una versión de PHP y no sabemos cuál es, es posible recurrir a la función denominada **PHPVERSION()**:

```
<?php
// EJEMPLO DE PHPVERSION()
echo "Version de PHP: ".phpversion();
?>
```

EXTENSIONES EN PHP

Cuando programamos en un lenguaje –no sólo en PHP–, normalmente nos valemos de funciones o procedimientos, ya sea para resolver problemas o modular el código para hacerlo más legible y reutilizarlo sin escribir lo mismo más de una vez. Las extensiones no son ni más ni menos que conjuntos de funciones que tenemos disponibles para programar pero, para ser más precisos, podríamos dividir las funciones en dos grupos: las funciones que vienen incorporadas con el lenguaje

EXTENSIÓN DE LA DISTRIBUCIÓN

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

(llamadas **built in**) y las que están en las bibliotecas añadidas, que se tienen que incluir en el sistema de forma específica.

Las extensiones —o bibliotecas— componen el segundo grupo. Se podría decir que para utilizar ciertas funciones hay que **extender** el lenguaje.

Una vez que se instalan y se habilitan esas bibliotecas (ya veremos cómo hacerlo), el comportamiento de las funciones componentes dentro del código de nuestros 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, es decir que 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.

Para instalar estas extensiones, tenemos que seguir una serie de pasos: el primero de ellos es abrir el archivo **php.ini** (si estamos trabajando con PHP versión 3 el archivo, probablemente, se llame **php3.ini**; desde la versión 4 en adelante se ha convenido en llamarlo simplemente **php.ini**), buscar la línea que comienza con **extension_dir** y reemplazarla por **extension_dir = "c:\PHP\extensions"**, en caso de que **C:\PHP\extensions** sea en donde tenemos almacenadas las extensiones que vienen incluidas con PHP. Para saber en qué lugar se encuentran las extensiones, debemos buscar dentro del directorio en donde instalamos PHP una carpeta llamada **extensions** (o **ext**) y copiar la ruta completa hasta allí a la línea **extension_dir** del **php.ini**, tal como se mostró anteriormente.

Guardar las extensiones en un solo lugar implica el beneficio de no tener que indicar la ruta completa a la extensión cada vez que queramos usarla. Según estemos trabajando sobre sistemas **Linux/Unix** o **Windows** habrá diferencias tales como:

- Linux usa la barra / para acceder a subdirectorios (el directorio de extensiones podría ser **/usr/lib/PHP3**), mientras que Windows usa la barra \.
- Las extensiones en Linux son archivos **.SO**, en Windows son **.DLL**.

Ahora sólo nos queda habilitar las extensiones que vayamos a usar. Lo único que debemos hacer es buscar dentro del archivo **php.ini** una lista del tipo (con archivos terminados en **.SO** o **.DLL** según corresponda):



CÓDIGO DE BARRAS

Una de las equivocaciones más frecuentes a la hora de publicar nuestros archivos PHP en un servidor con un sistema operativo diferente del nuestro, es la de escribir las rutas a archivos o directorios de forma incorrecta. Cabe recordar que los sistemas Unix compatibles usan / para acceder a subdirectorios, y los sistemas Windows usan \.

```
;extension=pgsql.so
;extension=mysql.so
;extension=gd.so
;extension=imap.so
;extension=ldap.do
;extension=xml.so
;....
```

Y quitar el punto y coma inicial para habilitar las bibliotecas que queramos usar. Para que los cambios tengan efecto habrá que reiniciar el servidor web.

Se pueden habilitar múltiples extensiones, incluso no existe ningún inconveniente en, por ejemplo, utilizar dos bibliotecas con el objetivo de acceder a diferentes tipos de bases de datos al mismo tiempo.

Bases de datos

PHP tiene extensiones para soportar, entre otras, las siguientes bases de datos:

- DBase
- Informix
- Interbase
- MS SQL Server
- Mysql
- msql
- Oracle
- Postgre SQL
- Sybase

Incluso con **ODBC** (*Open Data Base Connectivity, Conectividad Abierta de Bases de Datos*) se puede acceder a casi cualquier base de datos existente en el mercado: ODBC brinda un conjunto de comandos que son traducidos a instrucciones específicas de una base de datos en particular a través de drivers provistos por éstas.

Es claro que utilizar funciones nativas da más réditos, en cuanto a la velocidad de respuesta, comparado con trabajar con algún mediador tipo ODBC.



INSTALACIÓN

Hay varios programas que nos ofrecen descargar PHP, MySQL y Apache en un solo archivo, e instalarlos a través de un programa que nos guía y nos hace preguntas acerca de dónde queremos instalar cada elemento, qué tipo de instalación preferimos, etc., tal como lo hacen muchas aplicaciones que trabajan bajo Windows.

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 éstos, por no estar implementadas. Por lo tanto si se quiere utilizar todo el poder de la base de datos, es preferible acceder con funciones nativas para ésta, como las que ofrece PHP en sus extensiones. Por ejemplo, si en nuestro proyecto sabemos que vamos acceder sólo a la base de datos **Postgre SQL**, no hay necesidad de acceder con ODBC ya que PHP nos provee de una extensión (**php_pgsql.dll**) para hacerlo de forma nativa.

Ventajas de trabajar con extensiones

Sólo cargamos las bibliotecas cuando las usamos: PHP ya tiene demasiadas funciones incorporadas y sería poco recomendable iniciar el motor para soportar cientos de funciones de las cuales probablemente necesitemos sólo algunas. Además, al añadir una biblioteca no hace falta reinstalar PHP, sólo habilitar desde el archivo `php.ini` lo que necesitamos: esto significa modularidad.

Bibliotecas incorporadas

El lenguaje PHP incorpora sin necesidad de ningún tipo de instalación ni habilitación extras las funciones que listamos a continuación:

- Para manejo de matrices.
- Funciones matemáticas.
- BCMath (Desde PHP 4.0.4. Más funciones matemáticas).
- Para manejo de Clases/Objetos.
- Para manejo de variables de tipo de carácter.
- Para tratamiento de Fecha y Hora.
- Para acceso directo a Entrada/Salida.
- Funciones de directorio.
- Funciones de Gestión de Errores y Registros.



MIRAR Y APRENDER

Es importante tener en cuenta que se logra un mayor entendimiento del lenguaje cuando nos encargamos de leer código escrito por otras personas: un mismo problema puede resolverse de muchas maneras, y no quedarse sólo con un punto de vista ayuda a abrir nuestra mente e incorporar nuevas formas de encarar la escritura de un código.

- Funciones de Sistema de Archivos.
- Para utilizar el protocolo FTP.
- Para utilizar el protocolo HTTP.
- Funciones de correo.
- Funciones de Red.
- Funciones de Control de Salida.
- Para ejecución de Programas.
- Funciones para el manejo de sesiones.
- Funciones de secuencia.
- Funciones de cadenas.
- Funciones URL.
- Para manejo de Variables.

Para poder tener acceso a las demás bibliotecas, tendremos que activarlas a través del archivo **php.ini**, o bien incorporarlas al momento de compilar PHP e instalar las bibliotecas en forma separada (únicamente para aquellos sistemas operativos en los que para instalar PHP haya que compilarlo).

Para ver 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 biblioteca en particular en nuestro sistema, bastará con buscar la referencia en el manual de PHP, el cual encontramos en la dirección **www.php.net/manual**.

LAMP

En revistas especializadas, libros y/o artículos en Internet encontraremos referencias a los sistemas LAMP. Esta sigla se refiere a Linux, Apache, MySQL, y alguno de los lenguajes Perl/PHP/Python. Los sistemas tipo LAMP son muy utilizados en la actualidad, entre otras razones, debido a la estabilidad, la potencia y el bajo costo de desarrollo de las herramientas que los componen.

FACILIDAD DE APRENDIZAJE

Esto evidentemente no equivale a hacer un juicio de valor sobre otros lenguajes, simplemente se trata de una cualidad de PHP.

PHP se caracteriza por ser un lenguaje cuyo aprendizaje se vuelve sencillo incluso para aquéllos que nunca han trabajado con ningún otro lenguaje de programación, pero está claro que tener conocimientos previos ayuda a entender más rápidamente qué se está haciendo y cómo se está haciendo.

Un punto importante es que la sintaxis de PHP deriva o es similar a la del **lenguaje C**, que es realmente muy popular: el que tenga conocimientos en este lenguaje se acercará con más facilidad al uso de PHP.

La meta de este lenguaje es permitir escribir a los creadores de sitios web páginas dinámicas de una manera rápida y fácil, pero si se quiere ir más allá y llegar a desarrollar aplicaciones complejas necesitaremos avanzar dentro del lenguaje: quizás PHP sea fácil de aprender pero este aprendizaje debe ser constante.

PORTABILIDAD

PHP es un lenguaje multiplataforma, lo que significa que está preparado para trabajar sobre distintos sistemas operativos. Más adelante en este mismo capítulo podremos ver un listado de los sistemas operativos soportados.

Pero la portabilidad está también en que no es necesario realizar grandes modificaciones al código fuente de una aplicación escrita en PHP al momento de trasladarla de una plataforma a otra: si lo deseamos podemos desarrollar nuestra aplicación en Windows o MAC, pero luego subir el mismo código a un servidor que esté corriendo Linux, por ejemplo.

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 corre en una gran cantidad de sistemas operativos y sin necesidad de un componente adicional que debamos comprar.

Debemos tener en cuenta que el lenguaje de programación PHP se encuentra disponible para los siguientes sistemas operativos:

- Unix/HP-UX
- Unix/Linux
- Unix/OpenBSD
- Unix/Solaris
- Unix
- Windows (todas las versiones, salvo PHP5 que no corre bajo Windows 95)
- MAC OS

Actualmente, PHP 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, Oreilly Website Pro, Sambar, Xitami, Caudium, Netscape Enterprise Server, THTTPD, y otros.

Las posibles incompatibilidades entre plataformas son las siguientes:

- Ruta a archivos/directorios.
- Bibliotecas que sólo funcionan en algunos sistemas, por ejemplo:
 - Funciones W32api (sólo para plataformas Windows de 32 bits).
 - Funciones para el manejo de Impresoras (sólo en Windows 9x, ME, NT4 y 2000).
 - Las funciones COM para Windows.
 - Funciones de acceso directo a E/S (no disponibles para sistemas Windows).
 - Funciones GMP (funciones 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 existen bibliotecas que ofrecen compatibilidad con Windows NT/2000/XP/7 pero no con las versiones de Windows 9.x.

Normalmente cuando no recordamos si una biblioteca es o no compatible con nuestro sistema, basta con buscar la referencia en el manual de PHP, **www.php.net/manual/** y verificar si hay algún requerimiento.

... RESUMEN

En este capítulo hemos visto una reseña de las principales virtudes del lenguaje de programación PHP: qué hace, para qué sirve, en qué se usa mayormente, qué versiones existen. También vimos cuáles son los requisitos para instalarlo y cómo incorporar extensiones para hacer nuestro trabajo más fácil. En definitiva, un punto de partida para conocer un lenguaje que está dando mucho que hablar y que seguirá haciéndolo.



TEST DE AUTOEVALUACIÓN

1 ¿Cuánto dinero hay que pagar para poder usar PHP?

2 ¿Qué es una extensión?

3 ¿Cuál es la ventaja principal de trabajar con extensiones?

4 ¿En qué directorio de su sistema se encuentran las extensiones de PHP?

5 ¿En qué versión de PHP se incorporó la extensión MySQLi? ¿Para qué sirve?

6 ¿Podría nombrar cuatro razones por las cuales usaría PHP para programar sus aplicaciones?

7 ¿PHP sólo sirve para programar desarrollos web?

8 Nombre tres de las nuevas características que vienen con PHP 5.

9 Brevemente, ¿qué es SQLite?

10 ¿Desde qué sitio se puede descargar la última versión de PHP?

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.

Bases de Datos relacionales

Desde hace algún tiempo a esta parte se le da cada vez más importancia al hecho de reunir la mayor cantidad de información posible para cuando llegue el momento de tomar decisiones. Justamente, los sistemas de apoyo a la toma de decisiones y su diseño son parte fundamental de este capítulo.

¿Qué es y para qué sirve una Base de Datos?	32
Arquitectura de Bases de Datos	32
Modelo de datos	32
Modelo jerárquico	33
Modelo de red	34
Modelo relacional	35
Modelo orientado a objetos	36
Operaciones relacionales básicas	37
Diseño de la Base de Datos	42
Tabla	42
Relación	43
Atributo	43
Claves primarias	44
Claves candidatas	45
Vistas	46
Esquemas	46
Índices	47
Tipos de relaciones entre tablas	52
Cardinalidad Uno a Uno	52
Cardinalidad Uno a Varios	53
Cardinalidad Varios a Varios	53
Normalización	56
Datos redundantes	57
Primera Forma Normal	59
Segunda Forma Normal	59
Tercera Forma Normal	61
MySQL y el soporte para el modelo relacional actual	61
Integridad referencial	62
Subconsultas	67
Resumen	67
Actividades	68

¿QUÉ ES Y PARA QUÉ SIRVE 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 una base de datos por sí misma no nos produce ninguna mejora: el beneficio se logra diseñándola correctamente 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 las acciones de los mismos. Es una representación de los datos desde el punto de vista de una organización.
- **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

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 en concreto 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 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, los cuales detallamos a continuación:

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

A continuación, haremos un análisis de todos ellos para conocer sus diferencias y sus características más sobresalientes en lo que se refiere al diseño de bases de datos.

Modelo jerárquico

Las bases de datos que ingresan dentro de este modelo organizan su información utilizando niveles de jerarquías. Cada nivel de jerarquía puede tener un número **N** de nodos, con la particularidad de que cada uno no puede tener más de un padre.

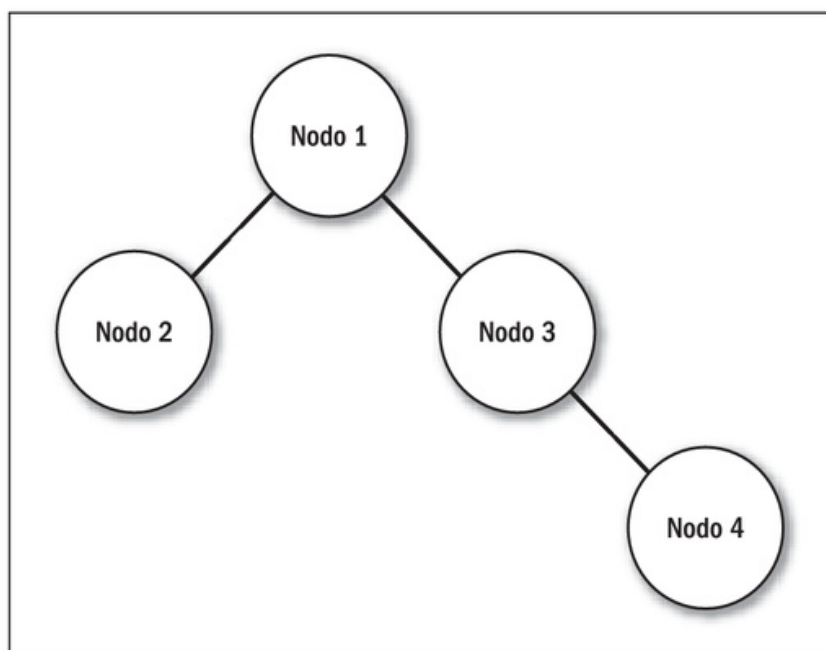


Figura 1. Modelo jerárquico.

En la **Figura 1**, el **nodo 1** no tiene padre –cuando se da esta situación se lo llama **nodo raíz**– y tiene dos hijos. De esos hijos sólo hay uno que es padre.

Haciendo un paralelismo entre los tipos de relaciones que admite el modelo relacional, el modelo jerárquico admite relaciones uno a varios y uno a uno.

Este modelo tiene muchos puntos en común con las estructuras de tipo árbol.

III PENSAR EN EL FUTURO

Debemos tener en cuenta que diseñar una base de datos puede ser la parte que más tiempo nos demande dentro de un desarrollo de importancia. Por esta razón es conveniente prestar atención a todos los detalles, ya que cambiar la estructura de la base de datos en medio de un desarrollo no es lo más recomendable por los problemas que pueden generarse.

En comparación con el **modelo relacional** (ver más adelante) aquí sí importa, como en el modelo de red, el orden en que se ubiquen los datos.

Uno de los inconvenientes del modelo jerárquico consiste en la redundancia de datos. Supongamos una relación **Maestro-Detalle**, como es **pedidos** y **productos**. Cada pedido puede contener más de un producto y, a su vez, cada producto puede encontrarse contenido en más de un pedido. Si intentamos simular esta situación por medio del **modelo jerárquico**, llegaremos a la conclusión de que es imposible, a menos que repita informaciones.

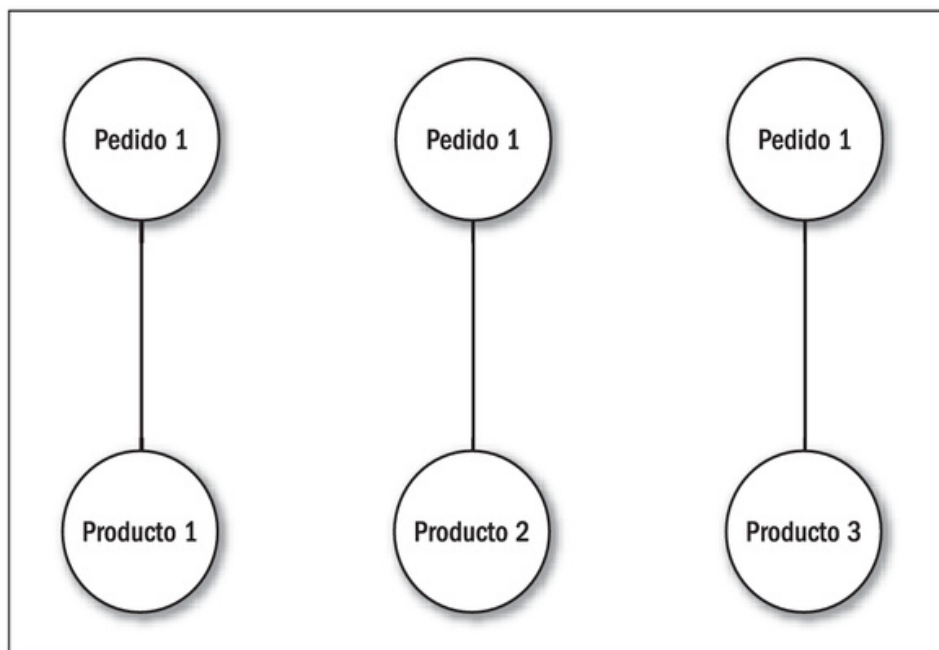


Figura 2. Maestro-Detalle en el modelo jerárquico.

Un problema aparte es el de eliminar un **nodo padre**. Se tendría que eliminar toda su descendencia. Esta clase de problemas se solucionan en los demás modelos.

Modelo de red

Es ligeramente similar al modelo jerárquico. Una de las diferencias fundamentales es que aquí un nodo puede tener varios padres.

III MEZCLA

Si bien el modelo de datos más utilizado en la actualidad es el conocido como **relacional**, en determinados sectores está imponiéndose el modelo llamado **orientado a objetos**. Por otra parte, es importante saber que ambos toman cosas del otro, y aunque mantienen sus principios fundamentales, intentan dar a los usuarios y desarrolladores lo que éstos buscan.

Administrar la información en una base de datos de este tipo puede llegar a ser demasiado difícil, debido a que existe la posibilidad de que haya una gran cantidad de interrelaciones entre nodos.

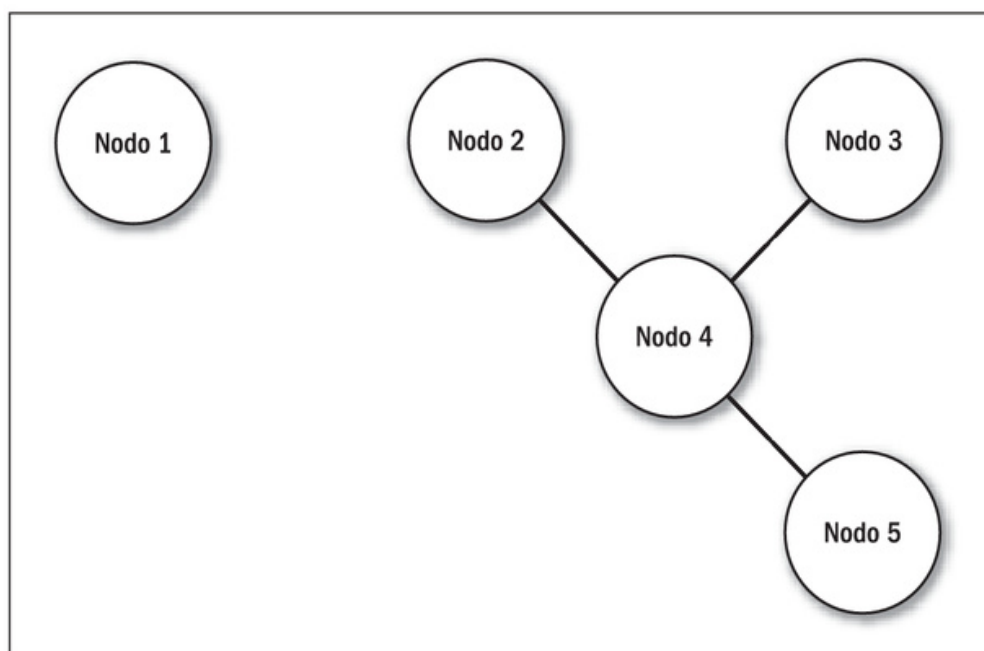


Figura 3. Modelo de red.

En la imagen anterior se pueden observar algunas de las posibles relaciones existentes entre nodos: el **nodo 2** y el **nodo 3** son padres del **nodo 4**, que a su vez es padre del **nodo 5**. El **nodo 1** no tiene padres ni hijos.

Este modelo admite relaciones uno a varios y uno a uno.

Modelo relacional

Se trata, sin dudas, del modelo más popular desde hace un tiempo, y con él se imponen conceptos tales como tabla –arreglo bidimensional–, fila y columna.

Es un modelo relativamente nuevo (tomó notoriedad en 1970 y su autor, **Edgar Frank Codd**, murió el 18 de abril del año 2003) y quizás el único que llegó tanto a los usuarios finales como también a los desarrolladores.

Se recuperan los datos por medio de lenguajes de programación de consulta (el más popular es SQL, pero también existen otros lenguajes de este tipo) que mantienen la compatibilidad aun entre sistemas gestores de bases de datos de distintas compañías e incluso entre sistemas operativos diferentes.

Entendemos el modelo relacional como una propuesta de ver los datos como si se trataran de objetos del mundo real, diferenciables entre sí por sus características básicas. Un objeto dado puede ser descrito por la colección de características que tiene (denominadas atributos), y diferenciable a partir de eso mismo de otros objetos. Este modelo admite relaciones uno a varios, uno a uno, y varios a varios.

En este modelo las tablas deben cumplir las siguientes reglas:

- Cada fila debe ser única.
- Cada columna debe ser única.
- Los valores de las columnas deben pertenecer al dominio de cada atributo.
- Debe tener un solo tipo de fila.
- El valor de la columna para cada fila debe ser único.

Modelo orientado a objetos

Al intentar trabajar con sistemas de información geográfica o también con sistemas multimedia, los modelos anteriormente mencionados no se sienten demasiado cómodos e incluso algunos ni siquiera pueden soportarlos por, entre otros, los motivos que se enumeran a continuación:

- La estructura de los objetos es más compleja;
- Las transacciones son de larga duración;
- Se necesitan nuevos tipos de datos para almacenar imágenes y textos;
- Hace falta definir operaciones no estándar, específicas para cada aplicación.

Todo esto sin contar con el hecho de que los lenguajes de programación orientados a objetos están teniendo un gran auge, y aprovechar una base de datos que comparta los mismos principios no es nada desdeñable.

Incluso muchas bases de datos relacionales –por ejemplo, **PostgreSQL**– están implementando varios conceptos propios de los orientados a objetos. Estas bases son llamadas a veces **Objeto-Relacionales**.

Las bases de datos orientadas a objetos en lugar de incorporar tablas como lo hacen las relacionales, utilizan objetos. Los objetos de una base de datos tienen las mismas características conocidas de los objetos de los lenguajes orientados a objetos (**herencia, polimorfismo, encapsulación**, otros).

Este modelo admite relaciones uno a varios, uno a uno, y varios a varios.



ORDEN Y PROGRESO

Es importante mencionar que si no tenemos demasiada experiencia en lo que se refiere al diseño y la implementación de bases de datos, es recomendable que comience a hacerlo de forma pausada –pero progresiva–, poniendo en práctica ejemplos simples para luego ampliarlos en funcionalidades o desarrollando otros más complejos.

OPERACIONES RELACIONALES BÁSICAS

Estas operaciones forman parte de lo que se considera el álgebra relacional y sirven para crear nuevas relaciones a partir de las ya existentes.

Si conocemos el lenguaje SQL encontraremos semejanzas directas con respecto a las operaciones que pasaremos a describir y es exactamente así porque este lenguaje no es más que una implementación de estas operaciones.

El álgebra relacional –junto con el cálculo relacional– fueron definidos por Codd como la “base de los lenguajes relacionales”. Se puede decir que el álgebra es un lenguaje procedural (de alto nivel), mientras que el cálculo relacional es un lenguaje no procedural. Sin embargo, ambos lenguajes son equivalentes: para cada expresión del álgebra, se puede encontrar una expresión equivalente en el cálculo, y viceversa.

El álgebra relacional es un lenguaje con una serie de operadores que trabajan sobre uno o varios conjuntos de datos –podemos asumir que son tablas– para obtener otro conjunto como resultado, sin que cambien los originales. La salida de una operación puede ser la entrada de otra operación. Esto permite anidar expresiones de álgebra del mismo modo que se pueden anidar las expresiones aritméticas (esta propiedad se denomina **clausura**).

Debemos saber que de los nueve operadores, sólo hay cinco que son fundamentales: **selección**, **proyección**, **producto**, **unión** y **diferencia**, que permiten realizar la mayoría de las operaciones de obtención de datos. Los operadores no fundamentales son **reunión**, **intersección**, **asignación** y **división**, que se pueden expresar a partir de los cinco operadores fundamentales.

A continuación se presenta una descripción de las nueve operaciones:

- | | |
|-----------------|---------------|
| 1. Unión | 6. Proyección |
| 2. Intersección | 7. Reunión |
| 3. Diferencia | 8. División |
| 4. Producto | 9. Asignación |
| 5. Selección | |

III CONJUNTOS

Habremos notado que algunas de las operaciones relacionales básicas son propias de las operaciones entre conjuntos y esto es, precisamente, porque de allí provienen. De hecho, podemos pensar una base de datos como una serie de conjuntos de datos y, a su vez, que cada uno de esos conjuntos posee subconjuntos que agrupan a los datos.

Para unión, intersección y diferencia supondremos los conjuntos de la **Figura 4**.

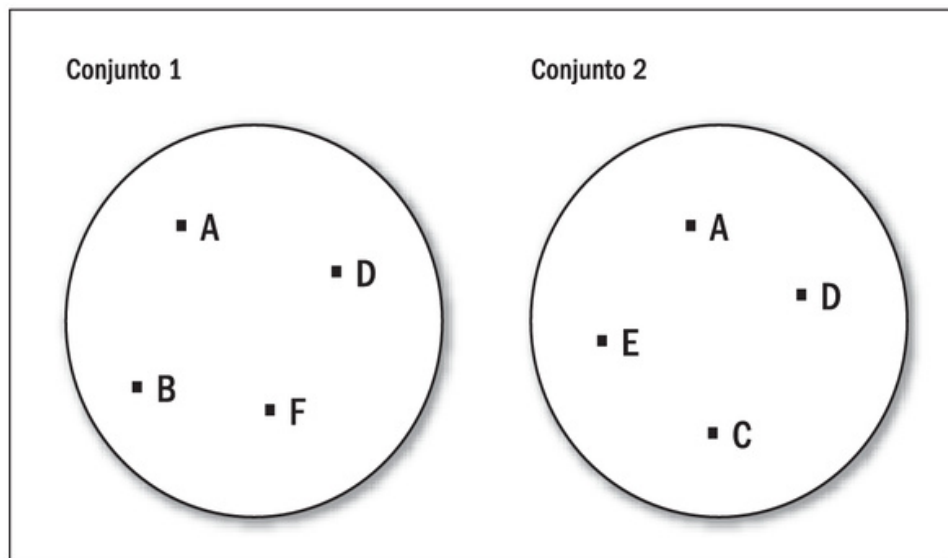


Figura 4. Situación inicial.

1. **Unión:** debemos saber que esta operación permite combinar datos de distintas tablas que tienen la misma estructura.

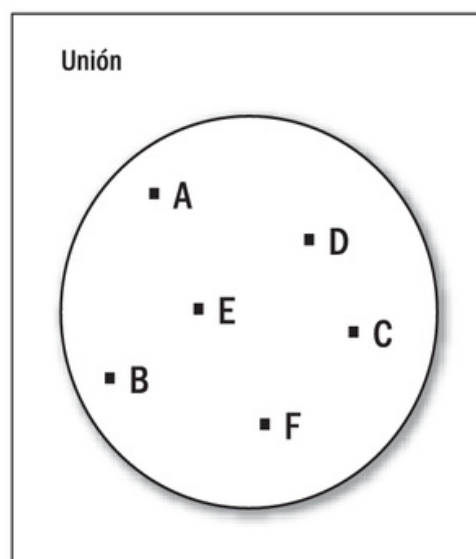


Figura 5. Unión.



PONER EN PRÁCTICA LOS CONOCIMIENTOS

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

2. Intersección: esta operación permite obtener filas comunes a varias tablas.

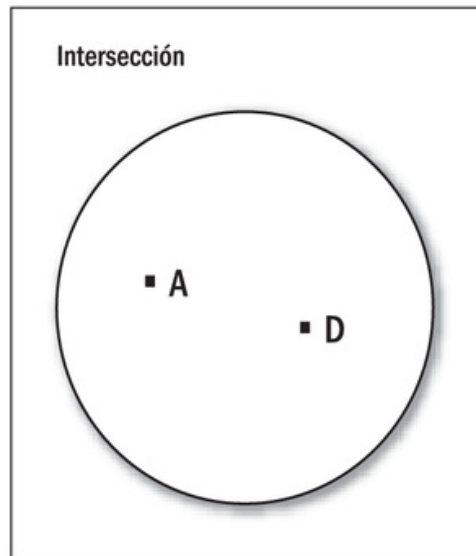


Figura 6. Intersección.

3. Diferencia: permite obtener filas que figuran en una tabla pero no en otra.

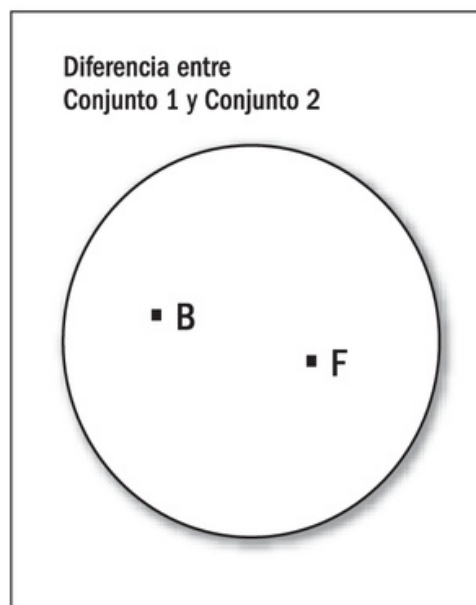


Figura 7. Diferencia.



EL SÍMBOLO

Para diseñar bases de datos generalmente se utilizan diagramas que describen de forma gráfica las estructuras de las tablas, las relaciones, las claves primarias, las claves foráneas, etc. Si bien hay símbolos clásicos para implementar en este tipo de diagramas, lo más importante es entender el funcionamiento del sistema, más allá de la simbología utilizada.

- 4. Producto:** nos permite obtener todas las posibles combinaciones (producto cartesiano) entre las dos tablas correspondientes.

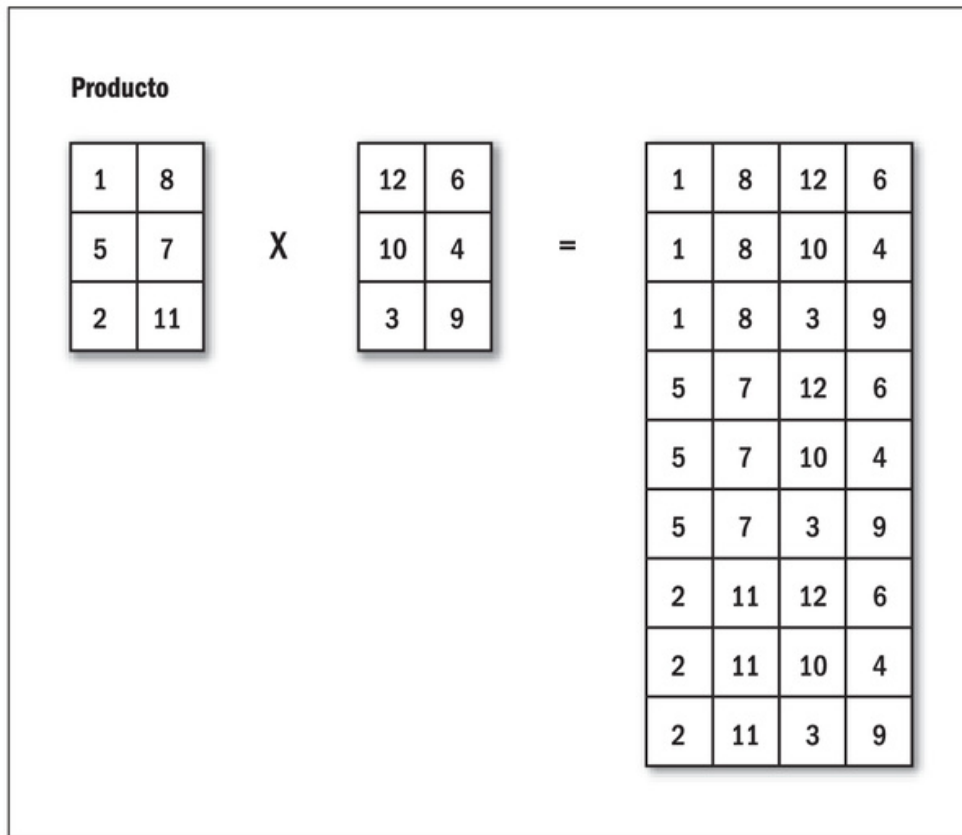


Figura 8. Producto.

- 5. Selección:** esta operación nos permite obtener un conjunto de filas de una o más tablas que satisfacen una condición especificada.

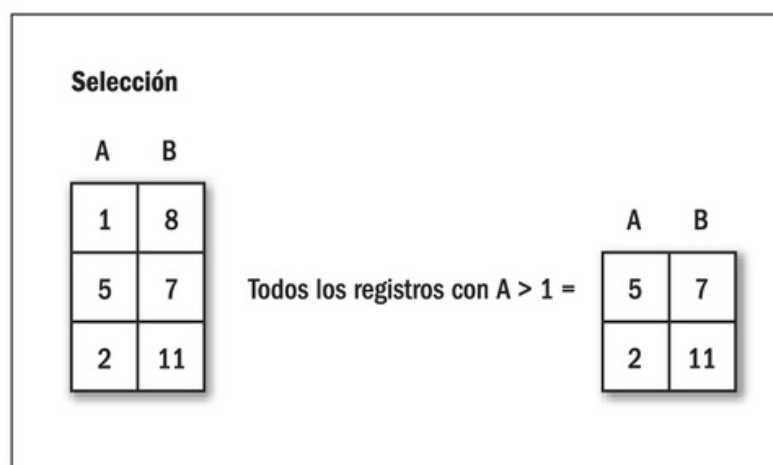
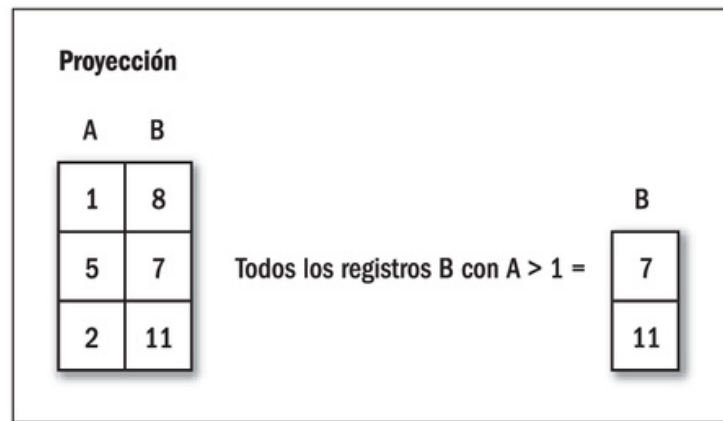
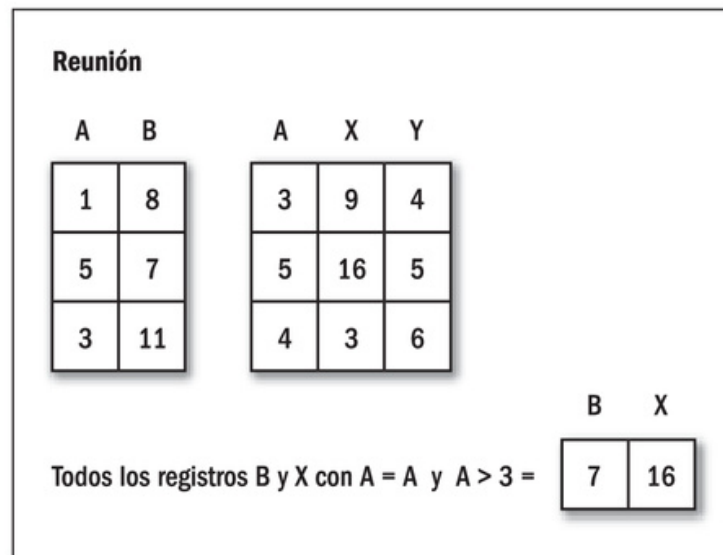


Figura 9. Selección.

- 6. Proyección:** es un caso concreto de la operación de selección, y permite discriminar las columnas que deseamos obtener.

**Figura 10. Proyección.**

7. Reunión: la operación unión permite obtener datos de distintas tablas con la misma estructura. La operación reunión permite obtener datos de distintas tablas con distintas estructuras. Su equivalente en SQL es JOIN.

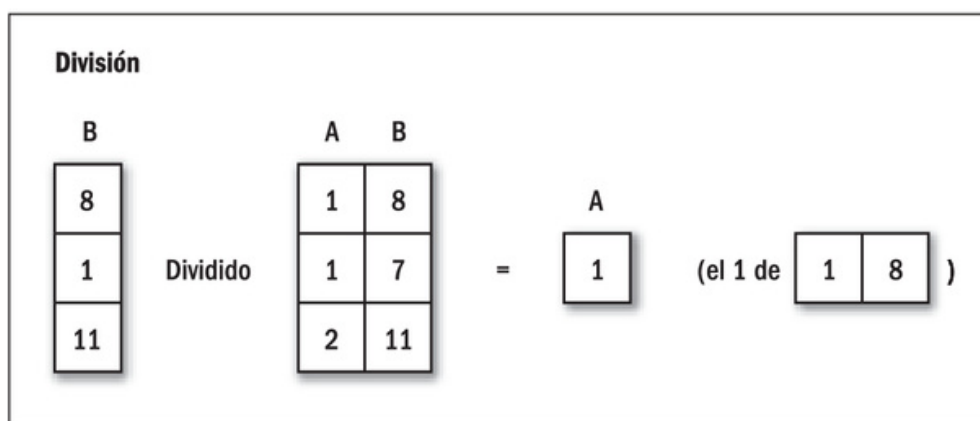
**Figura 11. Reunión.**

8. División: si dividimos una tabla B por una tabla A obtendremos como resultado una tercera tabla que contendrá los campos de la tabla B que no existen en la tabla A y las filas que están asociadas en la tabla B con cada fila de la tabla A.

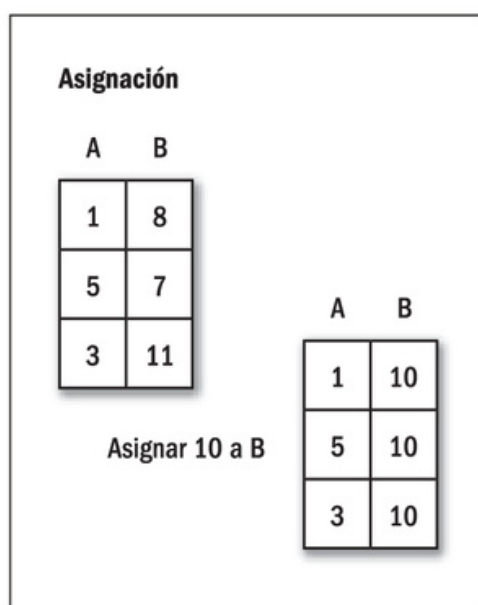


PRIMERA INSTANCIA

Debemos tener en cuenta que para poder poner en práctica los ejemplos entregados en este capítulo debemos antes instalar y poner en marcha el servidor web, el servidor de bases de datos y el lenguaje PHP. En este libro se incluyen notas referidas a estos temas, de forma que el lector pueda realizar estas tareas por su cuenta y sin complicaciones.

**Figura 12. División.**

9. Asignación: debemos saber que la utilidad de esta operación consiste en asignar valores a uno o más registros de una tabla.

**Figura 13. Asignación.**

DISEÑO DE LA BASE DE DATOS

Si bien las bases de datos están cada vez más cerca de los usuarios finales, siguen manejando una terminología propia y en esta sección intentaremos introducir algunos de los conceptos fundamentales para luego avanzar sobre otros temas.

Tabla

Una tabla es una colección de una o más columnas y cero o más filas. Las tablas pueden ser entendidas como una estructura de datos simple que se asemeja bas-

tante a una matriz de dos niveles: el primero podría representar el número de fila y el segundo, el número de columna. Para poder acceder a un valor de esta matriz debemos indicar alguno de estos dos datos.

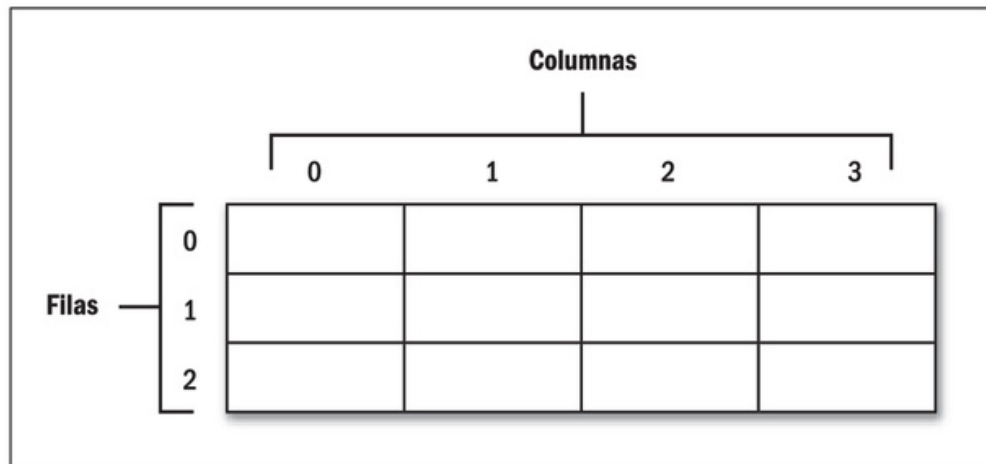


Figura 14. Estructura de datos Tabla.

Como se dijo, una tabla nos permite organizar los datos en **filas** (también llamadas tuplas o registros) y **columnas** (atributos, campos).

Cada una de estas filas (así como los datos que se encuentran contenidos en ella) debe ser distinta con respecto a las demás.

Cada columna dentro de una fila es el valor de un atributo de esa fila.

Relación

Una relación es una vinculación entre ideas, una forma de asociar entidades para lograr el objetivo en común que tienen. Una relación es una asociación entre entidades. Existen varios tipos de relaciones, que se verán más adelante.

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



ACTUALIZACIÓN

Es importante saber que la extensión denominada MySQLi provee grandes mejoras con respecto a la extensión anterior y, a la vez, intenta mantener la compatibilidad de las funciones ya existentes en versiones anteriores. Más información y una completa guía de referencia de las funciones de esta nueva extensión en el **Apéndice C** de este libro.

nombre, un dueño, un color de pelo, una edad, una comida preferida, y demás. Todos estos son atributos de un perro.

Nótese que no definimos ningún valor particular para estas características: el atributo es el nombre del perro y no un perro. **Rosita** es una **instancia** del atributo nombre.

Claves primarias

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

Suponiendo la tabla libros, tenemos la siguiente estructura:

Nombre
Cantidad_paginas
Editorial
ISBN
Autor

En este ejemplo, si repasamos los atributos uno por uno notaremos que sólo hay uno que no va a repetirse y es ISBN. Podemos tomarlo como clave primaria: ya que no habrá dos libros con el mismo ISBN, no habrá dos filas iguales.

Debemos saber que 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. Más adelante, en este mismo capítulo, en la sección **Normalización**, se responderán ésta y otras preguntas.

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 implican-



PARA PRESTAR ATENCIÓN

La versión 4.1 –junto a la 5– de MySQL marca un verdadero quiebre en cuanto a las funcionalidades prestadas por este gestor de bases de datos. Una muestra de esto es la inclusión en PHP de una nueva extensión llamada MySQLi, que viene con la distribución desde la versión 5 de PHP en adelante, y cuyo objetivo es soportar las nuevas funciones de MySQL.

cia para lo que es un libro: sólo se trata de un atributo con una restricción básica que será el hecho de no tener la capacidad de repetirse.

Este caso también se relaciona con las **técnicas de normalización**. Simplemente, 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:

Nombre_alumno
Direccion
Curso
Colegio
Nacionalidad

En este ejemplo, no hay ningún atributo que no pueda repetirse en los distintos alumnos. La solución es crear una clave y asignarla a cada alumno. La llamaremos **codigo_alumno** y al crearla nos aseguramos que no pueda tomar valores repetidos. La sintaxis utilizada en **MySQL** para crear claves primarias es la siguiente:

primary key (columna1 [, columna2 ...])

Por ejemplo:

```
CREATE TABLE tabla (
    Campo1 INT NOT NULL,
    Campo2 VARCHAR(30) NOT NULL,
    PRIMARY KEY (Campo1)
);
```

Las columnas que se definan como claves primarias no pueden admitir valores nulos.

Claves candidatas

Acabamos de ver que una fila puede ser 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 de esto nos permitirá identificar y localizar un registro de manera rápida y organizada.

Para identificar las claves candidatas de una tabla no hay que fijarse en un estado o instancia de la base de datos: que en un momento dado no haya duplicados para un atributo o conjunto de atributos, no garantiza que los duplicados no sean posibles. No obstante, vale la recíproca: la presencia de duplicados en un estado de la base demuestra que cierta combinación de atributos no es una clave candidata válida.

Vistas

En ciertas ocasiones, no nos interesa que todos los usuarios de una base de datos puedan ver todos los datos de la base (ya sea por seguridad, orden o, simplemente, para mostrar a cada usuario sólo lo estrictamente necesario para realizar su trabajo). Para esto existen las llamadas **vistas** que en resumidas cuentas lo que nos permiten es **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	Direccion_e

Y también tenemos un sector en nuestra empresa que se dedica a enviar tarjetas de felicitación a los empleados, seguramente no se necesitará conocer el sueldo de cada uno de ellos, así que sólo permitiremos que se vean los siguientes datos a través de una vista definida especialmente para este propósito:

Vista Tarjeta_Empleados			
Cod_e	Nom_e	Ape_e	Direccion_e

MySQL implementa las vistas desde la versión 5.0.1. De esta forma podemos crear vistas a partir de consultas de la siguiente manera:

```
CREATE VIEW nombre_vista AS SELECT col1, col2 FROM
tabla1;
```

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 éstas, los

dominios sobre los que se definen estos atributos, las claves primarias y las claves foráneas (la definición y ejemplos acerca de claves foráneas las encontraremos más adelante, en este mismo capítulo).

Índices

Un índice es una estructura de datos en donde se almacena información extra acerca de una columna (una columna indexada). Cuando MySQL encuentra un índice en una columna, lo usará en vez de recorrer todas las posiciones de la columna. MySQL se encarga de utilizar índices para encontrar filas con valores específicos en alguna de sus columnas de manera más rápida.

Debemos saber que existen en MySQL cuatro tipos de índices distintos: **de clave primaria**, **únicos**, **fulltext** e **índices normales**.

Índices de clave primaria

Estos índices se crean, justamente, al momento de definir como clave primaria –ver una de las secciones anteriores, Claves Primarias– una o más columnas de una tabla.

Índices normales

A menos que haya sido definido como **unique** (único), un índice normal admite valores duplicados, a diferencia de lo que sucede con los índices de clave primaria. Existen varias maneras en MySQL de crear índices normales, veamos algunas:

index nombre_indice (columna1 [, columna2 ...])

Por ejemplo:

```
CREATE TABLE tabla (
  Campo1 INT NOT NULL,
  Campo2 VARCHAR(30) NOT NULL,
  PRIMARY KEY (Campo1),
  INDEX indice1 (Campo2)
);
```

create index indice on tabla(columna1 [, columna2, ...])

```
CREATE INDEX indice1 ON tabla (Campo2);
```

La última es a través de un **ALTER TABLE**:

```
ALTER TABLE tabla ADD INDEX indice1 (Campo2);
```

Índices Fulltext

MySQL puede indexar cualquier tipo de dato: estos tipos de índices pueden contener campos de tipo **CHAR**, **VARCHAR** y **TEXT**. Los índices de este tipo están diseñados para facilitar y optimizar la búsqueda de palabras clave en tablas que tienen grandes cantidades de información en campos de texto. Se crean de las siguientes maneras:

- a través de la sintaxis **CREATE FULLTEXT INDEX**, similar a **CREATE INDEX**,
CREATE FULLTEXT INDEX nombre_indice ON tabla (columna1 [, columna2, ...]);

Por ejemplo:

```
CREATE FULLTEXT INDEX indice1 ON tabla (Campo2);
```

- O al momento de crear la tabla, como por ejemplo:

```
CREATE TABLE tabla (  
    Campo1 INT NOT NULL,  
    Campo2 CHAR(30) NOT NULL,  
    PRIMARY KEY (Campo1),  
    FULLTEXT indice1 (Campo2)  
);
```

- Otra opción sería hacer un **ALTER TABLE** sobre una tabla ya existente:

```
ALTER TABLE tabla ADD FULLTEXT indice1 (Campo1);
```

UNIÓN

En la actualidad, se pone de manifiesto la unión que existe entre el lenguaje PHP y el servidor de bases de datos MySQL, a pesar de algunas diferencias en cuanto a la forma de licenciamiento surgidas hace algún tiempo. PHP presta especial atención a los avances de MySQL, y trabaja para incorporarlos y soportarlos en su propia versión.

En las columnas **CHAR** y **VARCHAR** es posible que definamos que no se use la extensión completa del campo sino sólo una parte.

Para crear un índice como parte de un campo, sólo se tiene que especificar el tamaño entre paréntesis después del nombre de la columna:

```
CREATE TABLE tabla (
    Campo1 INT NOT NULL,
    Campo2 CHAR(30) NOT NULL,
    PRIMARY KEY (Campo1),
    FULLTEXT indice1 (Campo2(10))
);
```

En este último caso se indexarán sólo los primeros 10 caracteres de Campo2.

Índices únicos

Aquí los valores duplicados no son permitidos. Hay varias formas de crear esta clase de índices –muy similares a las expuestas anteriormente– y son las siguientes:

```
CREATE UNIQUE INDEX indice1 ON tabla (Campo2);
```

```
CREATE TABLE tabla (
    Campo1 INT NOT NULL,
    Campo2 CHAR(30) NOT NULL,
    PRIMARY KEY (Campo1),
    UNIQUE indice1 (Campo2)
);
```

```
ALTER TABLE tabla ADD UNIQUE indice1 (Campo1);
```

Como en las anteriores exposiciones, un índice puede abarcar más de una columna –índices compuestos, hay que separarlas con comas (,). Existen dos maneras de eliminar un índice. Usar una u otra dependerá del tipo de índice que queramos eliminar. A continuación veamos el primer caso que nos servirá para eliminar índices de tipo clave primaria, y cuya sintaxis es la siguiente:

```
ALTER TABLE tabla DROP PRIMARY KEY;
```


Notemos que no indicamos el nombre de ningún campo: esto es porque sólo puede haber una clave primaria por tabla y con dar el nombre de ésta es suficiente. Para eliminar el resto de los índices, la sintaxis para usar es la siguiente:

ALTER TABLE tabla DROP INDEX nombre_indice;

Donde **nombre_indice** representa el campo sobre el cual se implementó el índice. En lugar de la instrucción anterior también podemos utilizar la siguiente:

DROP INDEX nombre_indice ON tabla;

Tengamos en cuenta que en las últimas instrucciones nunca eliminamos el campo, sino la restricción que tiene efecto sobre él: el campo seguirá existiendo pero ya no tendrá las características de un índice.

Debemos saber que para ver los índices de una tabla podemos utilizar cualquiera de las instrucciones que detallamos a continuación:

SHOW KEYS FROM tabla;

o bien,

DESCRIBE tabla;

Hay otro comando que utilizado de determinada manera se convierte en un sinónimo de **DESCRIBE**. Este comando se llama **EXPLAIN** y su sintaxis es la siguiente:

EXPLAIN tabla;

Las tres últimas instrucciones devolverán la estructura de la tabla y algunas características más; entre ellas, algunas referidas a los índices implementados.

PRUEBAS

Es muy importante recordar que, aunque no tengamos el lenguaje PHP en su versión 5 o superior, siempre podremos utilizar y probar las características provistas por las últimas versiones de MySQL desde su consola de DOS y, también, desde su terminal de Linux. Para obtener información adicional sobre el tema, podemos consultar el Capítulo 3 de este libro.

Cuando creamos un índice sobre una columna, MySQL crea internamente una estructura de datos aparte, más accesible y compacta que la tabla original. Cuando MySQL encuentre que hay un índice en una columna, usará esta estructura en vez de hacer un escaneo completo de la tabla. El uso de índices no es indispensable para que nuestras instrucciones SQL funcionen ni para que nuestra base de datos trabaje normalmente, los índices influyen más que nada en el desempeño, en la velocidad de respuesta de la base de datos.

En MySQL el número máximo de índices por tabla varía según el tipo utilizado, pero todos ellos admiten un mínimo de 16. En caso de no utilizar índices, MySQL recorrerá secuencialmente la tabla en cuestión desde el primer registro hasta el último, lo que produce eventualmente inconvenientes con respecto a la velocidad de resolución de instrucciones SQL, y una sobrecarga de los recursos del sistema al tener éste un mayor trabajo –tanto de entrada/salida como de utilización de CPU y memoria. Aún en tablas no tan grandes, el uso de índices mejora la velocidad de acción de una base de datos.

Y después de esto nos surge una pregunta, ¿por qué no indexar todas las columnas de todas las tablas? La respuesta es que cada vez que nosotros implementamos un **INSERT**, **UPDATE**, **REPLACE**, o **DELETE** sobre una tabla, MySQL tiene que actualizar el índice correspondiente para reflejar los cambios en los datos, lo que evidentemente redunda en más trabajo para el servidor.

No hay un indicio especial y concreto que nos permita saber cuándo crear un índice o cuándo no –de todas maneras, siempre podemos trabajar sobre la base de prueba y error–, pero sí hay algunos aspectos que debemos tener en cuenta: qué tipo de consultas ejecutamos y con qué frecuencia lo hacemos.

Es importante destacar que lo del tipo de consulta tiene que ver con que indexar una columna en particular podría ser de utilidad si dicha columna aparece en la cláusula denominada **WHERE** en una consulta.

Y lo de la frecuencia es porque hay que determinar si se justifica crear un índice ante una determinada reiteración de uso: puede ocurrir que la columna indexada aparezca en las consultas pero no en la cantidad de veces necesarias para superar el gasto de mantenimiento que precisa el índice.



ANÁLISIS DE SISTEMAS

Diseñar un sistema requiere, en rigor, un análisis previo, que consiste en una serie de pasos, como recopilar información acerca de la empresa en la cual se lo va a utilizar, conocer sus tareas y cómo se las lleva a cabo, cuántas personas trabajan allí, cuáles son las perspectivas, y demás. Todas estas actividades son propias de un Analista de Sistemas.

TIPOS DE RELACIONES ENTRE TABLAS

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

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 pueden relacionar con un número de instancias de otra entidad. Quizás esto suene confuso pero con lo siguiente observaremos que no lo es tanto.

Hay tres tipos de cardinalidad:

- Uno a Uno
- Uno a Varios
- Varios a Varios

Cardinalidad Uno a Uno

Se llama relación de uno a uno entre dos tablas, 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.

Ejemplo: podemos citar como ejemplo la siguiente relación, cada país tendrá sólo un presidente y cada presidente tendrá a su cargo sólo un país.

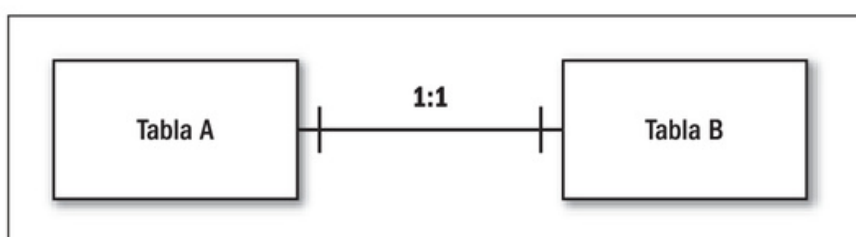


Figura 15. Relación uno a uno.

III IDEAR TABLAS

Es importante saber que la tarea de relacionar tablas es, finalmente, relacionar ideas. Debemos hacerlo como nos resulte más cómodo: sobre un papel, con palabras, con dibujos, con diagramas, etcétera. El punto fundamental es que intentemos llegar a tener un entendimiento profundo sobre el sistema que estamos diseñando, para, posteriormente, implementarlo con éxito.

Llegado el caso, existe la posibilidad de unificar las tablas A y B en una sola, pero esto dependerá de los siguiente aspectos:

- Las necesidades de nuestro sistema.
- De que 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 (posee una gran cantidad de campos) y, por lo tanto, difícil de manejar.

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.

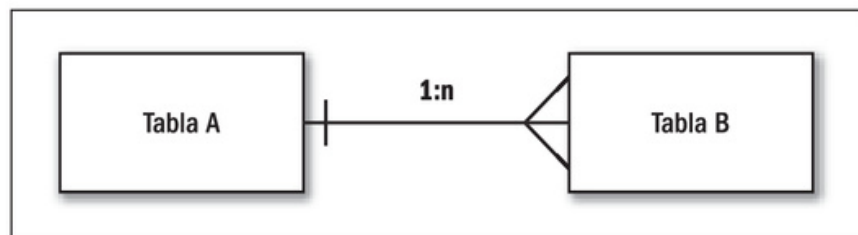


Figura 16. Relación uno a varios.

Ejemplo: cada ciudadano vota por un candidato y cada candidato tendrá la posibilidad de ser votado por más de un ciudadano.

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.

III RELACIONES CON UNO MISMO

Una tabla puede relacionarse consigo misma a través de sus campos. Esto se da comúnmente cuando se necesita implementar estructuras de directorio con niveles desconocidos. Obviamente, no se puede crear una tabla por nivel, y la solución está en centralizar todos los niveles en una sola. Los buscadores que permiten este tipo de búsquedas implementan esta forma de trabajar.

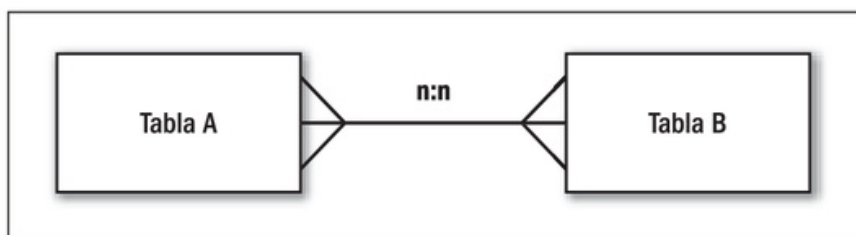


Figura 17. Relación varios a varios.

Ejemplo: un libro determinado pudo haber sido escrito por varios autores y a su vez cada autor pudo haber escrito varios libros.

Generalmente, 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**, creando una tercera tabla (llamémosla C) que hace de intermediaria entre la A y la B, que permanecen intactas.

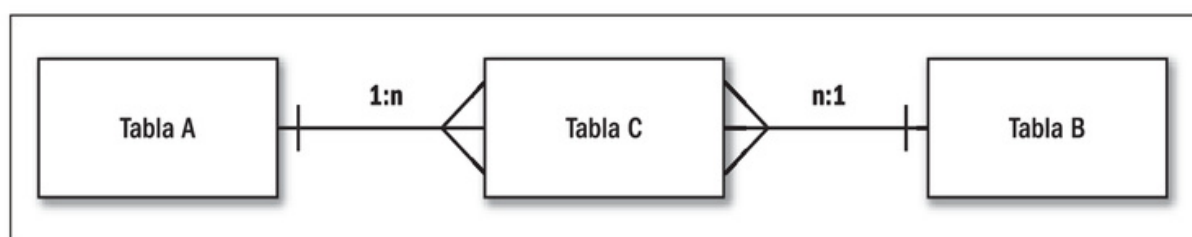


Figura 18. Transformación de relaciones varios a varios.

Veamos un ejemplo de esto último (seremos capaces de identificar con símbolo # el campo que se encarga de actuar como clave primaria):

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 19**.



Figura 19. Transformación de relaciones varios a varios.

La **modalidad** indica si es obligatorio o no que una instancia participe en la relación. Vimos con anterioridad 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 la posibilidad de definir si es obligatoria o no la participación de cada instancia en esa relación. Volviendo a los ejemplos que vimos anteriormente: 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 a esta clase de preguntas, estaremos en condiciones de definir la modalidad implementada en nuestras relaciones.

La forma gráfica de definir la modalidad en las distintas relaciones depende del autor; no hay sólo una manera, pero la forma clásica o la más usada es la siguiente:

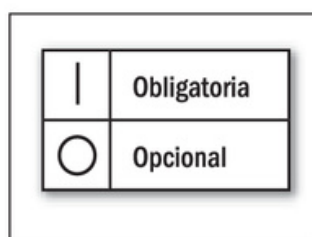


Figura 20. Simbología de modalidad en las relaciones.

Veamos algunos ejemplos:

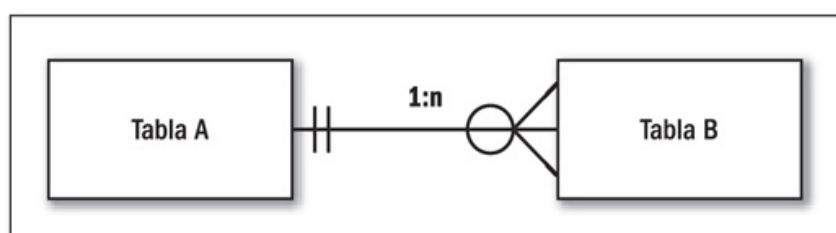


Figura 21. Ejemplo de modalidad en las relaciones.

En la figura anterior, 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.

NORMALIZACIÓN

Cuando normalizamos una base de datos lo que hacemos es aplicar una serie de reglas con el objetivo fundamental de evitar problemas futuros relacionados con la lógica de diseño implementada en ella.

Las llamadas reglas o formas normales pueden parecer confusas al principio y lo son: en este punto, poner en práctica los conocimientos adquiridos tiene una importancia fundamental, y al poco tiempo veremos que ni siquiera hará falta recordar de memoria estas reglas, ya que con sólo ver el diagrama de entidad-relación de la base de datos, notaremos de inmediato los errores cometidos.

Aplicar las reglas de normalización nos asegurará, en principio, dos cosas:

- Evitar la redundancia de datos.
- Evitar errores en lo que se refiere a la inserción, actualización y borrado de registros.

Al aplicar estas reglas, debemos tener en cuenta en todo momento lo que necesitamos de nuestro sistema; es decir, sus requerimientos.

Supongamos que tenemos un sistema de alquiler de DVDs, si el sistema admite que un usuario pueda alquilar más de uno a la vez, tendremos un diseño y si el sistema no lo admite, tendremos otro. En síntesis, el diseño depende de las exigencias que tengamos.

Existen varios niveles de normalización:

- Primera Forma Normal
- Segunda Forma Normal
- Tercera Forma Normal
- Forma Normal Boyce-Codd
- Cuarta Forma Normal
- Quinta Forma Normal o Forma Normal de Proyección-Unión
- Forma Normal de Proyección-Unión Fuerte
- Forma Normal de Proyección-Unión Extra Fuerte
- Forma Normal de Clave de Dominio

Cada nuevo nivel, evidentemente, acerca más y más a una base de datos a ser verdaderamente relacional, pero aquí sólo veremos las tres primeras formas normales, por ser suficientes para la gran mayoría de los casos y porque para comprender las demás haría falta tener un conocimiento profundo.

Datos redundantes

Uno de los principios fundamentales en lo que respecta al diseño de bases de datos es el de evitar la redundancia de datos. De hecho –como veremos en la próxima sección–, luego de aplicar las técnicas de normalización de nuestro sistema, esto debería darse en forma completamente natural.

Veamos un ejemplo para saber de qué hablamos cuando nos referimos a la redundancia de datos. Supongamos que tenemos una tabla con datos de nuestros clientes:

Tabla Cliente		
#	Código	Numérico
	Nombre y Apellido	Alfanumérico
	Fecha de nacimiento	Fecha
	Nacionalidad	Alfanumérico
	Numero_Cuenta_Corriente	Alfanumérico

Aquí el campo que tiene claras posibilidades de repetir datos es Nacionalidad (si tenemos más de un cliente con la misma nacionalidad, ya hay redundancia, pero no hay que esperar a que esto suceda: si existe la posibilidad ya es suficiente).

¿Por qué no pensar lo mismo de Nombre y Apellido o de Fecha de nacimiento? Porque tanto los nombres como los apellidos son infinitos (hay países en los que no hay restricciones al momento de elegir un nombre) y lo mismo ocurre con las fechas.

Incluso a la hora de ingresar los datos de un cliente, es más seguro que seleccionemos su nacionalidad desde una lista, y no que la ingresemos manualmente, ya que esto posibilitaría errores de tipeo o distintas formas de ingreso (Argentina, Argentino, Arg., Argent., etcétera) y, por ende, datos erróneos y difíciles de tratar en nuestra base. Si

III APLICAR LOS CONOCIMIENTOS ADQUIRIDOS

Debemos tener en cuenta que si bien cada sistema tiene sus particularidades, al ganar experiencia, notaremos que nos enfrentamos a situaciones en las cuales podremos aplicar soluciones implementadas con anterioridad. Esto no enseña que resulta fundamental prestar atención al "qué", y no tanto al "cómo" o al "por qué".

quisiéramos saber la cantidad de clientes por país que tienen depósitos en el banco, aunque construyamos correctamente la consulta SQL, posiblemente los datos recuperados no reflejen la realidad. Guardando las nacionalidades en una tabla podríamos listarlas luego. Si no lo hiciéramos, nuestra tabla quedaría de la siguiente forma:

Tabla Cliente				
1	Sergio Batista	12-11-1956	Argentina	56-34343434-3444
2	Guillermo Batista	01-03-2000	Méjico	99-23666788-8784
3	Violeta Sales	23-12-1900	Brasil	85-63434323-2323
4	Santiago Borrás	12-12-1890	Argentino	12-11222145-5666
5	Victor Gomez	04-11-1956	Arg.	45-45454566-6666

Para solucionar el problema anterior y otros derivados que surgen a partir de la redundancia de datos, modificamos la estructura de la tabla de la siguiente manera:

Tabla Cliente				
1	Sergio Batista	12-11-1956	1	56-34343434-3444
2	Guillermo Batista	01-03-2000	2	99-23666788-8784
3	Violeta Sales	23-12-1900	3	85-63434323-2323
4	Santiago Borrás	12-12-1890	1	12-11222145-5666
5	Victor Gomez	04-11-1956	1	45-45454566-6666

Tabla Pais	
1	Argentina
2	Méjico
3	Brasil

La idea madre de todo esto es que los índices numéricos ocupan menos espacio que las cadenas de caracteres: así, en nuestro ejemplo, escribimos la palabra “Argentina” sólo una vez, y el número 1 (código correspondiente a Argentina) varias veces en reemplazo de “Argentina”. Por eso, casi nunca se habla de redundancia de datos cuando los campos en cuestión son numéricos.

III EXPERIENCIAS

Debemos saber que las técnicas de normalización no son reglas caprichosas que se deben cumplir porque sí. La mayoría de ellas tiene una gran utilidad práctica y es el resultado de estudios teórico-prácticos muy complejos. Se recomienda desde aquí tratar de incorporarlas rápidamente y tenerlas presentes en cada proyecto.

Primera Forma Normal

Una tabla se encuentra en la primera forma normal si cada una de las columnas admite un único valor para cada registro. Veamos un ejemplo:

Codigo_Usuario	Nombre	Apellido	Numero de Recibo	Producto
1	Juan	Lopez	3333	33, 34, 35
2	Rodolfo	Gaspar	3334	12, 12, 33, 34
3	Victor	Zapata	3335	10

En la tabla anterior existe una serie de problemas. A través de instrucciones SQL no podríamos contestar, por ejemplo, cuántos productos compró Juan López o cuál es el producto más vendido de la empresa.

Estos inconvenientes se dan posiblemente porque no se respeta la primera forma normal: la columna Producto contiene más de un valor por registro.

Es importante mencionar que aplicando la primera forma normal, nuestra tabla original se divide en dos y queda de la siguiente manera:

Codigo_Usuario	Nombre	Apellido	Numero de Recibo
1	Juan	Lopez	3333
2	Rodolfo	Gaspar	3334
3	Victor	Zapata	3335

Numero de Recibo	Producto	Cantidad
3333	33	1
3333	34	1
3333	35	1
3334	12	2
3334	33	1
3334	34	1
3335	10	1

Ahora sí estamos en condiciones de contestar fácilmente cualquier pregunta, e incluso nuestra visión de la base de datos a través de las tablas se ha simplificado considerablemente, haciéndola más entendible y lógica.

Segunda Forma Normal

En esta forma normal, lo que se trata de lograr es que cada uno de los campos de una tabla dependa directamente de la clave primaria. Veamos.

Pensemos en el siguiente ejemplo que consiste en guardar datos de empleados (nombre, empresa en la que trabaja, dirección de la empresa):

Cod_empleado	Nombre_empleado	Nombre_empresa	Direccion_empresa
1	Lopez, Ariel	Benavidez S.A.	Pasco 122
2	Diaz, Rodrigo	Benavidez S.A.	Pasco 122
3	Cruz, Julio	Banco Mar	Alberti 200

El campo **Nombre_empleado** depende directamente de la clave **Cod_empleado**. En cambio, **Direccion_empresa** no depende de **Cod_empleado**, sino de la empresa. Ver este tipo de situaciones depende de la experiencia que vayamos logrando .

En definitiva la situación final podría ser la siguiente:

Cod_empleado	Nombre_empleado
1	Lopez, Ariel
2	Diaz, Rodrigo
3	Cruz, Julio

Cod_empresa	Nombre_empresa	Direccion_empresa
1	Benavidez S.A.	Pasco 122
2	Banco Mar	Alberti 200

Cod_empleado	Cod_empresa
1	1
2	1
3	2

O también:

Cod_empleado	Nombre_empleado	Cod_empresa
1	Lopez, Ariel	1
2	Diaz, Rodrigo	1
3	Cruz, Julio	2

Cod_empresa	Nombre_empresa	Direccion_empresa
1	Benavidez S.A.	Pasco 122
2	Banco Mar	Alberti 200

En el primer caso podríamos suponer que un empleado puede trabajar en más de una empresa.

Tercera Forma Normal

Una tabla está en tercera forma normal si sus campos dependen únicamente de la clave, es decir, no dependen unos de otros. Si repasamos el ejemplo anterior veremos que esto no sucede. Veamos otro ejemplo:

Ciente	Nombre	Apellido	DVD	Duracion
1	Juan	Lopez	Toro Salvaje	122
2	Rodolfo	Gaspar	Taxi Driver	121
3	Victor	Zapata	El aviador	132

Si prestamos atención, veremos que el campo **Duracion** depende más del campo DVD que de **Ciente**. Aplicando la tercera forma normal obtendríamos el siguiente escenario:

Ciente	Nombre	Apellido	Cod_DVD
1	Juan	Lopez	2
2	Rodolfo	Gaspar	5
3	Victor	Zapata	3

Cod_DVD	DVD	Duracion
2	Toro Salvaje	122
5	Taxi Driver	121
3	El aviador	132

MYSQL Y EL SOPORTE PARA EL MODELO RELACIONAL ACTUAL

MySQL es un gestor de bases de datos bastante nuevo, quizás por eso se hayan implementado recién en las últimas versiones conceptos fundamentales del **Modelo Relacional**. A continuación, un repaso éstos con ejemplos en MySQL.



RESPETAR LAS REGLAS

Las técnicas de normalización son fáciles de recordar luego de haberlas aplicado un par de veces. Pero si aún dudamos de si su aplicación es o no necesaria, podemos probar a diseñar un sistema e implementarlo sin tomar en cuenta estas técnicas: veremos que cada modificación o nueva implementación que queramos realizar se volverá difícil –o imposible– de poner en práctica.

Integridad referencial

La integridad referencial implica que cuando un registro haga referencia a otro, ese otro debe necesariamente existir. Supongamos el siguiente caso:

Tabla Autor		
Cod_autor	Nombre_autor	Cód_nacionalidad
1	Ernesto Sábato	1
2	Jorge Luis Borges	1
3	Mario Vargas Llosa	5
4	Jorge Amado	2
5	Octavio Paz	3
6	Carlos Fuentes	3

Tabla Nacionalidad	
Cod_nacionalidad	Desc_nacionalidad
1	Argentina
2	Brasil
3	Méjico
4	España
5	Perú
6	Chile

Si hiciéramos una consulta del tipo:

```
SELECT * FROM autor, nacionalidad WHERE autor.cod_nacionalidad =
nacionalidad. cod_nacionalidad;
```

Necesitaríamos fundamentalmente dos cosas:

- Que **autor.cod_nacionalidad** no admitiera valores nulos.
- Que cada valor de la columna **autor.cod_nacionalidad** estuviera correspondido en **nacionalidad. cod_nacionalidad**.

Lo primero lo logramos al momento de crear la tabla, con el atributo **NOT NULL** sobre **autor.cod_nacionalidad**; y lo segundo, imponiendo integridad referencial.

¿Cómo se hacía antes de la implementación de la integridad referencial en las tablas tipo **INNOdb**, vistas más adelante? Se le dejaba el control absoluto a la aplicación y ninguno a la base de datos. Era el programador el que debía manejar las decisiones ante las posibles situaciones en las que esto sucediera. Hablamos en pasado, pero esto sigue siendo así hoy en día en muchas aplicaciones.

Claves foráneas

Una clave foránea es un campo común y corriente que tiene la particularidad de corresponderse con la clave primaria de otra tabla.

Estas claves están muy estrechamente ligadas al concepto de integridad referencial, debido a que si una clave foránea contiene un valor, ese valor se refiere a un registro existente en la tabla relacionada.

MySQL soporta claves foráneas únicamente en las tablas tipo **INNodb**. Si intentamos crear estas claves en otro tipo de tablas, no recibiremos errores ni advertencias, pero tampoco obtendremos ningún efecto: en otras palabras, las tablas se comportarán como si no hubiéramos creado las claves foráneas.

Cuando se crean tablas con claves foráneas, MySQL no crea de manera automática índices en las claves foráneas ni en las claves referenciadas. Si queremos crear estos índices debemos hacerlo explícitamente. Por ejemplo, si quisiéramos crear índices en la clave primaria y en la clave foránea, deberíamos hacer lo siguiente:

```
CREATE TABLE tabla2 (
    Campo2 INT NOT NULL,
    Campo4 INT NOT NULL,
    PRIMARY KEY(Campo2)
) ENGINE = INNODB;

CREATE TABLE tabla1 (
    Campo1 INT NOT NULL,
    Campo2 INT NOT NULL,
    Campo3 VARCHAR(30),
    PRIMARY KEY(Campo1),
    INDEX (Campo1),
    INDEX (Campo2),
    FOREIGN KEY (Campo2) REFERENCES tabla2(Campo2)
) ENGINE = INNODB;
```



SOLUCIONES

En ocasiones, al diseñar un sistema, veremos que hay distintas maneras de resolver un problema, y decidirse por una u otra dependerá de diversos factores. Esto nos permite notar que no existe un único camino o una sola respuesta para un problema: sólo la acumulación de experiencia hará que veamos cuál es la solución más conveniente en cada caso.

Volviendo al ejemplo de Autores y Nacionalidades, una forma de resolver el problema planteado sería la siguiente:

```
CREATE TABLE nacionalidad (  
    cod_nacionalidad INT,  
    desc_nacionalidad VARCHAR(50),  
    PRIMARY KEY(cod_nacionalidad)  
) ENGINE = INNODB;  
  
CREATE TABLE autor (  
    cod_autor INT,  
    nombre_autor VARCHAR(50),  
    cod_nacionalidad INT NOT NULL DEFAULT '0',  
    PRIMARY KEY(cod_autor),  
    INDEX (cod_autor),  
    INDEX (cod_nacionalidad),  
    FOREIGN KEY (cod_nacionalidad) REFERENCES nacionalidad(cod_nacionalidad)  
) ENGINE = INNODB;
```

Debemos saber que también es posible modificar una tabla existente agregándole una clave foránea, de la siguiente manera:

```
CREATE TABLE autor (  
    cod_autor INT,  
    nombre_autor VARCHAR(50),  
    cod_nacionalidad INT NOT NULL,  
    PRIMARY KEY(cod_autor)  
) ENGINE = INNODB;  
ALTER TABLE autor ADD FOREIGN KEY (cod_nacionalidad) REFERENCES nacionalidad  
(cod_nacionalidad);
```



PHP, SUS MEJORAS

Es muy interesante saber que el lenguaje PHP se encarga de incorporar muchas mejoras en su versión 5 además de las relacionadas con MySQL. Podemos obtener un listado de ellas en el **Capítulo 1** de este libro. Además, allí podremos observar el desarrollo a través de los años de las diversas versiones del lenguaje PHP que se han puesto a disposición de los usuarios.

No es recomendable aplicar sentencias **ALTER TABLE** o **CREATE INDEX** sobre tablas que están siendo referenciadas, por posibles problemas internos de MySQL. Lo que conviene hacer aquí es eliminar la tabla en cuestión y volverla a crear actualizada. ¿Qué ocurre cuando se borra algún registro que tiene, a su vez, registros asociados (en el caso de nuestro ejemplo la pregunta sería qué pasa si eliminamos algún registro de la tabla Nacionalidad y esa nacionalidad aparece en la tabla Autor)? Lo que pasa podemos manejarlo nosotros a través de las siguientes opciones:

- **ON DELETE RESTRICT:** es la acción por defecto, y directamente no permite la eliminación. Muestra un mensaje de error.
- **ON DELETE NO ACTION:** hace exactamente lo mismo que la opción anterior.
- **ON DELETE SET DEFAULT:** todavía no está implementada, pero lo que debería hacerse es reemplazar las claves eliminadas por el valor que se dio por defecto (**DEFAULT**) al momento de crear la tabla. En el ejemplo de autores y nacionalidades reemplazaría el valor eliminado por **0**.
- **ON DELETE CASCADE:** elimina también las filas de la tabla hija. En nuestro ejemplo, si eliminamos el registro **Méjico** de nacionalidad, también borraríamos a todos los escritores mejicanos.
- **ON DELETE SET NULL:** se encarga de poner a **NULL** los campos de la tabla hija que perdieron a su padre. En nuestro ejemplo, si eliminamos el registro **Méjico** de nacionalidad, pondríamos el campo denominado **cód_nacionalidad** igual a **NULL** para todos los escritores mejicanos.

Para implementar alguna de estas opciones hacemos lo siguiente:

```
ALTER TABLE tablaPadre ADD FOREIGN KEY(nombre_campo)
REFERENCES tablaHija(nombre_campo) ON DELETE CASCADE;
```



SUBCONSULTAS

El uso de subconsultas cuando trabajamos desde un lenguaje de programación no es realmente imprescindible, si bien es cierto que puede alivianarnos las tareas en ciertas ocasiones. La forma clásica de suplir las subconsultas es realizar una consulta, guardar los datos en variables temporales y utilizarlos como argumentos para otras consultas.

Al trabajar con referencias entre tablas, debe crearse primero la tabla referenciada y luego la que hace referencia a ella (esto vale también para cuando cargamos la base desde un archivo **.SQL**). En nuestro ejemplo, como la tabla **Autor** hace referencia a la tabla **Nacionalidad**, debe crearse la tabla Nacionalidad antes que la tabla Autor.

Si no queremos tener complicaciones con el orden de creación, podemos crear todas las tablas sin especificar cuáles claves son foráneas (como en el **CREATE TABLE** del último ejemplo) y al final ejecutar los **ALTER TABLE** necesarios para especificar las claves foráneas correspondientes.

Cuando queramos borrar una tabla, debemos asegurarnos de que no esté referenciada por ninguna otra (esto vale también para cuando cargamos la base desde un archivo **.SQL**). En nuestro ejemplo, como la tabla **Autor** hace referencia a la tabla Nacionalidad, debemos eliminar la primera antes que la segunda.

Cuando se crea una clave foránea, MySQL le asigna internamente un identificador. Para eliminar una restricción (sólo la restricción, no el campo) de clave foránea necesitamos conocer esa clave. Para conocer esa clave podemos ejecutar la instrucción:

```
SHOW CREATE TABLE nombre_tabla;
```

Que nos devolverá algo como lo que sigue:

```
CREATE TABLE `autor` (  
  `cod_autor` int(11) NOT NULL default '0',  
  `nombre_autor` varchar(50) default NULL,  
  `cod_nacionalidad` int(11) NOT NULL default '0',  
  PRIMARY KEY (`cod_autor`),  
  KEY `cod_autor` (`cod_autor`),  
  KEY `cod_nacionalidad` (`cod_nacionalidad`),  
  CONSTRAINT `autor_ibfk_1` FOREIGN KEY (`cod_nacionalidad`)  
    REFERENCES `nacionalidad` (`cod_nacionalidad`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

La clave que necesitamos está entre comillas e inmediatamente después de la palabra **CONSTRAINT**. Una vez que tomamos conocimiento de esto, debemos ejecutar la instrucción para eliminar la restricción:

```
ALTER TABLE autor DROP FOREIGN KEY autor_ibfk_1;
```

La clave varía según el caso.

En cuanto a actualizar registros con claves foráneas, tenemos cuatro opciones que están disponibles a partir de la versión 4.0.8 y tienen la misma funcionalidad que las ya mencionadas en **ON DELETE**.

- **ON UPDATE CASCADE**
- **ON UPDATE NO ACTION**
- **ON UPDATE SET NULL**
- **ON UPDATE RESTRICT**

Si vamos a usar al mismo tiempo las opciones denominadas **ON DELETE** y **ON UPDATE**, será necesario que prestemos atención al orden que utilizamos para escribirlas: en primer lugar ingresamos **ON DELETE** y posteriormente **ON UPDATE**.

Subconsultas

Las subconsultas se implementaron en MySQL a partir de la versión 4.1.

```
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.
```

... RESUMEN

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



TEST DE AUTOEVALUACIÓN

- 1 ¿Cuáles son las ventajas de trabajar con bases de datos?

- 2 ¿Qué es una tabla? Dé tres ejemplos.

- 3 ¿Qué es un 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 las elegirá?

- 5 Brevemente, ¿qué es un índice?

- 6 ¿Cuál es una de las principales diferencias entre el modelo de red y el modelo jerárquico?

- 7 ¿Qué entiende por *Integridad Referencial*?

- 8 ¿Qué entiende por *Clave Foránea*?

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 una base de datos en donde intervengan relaciones uno a varios e implemente un esquema de integridad referencial como se hizo en este capítulo.

MySQL

Ya vimos los distintos modelos de bases de datos e hicimos hincapié en el modelo relacional. Para poner en práctica los conceptos teóricos vistos, elegimos una base de datos muy popular en este momento y con mucho futuro por delante: MySQL. Analizaremos en este capítulo cuáles son las razones de esta sentencia y por qué MySQL se ha convertido en el complemento ideal de PHP.

¿Por qué MySQL?	70
Obtención de MySQL	71
Licencia de uso	71
Diferencias entre versiones	73
Tipos de datos	73
Cadenas de caracteres	73
Númericos	76
Fecha y hora	78
Tipos de tablas	80
ISAM	81
MyISAM	82
MERGE	83
HEAP	85
InnoDB	85
BerkeleyDB	86
¿En qué casos usar cada una?	86
Referencia de funciones	87
Funciones para trabajar con cadenas de caracteres	87
Funciones para trabajar con campos numéricos	98
Funciones para trabajar con fecha y hora	104
Funciones de conversión	113
Funciones agregadas o estadísticas	114
Operadores de comparación	114
Operadores lógicos	115
Introducción a los procesos almacenados	117
El uso del monitor de MySQL	123
Resumen	127
Actividades	128

¿POR QUÉ MYSQL?

MySQL es un sistema gestor de bases de datos muy utilizado en la actualidad por, entre otros, 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).

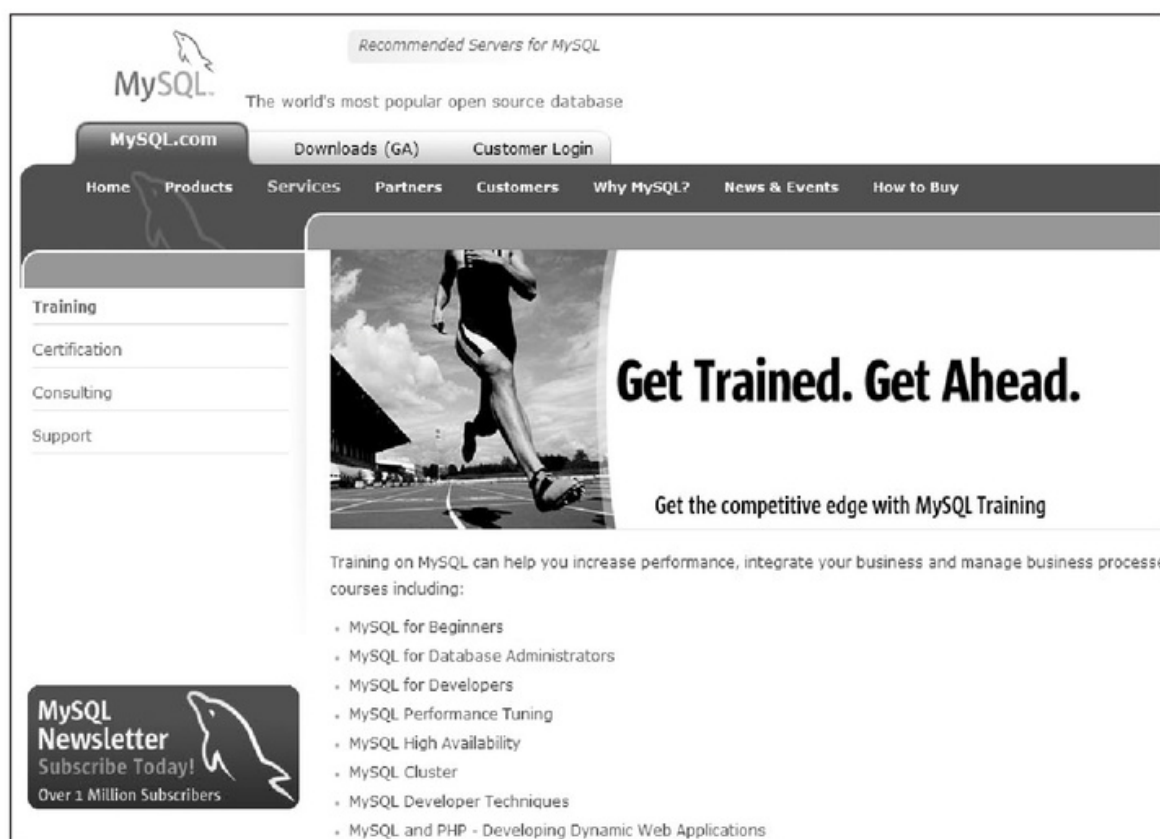


Figura 1. Sitio web de MySQL.



ANTES DE LA CREACIÓN

Aunque es posible modificar el tipo de dato de una columna de una tabla, no es recomendable cambiar la base de datos durante un desarrollo, a menos que sea absolutamente necesario. Por este motivo, es de suma importancia estudiar los valores máximos y mínimos que vaya a tomar cada campo antes de crear las estructuras correspondientes.

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 mayormente en proyectos relacionados con sitios web.

Obtención de MySQL

MySQL puede obtenerse de forma completamente libre a través de cualquier tipo de distribución: revistas, Internet, copias en CD provistas por amigos, etcétera. Pero sin dudas, una de las formas más utilizadas es acceder al sitio en Internet de **MySQL** (www.mysql.com), y desde allí descargar la versión más adecuada con respecto al sistema sobre el cual desarrollaremos nuestras aplicaciones. Ésta 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

El hecho de que MySQL este disponible para tantos sistemas operativos ha sido uno de los puntos decisivos para lograr la popularidad actual.

Licencia de uso

MySQL pone a disposición de los usuarios dos tipos de licenciamiento:

- Una licencia comercial.
- Una licencia **GPL** (*General Public License*).

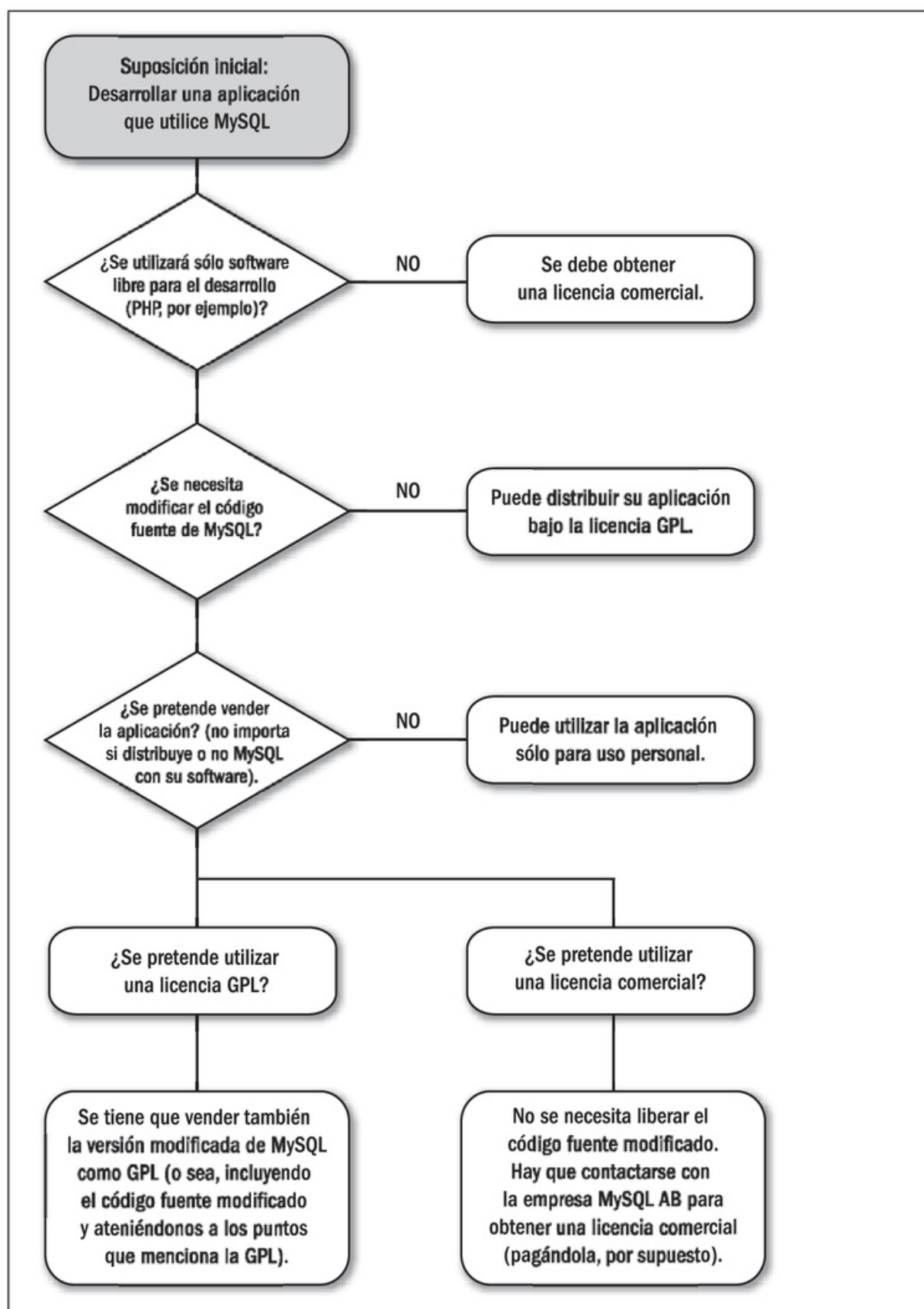


Figura 2. ¿En qué casos usar una licencia u otra? Sólo se puede responder a esta pregunta teniendo claro nuestro objetivo.

La licencia comercial brinda, entre otras cosas, soporte técnico y garantía.

Diferencias entre versiones

Una de las constantes mejoras que MySQL va ofreciendo a lo largo del lanzamiento de sus nuevas versiones es el soporte para el estándar del lenguaje de consulta SQL (**Structured Query Language**) y, por ende, el esquema relacional.

La versión que le dio más satisfacciones al equipo de MySQL fue la **3.23**, ya que con ella alcanzó estabilidad y un desempeño que hizo que se la tuviera en cuenta entre las principales opciones al elegir un gestor de base de datos relacionales (entre otras mejoras, la inclusión de tres tipos de tablas: **MyISAM**, **InnoDB** y **BerkeleyDB**).

Desde la versión 4.0 se incluyen las tablas **InnoDB** en la distribución estándar, lo que implica la posibilidad de relacionar tablas y proveerles de mayor seguridad y rapidez. Desde la versión 4.1, MySQL soporta subconsultas y consultas preparadas.

Para la versión 5, entre otras mejoras se destacan la inclusión de soporte para programar procedimientos almacenados.

Para saber qué versión de MySQL tenemos instalada en nuestro sistema, podemos realizar la consulta que vemos a continuación:

```
SELECT VERSION();
```

Para obtener más información acerca de cómo realizar consultas a MySQL, ver en este mismo capítulo la sección **El uso del monitor de MySQL**.

TIPOS DE DATOS

Se utilizan para definir los tipos de datos de las columnas de una tabla al crearla. MySQL soporta una gran variedad de datos, uno para cada necesidad.

Cadenas de caracteres

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

III ACERCA DEL SOFTWARE LIBRE

Es muy importante tener en cuenta que podemos obtener más información sobre la licencia GPL y conceptos generales acerca del software libre si visitamos los sitios web que se encuentran en las direcciones www.gnu.org/licenses/gpl-faq.html y <http://es.gnu.org/licencias>. En ellos podremos consultar datos interesantes e información útil.

CHAR y VARCHAR

Son muy similares, y quizás la diferencia más notable sea la forma 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**, en este caso no se rellenará con espacios. Cuando obtenemos información a través de una consulta SQL, no obtenemos 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. Para más información, ver función **BINARY**.

BLOB y TEXT

Se usan 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. Otra diferencia podría ser su uso. **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, hasta imágenes, archivos de sonido o video. **BLOB** (es un acrónimo de *Binary Large Object*, objeto binario de gran tamaño) se subdivide en cuatro tipos que difieren sólo 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.

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.

SELECCIÓN DE TIPOS DE DATOS

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

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:

Definiendo columnas de tipo **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, sólo que aquí se puede seleccionar ninguno o más de un valor de la lista (hasta 64, los muestran separados por comas).

Al ser **ENUM** semejante a una lista, podemos realizar consultas tales como:

```
SELECT * FROM nom_tabla WHERE set_col LIKE '%value%';
```

```
SELECT * FROM nom_tabla WHERE FIND_IN_SET('value',set_col)>0;
Ver funcion FIND_IN_SET()
```

```
SELECT * FROM nom_tabla WHERE set_col = 'val1, val2';
Donde val1 y val2 son opciones de la lista.
```

```
SELECT * FROM nom_tabla WHERE set_col & 1;
Busca solo los que contengan el primer elemento de la lista.
```

III NÚMEROS Y CADENAS DE CARACTERES

Es una buena costumbre definir como numéricos sólo aquellos campos sobre los cuales se realizarán operaciones matemáticas, como la edad de una persona o el número de productos de una empresa. Pero en el caso de, por ejemplo, números de documento, esto no sería eventualmente necesario (recordemos que estas decisiones dependerán del sistema en cuestión).

Numéricos

Se definen los subtipos **DECIMAL** (o **NUMERIC**, o **DEC**), **INTEGER** (o **INT**), **TINYINT**, **BIT**, **BOOL**, **MEDIUMINT**, **BIGINT**, **SMALLINT**, **FLOAT** y **DOUBLE** (o **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, sólo 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**.

Veamos las características de cada tipo de dato:

TINYINT[(M)]

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 **M** dígitos.

- 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**.
- Sin signo (**UNSIGNED**) va desde **0** hasta **255**.

BIT

Es un **TINYINT** de un dígito (**TINYINT(1)**).

BOOL

Es un **TINYINT** de un dígito (**TINYINT(1)**).

SMALLINT[(M)]

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 **M** dígitos.

- 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**.
- Sin signo (**UNSIGNED**) va desde **0** hasta **65535**.

MEDIUMINT[(M)]

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 **M** dígitos.

- 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**.
- Sin signo (**UNSIGNED**) va desde **0** hasta **16777215**.

INT[(M)]

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 **M** dígitos.

- Si se omite o se sobrepasa la capacidad correspondiente a **INT**, se toma la cantidad máxima soportada por este tipo de dato.
- Si se define con signo va desde **-2147483648** a **2147483647**.
- Sin signo (**UNSIGNED**) va desde **0** hasta **4294967295**.

INTEGER[(M)]

Sinónimo de **INT**.

BIGINT[(M)]

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 **M** dígitos.

- Si se omite o se sobrepasa la capacidad correspondiente a **BIGINT**, se toma la cantidad máxima soportada por este tipo de dato.
- Si se define con signo va desde **-9223372036854775808** a **9223372036854775807**.
- 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**.

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 **M** dígitos.

**COMPATIBILIDAD**

Una de las características que hacen de MySQL uno de los sistemas gestores de bases de datos más utilizados en la actualidad es su total compatibilidad y funcionamiento en las plataformas Linux y Windows, aunque hay que tener en cuenta que no se recomienda su uso en esta última para sitios que se encuentran en etapa de producción.

El rango de posibles valores va de **-3.402823466E+38** a **-1.175494351E-38** , **0**, y desde **1.175494351E-38** hasta **3.402823466E+38**.

DOUBLE[(M,D)]

Sirven para definir números con coma, con más precisión que **FLOAT**.

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 **M** dígitos.

El rango de posibles valores va:

de **-1.7976931348623157E+308** a **-2.2250738585072014E-308**, **0**,

y de **2.2250738585072014E-308** a **1.7976931348623157E+308**.

DOUBLE PRECISION[(M,D)]

Sinónimo de **DOUBLE**.

REAL[(M,D)]

Sinónimo de **DOUBLE**.

DECIMAL[(M[,D])]

Debemos saber que 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** a **9999999999** para números con signo.

Y por defecto, **D** es igual a **0**.

DEC[(M[,D])]

Sinónimo de **DECIMAL**.

NUMERIC[(M[,D])]

Sinónimo de **DECIMAL**.

Fecha y hora

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

DATETIME

DATETIME: Se utiliza cuando se necesita trabajar con fechas y horarios a la vez. El formato por defecto es '**YYYY-MM-DD HH:MM:SS**' pero se le puede dar uno diferente si por algún motivo necesitáramos hacerlo, por ejemplo: '**YYYY/MM/DD HH%MM%SS**'.

El rango va desde:

'1000-01-01 00:00:00' hasta **'9999-12-31 23:59:59'**.

Si al momento de ingresar un **DATETIME** definimos un valor inválido (por ejemplo, minuto superior a **60**) se almacenará la fecha nula. Por ejemplo: para el formato **'YYYY-MM-DD HH:MM:SS'** sería **'0000-00-00 00:00:00'**.

DATE

DATE se utiliza cuando se necesita trabajar sólo con fechas. El formato por defecto es **'YYYY-MM-DD'** pero se le puede dar un formato diferente si por algún motivo necesitáramos hacerlo, por ejemplo: **'YYYY*MM*DD'**. Rango **'1000-01-01'** a **'9999-12-31'**. Si al momento de ingresar un **DATE** definimos un valor inválido (por ejemplo, mes superior a **12**) se almacenará la fecha nula. Por ejemplo: 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.

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

Tabla 2. Formatos de la función **TIMESTAMP**.

Si al 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 al momento de ingresar un **TIMESTAMP** 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 no ingresados como ceros: esto supone un problema para las fechas, ya que si tenemos un **TIMESTAMP(2)** no podemos ingresar sólo 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 **TIMESTAMP**.

TIME

Debemos saber que **TIME** se utiliza cuando se necesita trabajar sólo 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 un formato diferente si por algún motivo necesitáramos hacerlo, por ejemplo: '**HH*MM*SS**'.

El rango va de '**-838:59:59**' a '**838:59:59**' (es importante que el hecho de poder almacenar **TIME** negativos nos da la pauta de que existen más usos que el de simplemente guardar el horario de un determinado suceso).

- Si al momento de ingresar un **TIME** definimos un valor inválido (por ejemplo, minuto superior a **60**) se almacenará la fecha nula. Por ejemplo: para el formato '**HH:MM:DD**' sería '**00:00:00**'.
- Si ingresamos valores fuera de rango, éstos se terminan reemplazando 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 sólo 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 ilegal, éste será convertido a **0000**.

TIPOS DE TABLAS

Cuando se trabaja con MySQL, existe la opción de variar el tipo de tabla después de creada (salvo con 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) UNSIGNED,  
    campo2 VARCHAR(25) NOT NULL  
) ENGINE=MYISAM;
```

Si se omite la opción **ENGINE=...** por defecto se crea una tabla MyISAM.

Importante: según MySQL, la opción **TYPE** (similar a **ENGINE**) es soportada hasta la versión 4.x de MySQL (dejará de usarse a partir de la 5). La opción

ENGINE fue añadida en la versión 4.0.18 del lenguaje (para las series 4.x) y en la versión 4.1.2 (correspondiente a las versiones 4.1).

Si se intenta crear un tipo de tabla no disponible en nuestra versión, MySQL optará por crear una tabla tipo MyISAM.

A continuación, haremos una reseña de los tipos de tablas soportados.

ISAM

En un principio, el gestor de bases de datos MySQL empezó utilizando este tipo de tablas y, actualmente, se las considera en desuso.

Entre sus desventajas figura el hecho de no poder transportar archivos entre máquinas con distinta arquitectura (tiene un formato diferente para cada arquitectura / sistema operativo, lo cual resulta más rápido, pero presenta el problema de la incompatibilidad) y el de no manejar archivos de tablas superiores a 4 GB.

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 basada sobre las tablas ISAM, por supuesto que ofreciendo más opciones que éstas.

Cuando se usan estas tablas, los datos se almacenan físicamente en dos archivos: uno que tiene la extensión **.MYI (MYISAM INDEX)**, en donde se almacenan los índices, y otro **.MYD (MYISAM DATA)**, donde se almacenan los datos.

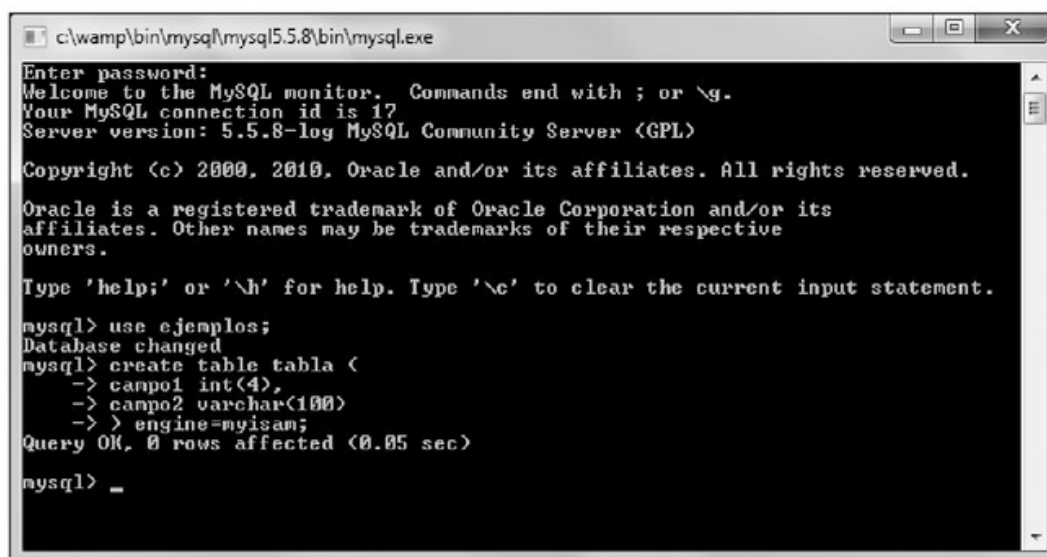


VERSIONES

Es recomendable utilizar la última versión de MySQL, pero en ocasiones no debemos hacerlo, ya que las empresas que desarrollan software suelen liberar las llamadas versiones beta, no aconsejables para sistemas que estén en etapa de producción, pero sí para los que estén en desarrollo. Queda a criterio del lector utilizar las versiones beta o las estables.

Se encargan de proveer un almacenamiento independiente: esto significa que se pueden copiar tablas de una máquina a otra de distinta plataforma.

Si estamos trabajando con MySQL en forma local, normalmente, se podrá ver en el propio equipo que estos archivos se almacenan en una carpeta que tiene por nombre 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 3. Creando una tabla *MyISAM* desde el monitor de *MySQL*.

Además:

- Soportan archivos de gran tamaño (63 bits, archivos de tablas superiores a 4 GB) en comparación con los que soportaban las ISAM.
- Están optimizadas para sistemas operativos de 64 bits.
- Posibilidad de indexar campos **BLOB** y **TEXT**.
- Se permiten valores **NULL** en columnas indexadas.
- 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.

ORACLE

Hace algún tiempo la empresa Oracle tomó posesión de MySQL (la primera versión lanzada bajo la nueva administración fue la 5.5, y al momento de escribir este relanzamiento va por la 5.5.8). Esa compra generó un gran revuelo entre los usuarios de este popular servidor de bases de datos, que aguardan expectantes cuáles serán los pasos a seguir por parte de la compañía.

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 éste persiste, se vuelve a crear el archivo que contiene la tabla. Y si continúa existiendo, se intenta repararlo escribiendo los registros sin un ordenamiento.

- Inserts concurrentes (se pueden insertar varios registros al mismo tiempo).

MERGE

Este tipo de tabla es muy utilizada en los casos en que se precise tratar 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 es de gran tamaño, y acceder a su contenido llevará una cantidad considerable de tiempo y recursos. Obviamente, estamos hablando de una tabla realmente muy grande.

Otras características:

- Sólo 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 MyISAM, en un archivo **.MRG** (aquí hay en realidad un índice de los archivos **.MYI** usados. Ver tablas **MYISAM**).
- Permiten de alguna forma burlar el tamaño máximo de una tabla y el tamaño máximo de un archivo en un sistema operativo específico.
- Debemos saber que si queremos borrar una de las tablas MyISAM que forma parte de la tabla **MERGE**, no podremos hacerlo bajo el sistema operativo Windows, ya que éste 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 forma 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 no abiertas.

Cuando se crea una tabla de este tipo hay que especificar (con la instrucción **UNION**) la lista de tablas asociadas. Veamos un ejemplo.

```
CREATE TABLE t1 (a INT AUTO_INCREMENT PRIMARY KEY, mensaje CHAR(20));

CREATE TABLE t2 (a INT AUTO_INCREMENT PRIMARY KEY, mensaje CHAR(20));

INSERT INTO t1 (mensaje) VALUES ("uno");
INSERT INTO t1 (mensaje) VALUES ("dos");
INSERT INTO t1 (mensaje) VALUES ("tres");

INSERT INTO t2 (mensaje) VALUES ("cuatro");
INSERT INTO t2 (mensaje) VALUES ("cinco");
INSERT INTO t2 (mensaje) VALUES ("seis");

CREATE TABLE total (
    a INT AUTO_INCREMENT PRIMARY KEY,
    mensaje CHAR(20)
)
ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;

SELECT * FROM total;
```

Devolvería algo como:

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



DIFERENCIAS QUE ENRIQUECEN

Quizás la mayor diferencia que encontraremos al desarrollar aplicaciones utilizando un sistema gestor de bases de datos u otro estará dada por las funciones implementadas por cada uno de ellos: funciones propias no disponibles en otras bases, nombres diferentes, argumentos distintos en cuanto a la cantidad y el tipo de datos, etc.

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 hace diferentes del resto: son tablas en memoria, son temporales y desaparecen cuando el servidor se cierra.

Es importante saber que esto las hace realmente rápidas y que, a diferencia de una tabla **TEMPORARY**, que sólo puede ser accedida por el usuario que la crea, una tabla **HEAP** puede ser utilizada por diversas personas.

Algunas otras particularidades:

- No soportan columnas de tipo **BLOB** o **TEXT**.
- No soportan columnas de tipo **AUTO_INCREMENT**.
- No se permiten valores **NULL** en las columnas que han sido indexadas (antes de MySQL en la versión correspondiente a la 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

Debemos saber que estas tablas, al igual que las **BerkeleyDB**, son **TST**: este término significa *Transactions Safe Tables*, o tablas para transacciones seguras. Las tablas tipo TST son menos rápidas y ocupan más memoria, pero a cambio ofrecen mayor seguridad frente a fallos durante la consulta.

Además, las tablas InnoDB tienen las siguientes características:

- Proveen la posibilidad de transacciones seguras. ACID (**Atomicidad**; **Consistencia**; **Separación**, en inglés *Isolation* y **Durabilidad**).
 - **Atomicidad**. Consultas tratadas como una sola, de tal forma que sólo se ejecutan cuando todas ellas tienen éxito, en caso de que alguna falle no se ejecuta ninguna.
 - **Consistencia**. Sólo 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 exitosamente, los cambios son permanentes y no se podrá volver atrás.
 - Soporta operaciones **COMMIT** y **ROLLBACK** (beneficio propio de ser **TST**).
- Recuperación ante fallos.
- Soporta **FOREIGN KEY** (**Claves foráneas**). Primera vez que se da esto en MySQL.

- Bloqueo a nivel de fila.
- Permite realizar copias de seguridad mientras la base está funcionando.
- Gran 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 las borra una por una –lo que produce problemas relacionados a la velocidad. Hasta ahora puede truncar tablas.

Fueron agregadas en la versión 4.0 de MySQL.

BerkeleyDB

Estas tablas pueden ser usadas independientemente de MySQL: están desarrolladas por otra empresa (**Sleepycat**) y el gestor de bases de datos MySQL ofrece una interfaz para trabajar con ellas como una posibilidad más.

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

¿En qué casos usar cada una?

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 respecto de la seguridad, además de las transacciones.



MÁS SOBRE INNODB

Las tablas conocidas como **InnoDB** se vuelven cada vez más populares en el gestor de bases de datos MySQL y conviene estar al tanto de los avances que se producen en este campo. Para eso existe un sitio de consulta permanente que podemos visitar accediendo a **www.innodb.com**. Está en inglés y trata en detalle las características de estas tablas.

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 se necesita solucionar, y ver en cada caso qué conviene.

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 particulares 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 el 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.

En los ejemplos que se dan al referenciar estas funciones cabe destacar que los parámetros (o argumentos) son genéricos, es decir que se les da un nombre cualquiera para que resulte más sencillo entender que es lo que hace la función; pero la idea es que para sacarle partido a las funciones los parámetros pueden ser otras funciones o bien campos de la/s tabla/s referenciada/s en la instrucción SQL.

Además, los argumentos que se encuentran entre corchetes (**[arg]**) son opcionales y las funciones están ordenadas alfabéticamente.

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:

```
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 **str** es una cadena nula.

BIN(N)

Vea la función **CONV**.

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)
```

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, toma 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 **y** y así.

CHAR_LENGTH(str)

Ver función **CHARACTER_LENGTH**.



¿QUÉ HORA ES?

Cuando se trabaja con funciones para el manejo de fechas y horas, hay que 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 **NOW()**. Podemos obtener la fecha y hora actuales de nuestro lugar de residencia a través de otros lenguajes que trabajan del lado cliente (por ejemplo, **JavaScript**).

CHARACTER_LENGTH(str)

Devuelve la longitud de **str**.

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

Devuelve una concatenación (unión) de las cadenas pasadas como parámetros. Devuelve **NULL** si alguna cadena es nula. Es importante que un argumento numérico es convertido a cadena de caracteres.

Por ejemplo:

```
SELECT CONCAT('una ', 'sola', ' cadena', ' - ejercicio ', 1);
```

Devuelve 'una sola cadena - ejercicio 1'

La función denominada **CONCAT_WS** hace lo mismo sólo que el primer parámetro se encarga de actuar como separador. Por ejemplo:

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

Devuelve 'uno,dos,tres'

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

Ver función **CONCAT**.

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 y **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 ésta que enunciamos aquí.

- BIN(N)

Devuelve una representación binaria de **N**. Devuelve **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 **P** es una cadena de caracteres, la función 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 denominado **LONG**).

Devuelve **NULL** si **N** es nulo. Su sintaxis es más sencilla pero la funcionalidad puede obtenerse utilizando la función **CONV**.

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

Se encarga de devolver **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**. Devuelve **0** si no está en la lista o si la lista está vacía. Devuelve **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.

HEX(P)

Vea la función **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'
```

INSTR(str,subcadena)

Vea la función **POSITION**.

LCASE(str)

Devuelve la cadena denominada **str** en minúsculas. Esta función es idéntica a la función **LOWER**. Para pasar a mayúsculas ver **UCASE**.

LEFT(str, N)

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

LENGTH(str)

Vea función **CHARACTER_LENGTH**.

LIKE

Es realmente muy utilizado y trae muchos beneficios al permitirnos incorporar las llamadas expresiones regulares en nuestras consultas SQL.

Devuelve **1** si hubo coincidencias y **0** si no.

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

Tabla 3. 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. Puede usar la función **BYNARY** para forzar lo contrario.

Por ejemplo:

```
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). Esta función es similar a INSTR sólo que los argumentos están en orden inverso (primero **str** y luego **subcadena**).

Si a **LOCATE** se le da un tercer argumento (llamémosle **POS**) la función comenzará a buscar coincidencias a partir de la posición **POS**.



VISITAR LOS SITIOS WEB

Cabe recordar que siempre podremos recurrir tanto al sitio de PHP (www.php.net) como al de MySQL (www.mysql.com) para obtener datos e información de último momento acerca de estas dos herramientas de desarrollo. MySQL incluso suele incluir allí artículos y tutoriales sobre temas específicos como el diseño de bases de datos escritos por expertos.

En MySQL versión 3.23 estas funciones (**LOCATE** y **INSTR**) son sensibles a mayúsculas y minúsculas. Ver la función **POSITION**.

LOWER(str)

Vea la función **LCASE**.

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 se encarga de devolver 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). Por ejemplo:

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

Vea 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 utilizar esta opción hay que definir la tabla al 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** al 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**.

OCT(N)

Vea la función **CONV**.

OCTET_LENGTH(str)

Vea función **CHARACTER_LENGTH**.

POSITION(subcadena IN str)

Devuelve la posición de la primera ocurrencia de **subcadena** en **str** (0 si no hay coincidencias). A continuación vemos un ejemplo:

CERTIFICACIÓN

Desde hace relativamente poco tiempo, MySQL comenzó a ofrecer certificaciones a sus usuarios. Ésta es una muestra más del afianzamiento que está teniendo en el mercado informático y de la calidad buscada en los desarrolladores. Sin dudas, una oportunidad interesante para aquellos que quieran orientar su conocimiento a las bases de datos.


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

En MySQL versión 3.23 esta función es sensible a mayúsculas y minúsculas. Debemos saber que esta función tiene la misma funcionalidad que **LOCATE**, sólo cambia la forma de pasarle los argumentos.

QUOTE(str)

Reemplaza los caracteres que correspondan por sus respectivas secuencias de escape.

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

Agregada en MySQL versión 4.0.3.

REPEAT(str, N)

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

```
SELECT REPEAT('lapiz japones ', 2);
Devuelve 'lapiz japones lapiz japones'
```

REPLACE(str, BUSCAR, REEMPLAZAR)

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

```
SELECT REPLACE('la momia blanca', 'blanca', 'negra');
Devuelve 'la momia 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 se encarga de devolver los primeros **LEN** caracteres de **str**. Por ejemplo:

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

Para completar por la izquierda ver la función **LPAD**.

RTRIM(str)

Devuelve la cadena **str** sin los espacios a la derecha (si es que los tiene).

Por ejemplo:

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

Vea también **LTRIM**, **TRIM**.

SPACE(N)

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

STRCMP(str1, str2)

Devuelve **0** si **str1** y **str2** son iguales. Devuelve **-1** si **str1** es más chico (de acuerdo al ordenamiento correspondiente) que **str2**. Devuelve **1** si **str2** es más chico (de acuerdo al ordenamiento actual) que **str1**.

Por ejemplo:

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

III AUSTERIDAD EN SQL

Es importante saber que cuando realizamos una consulta a un servidor de bases de datos, éste 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 lo que necesitamos.

```
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 ésta.

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 ésta.

```
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.

Si **cadena_a_borrar** no se especifica, se asume por defecto que es un espacio.

Con **LEADING** se borra sólo las ocurrencias de **cadena_a_borrar** desde la izquierda.

Con **TRAILING** se borra sólo las ocurrencias de **cadena_a_borrar** desde la derecha.

```
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 'algoxyz');  
Devuelve 'algox'
```

UCASE(str)

Devuelve la cadena **str** en mayúsculas.

Esta función es idéntica a la función **UPPER**. Para pasar a minúsculas ver **LCASE**.

UPPER(str)

Vea la función **UCASE**.

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 CEILING(1.23);
```

Devuelve 2

```
SELECT CEILING(-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 entera.

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 versión 4.1.0.

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 siguiendo. Se detiene a la primera verdad (si **N** es menor **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**. 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 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 versión 4.0.3.

MOD(N,M)

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

Por ejemplo:

```
SELECT MOD(10, 2);
```

Devuelve 0

```
SELECT MOD(29,9);
```

Devuelve 2

```
SELECT 29 % 9;
```

Devuelve 2

Otra forma de usar la función, disponible desde la versión 4.1:

```
SELECT 29 MOD 9;
```

Devuelve 2

PI()

Devuelve el valor de **PI**.

POW(X,Y)

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

**ANIDAR FUNCIONES**

Cabe mencionar que la utilidad de las funciones provistas por el gestor de bases de datos MySQL es ya de por sí muy poderosa, pero se puede incrementar anidando las funciones para obtener el resultado buscado. Por ejemplo: **CONCAT(TRIM(UPPER(cadena)), " aquí texto")**. Incluso, es posible anidar funciones que trabajan con distintos tipos de datos.

POWER(X,Y)

Vea la función **POW**.

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:

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

El orden de los registros devueltos variará aleatoriamente cada vez que se ejecute la consulta. Sólo 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**).

```
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)

Debemos saber que devuelve **X** truncado a **D** decimales. Truncar no significa redondear, sino más bien 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?

Es importante saber que hay algunas funciones matemáticas que tal vez la mayoría de los usuarios no utilicen nunca, pero que son de gran utilidad para programar aplicaciones orientadas al cálculo de probabilidades y estadísticas de sucesos, al poder unir el hecho de manejar gran cantidad de datos con una alta precisión en los cálculos.

Funciones para trabajar con fecha y hora

Debemos saber que cuando en algunas funciones se representa el año con sólo 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)

Vea la función **SUBDATE**.

CURDATE()

Ver la función **CURRENT_DATE**.

CURRENT_DATE

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

CURRENT_TIME

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

CURRENT_TIMESTAMP

Vea la función **NOW**.

CURTIME()

Ver función **CURRENT_TIME**.

DATE_ADD(date,INTERVAL expr type)

Vea la función **SUBDATE**.

DATE_FORMAT(date,formato)

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

III DISPONIBILIDAD

Debemos tener presente que no todas las funciones provistas por MySQL funcionan en todas las versiones del servidor: a través de los nuevos lanzamientos se han ido incorporando nuevos complementos y, en caso de que no contemos con la última versión disponible, será importante verificar la disponibilidad de las funciones que vayamos a utilizar.

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. Luego 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 la semana 01. Toma como primer día de la semana el 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).
%l (i mayúscula)	Hora (01..12).
%l	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 4. Parámetros de *DATE_FORMAT*.

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()
```

DATE_SUB(date,INTERVAL expr type)

Vea la función SUBDATE.

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.



SOPORTE PARA FUNCIONES

Al momento de utilizar una u otra función, es conveniente que consideremos la versión del servidor de bases de datos MySQL en donde finalmente se alojará el sitio web: una función no soportada nos obligaría a replantear partes del código. Recordemos que se puede recurrir a la ayuda que incorpora MySQL para saber ambas cosas: la versión del servidor y los requerimientos de las funciones.

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)

Debemos saber que 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 el 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**.

```
SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s');
```

Devuelve '2005 6rd February 21:57:10'

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

HOURL(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()

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 sólo pueden ser de formato **YYMM** o **YYYYMM**). Por ejemplo:

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

QUARTER(date)

Divide el 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)

Se encarga de convertir los segundos al siguiente formato: **'HH:MM:SS'** o también **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**.

Los valores posibles de **type** son:

VALORES SOPORTADOS POR TYPE	FORMATO DE EXPR
SECOND	SECONDS
MINUTE	MINUTES
HOURL	HOURS
DAY	DAYS
MONTH	MONTHS
YEAR	YEARS
MINUTE_SECOND	"MINUTES:SECONDS"
HOURL_MINUTE	"HOURS:MINUTES"
DAY_HOURL	"DAYS HOURS"
YEAR_MONTH	"YEARS-MONTHS"
HOURL_SECOND	"HOURS:MINUTES:SECONDS"
DAY_MINUTE	"DAYS HOURS:MINUTES"
DAY_SECOND	"DAYS HOURS:MINUTES:SECONDS"

Tabla 5. Valores de **type**.

Por ejemplo,

Sumar un segundo a la fecha dada:

- Utilizando signos.

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

- Utilizando función.

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

Ambas devuelven **1999-10-31 22:49:50**.

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**.

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**.

Sumar una hora a la fecha dada:

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

Devuelve **1999-01-01 01:00:00**.

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.

SYSDATE()

Vea la función **NOW**.

TIME_FORMAT(time, formato)

Funciona de la misma forma que la función **DATE_FORMAT()**, sólo 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 la función **UNIX_TIMESTAMP** se 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])

Debemos saber que si se le da sólo el primer argumento, se encarga de devolver 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**. (corresponde a **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
```

{ } TIMESTAMP

PHP también provee funciones de fecha y hora para trabajar con el formato **TIMESTAMP** de **Unix**, que parece dificultoso al comienzo, pero que una vez entendido y acompañado de ciertas funciones, es realmente muy completo, útil y utilizado tanto en MySQL como en PHP y en Unix / Linux. Al decir "**de Unix**" se quiere indicar su origen, pero es aplicable en cualquier sistema operativo.

YEARWEEK(date [, inicio])

Devuelve el año y el número de semana de la fecha **date**.

Debemos saber que el segundo argumento es opcional y se encarga de funcionar de igual forma que el segundo argumento de la función **WEEK**.

Funciones de conversión

CAST(expresion AS tipo)

La función denominada **CAST** se utiliza para realizar la conversión un tipo de dato en otro. El argumento tipo puede corresponder a lo siguiente:

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

Por ejemplo: por algún motivo tenemos un campo tipo texto (definido como **VAR-CHAR(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.

Y usando CAST:

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



FALTAN SIETE PARA LAS SIETE

Es necesario tener presente que la función **NOW** devuelve la fecha y la hora actuales del servidor. Este detalle es importante dependiendo de para qué utilicemos esa fecha. Si quisiéramos obtener la fecha de la máquina del cliente, deberíamos utilizar una función provista por otro lenguaje; la opción clásica es JavaScript.

Obtendríamos la secuencia 1,2,3,4,5,6,7,8,9,10.

CAST es sinónimo de la función **CONVERT(expresion,tipo)**.

Las funciones **CAST()** y **CONVERT()** 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:

- **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*)
- **NULL** (*Nulo*)



OPERADORA I

Es importante mencionar que los operadores de comparación, normalmente, varían de un lenguaje a otro, por lo que debemos presta atención a estos y otros detalles de la sintaxis de PHP, que pueden traernos problemas al principio. En algunos lenguajes se utiliza $\rightarrow =$ en vez de $= \rightarrow$, en otros se comparan dos valores con el operador $=$ en vez del $==$, etcétera.

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 **<=>** (Tabla 6).

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
¿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 6. Lista de operadores.

Operadores lógicos

Veamos qué tipo de operadores lógicos existen y cómo trabajar con ellos:

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 el valor 1 si alguno de los operandos es verdadero, el valor **NULL** si alguno es **NULL** y el otro es 0; si no devuelve el valor 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 sólo 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 evalúa 1 XOR 1 (que devuelve 0), luego el resultado anterior y 1 (0 XOR 1)

INTRODUCCIÓN A LOS PROCESOS ALMACENADOS

Una de las grandes novedades de la versión 5 de MySQL es sin dudas la inclusión de soporte para procesos almacenados. A continuación veremos los fundamentos teóricos y este tema más algunos ejemplos básicos.

Si ya usamos bases de datos como Oracle, Interbase / Firebird, PostgreSQL, seguro escuchamos hablar de procedimientos almacenados. Sin embargo, en MySQL esto es toda una novedad y un paso enorme para que esta base de datos se convierta en un verdadero sistema gestor de bases de datos.

Ahora bien, ¿qué son en realidad los procedimientos almacenados? Luego de sumergirnos en este tema veremos que el nombre es plenamente identificatorio y casi explica lo que es un procedimiento almacenado.

OPERADORA II

Un caso típico es el de confundir el operador de comparación (==) con el de asignación (=). Para no cometer un error, se recomienda ubicar primero el valor y, luego, la variable en una comparación. Supongamos que nos equivocamos escribiendo lo siguiente: **if (10 = \$var) { ...** Esto daría error inmediatamente; en cambio lo siguiente no: **if (\$var = 10) { ...**

Los **procedimientos almacenados** son un conjunto de instrucciones SQL más una serie de estructuras de control que nos permiten dotar de cierta lógica al procedimiento. Estos procedimientos están guardados –almacenados– en el servidor y pueden ser accedidos a través de llamadas, como veremos más adelante.

Se puede decir que estos procedimientos tienen un lenguaje de programación asociado y entonces, surge una pregunta cuyas respuestas nos ayudarán a empezar a ver los casos de utilidad de estos procedimientos: ¿cuál es la novedad en tener un lenguaje de programación para interactuar con una base de datos? ¿Para qué queremos un lenguaje –por cierto limitado– teniendo a nuestra disposición otros mucho más avanzados y completos –entre ellos PHP– para acceder a una base de datos? Dijimos que estos procedimientos pueden programarse utilizando un lenguaje: ésta no es la ventaja, la ventaja consiste en que residen en el servidor, y esto implica que:

- Se garantiza que todas las aplicaciones (sea cual fuere el lenguaje en el que estén escritas o el sistema sobre el que funcionen) obtendrán los mismos resultados.
- Un derivado de la ventaja anterior: si por algún motivo se debe cambiar de lenguaje de programación, no importa ya que los procedimientos almacenados forman parte de la base de datos, y esto posibilita el mantenimiento de la lógica.
- En principio, utilizando una base de datos se centraliza la información, ahora se centraliza la forma de acceder a esa información.
- La mayor parte del trabajo se realiza en el servidor y no en el cliente. Esto último reduce el tráfico y libera cargas de trabajo en las aplicaciones cliente. Supongamos que deseamos realizar una tarea sin utilizar procedimientos almacenados. Normalmente necesitaríamos interactuar con la base de datos pidiéndole información desde nuestra aplicación y enviando a su vez otras, lo que provoca inevitablemente un tráfico en la red que puede atentar contra nuestros recursos.
Si en cambio utilizamos procedimientos almacenados, haríamos tal vez sólo una petición y el resto de las tareas se resolverían en el servidor. Nuestra aplicación sólo esperaría esta resolución para recibir los datos.
- Con respecto a la seguridad, se podría evitar que las aplicaciones clientes accedan directamente a las tablas: ahora los procedimientos almacenados harán las veces de intermediarios, y se podrán evitar comportamientos maliciosos.



TIEMPO COMPARTIDO

El hecho de trabajar con procedimientos almacenados no es excluyente: podemos trabajar como normalmente se hace con lenguajes de programación separados de la base de datos e implementar a la vez operaciones o tareas que utilicen procedimientos. Lo mismo vale para las funciones, vistas también en esta sección.

Este último punto es importante y denota algo que quizás se deba aclarar: tal vez muchos usuarios tengan acceso a una base de datos o a programar aplicaciones que accedan a bases de datos, pero los procesos almacenados son programados por pocas personas y son esas personas quienes deciden qué datos poner a disposición de los demás, de qué forma hacerlo y cuándo modificar ese comportamiento. Evidentemente, utilizando procedimientos almacenados se tiene un mayor control sobre lo que se quiere mostrar y de qué forma hacerlo.

Más adelante veremos que MySQL crea algunas tablas durante su instalación, tablas de sistema, por ejemplo, para administrar usuarios, y si instalamos MySQL versión 5 veremos una tabla que no habíamos visto antes: una tabla llamada **proc**.

Desde esta tabla es posible gestionar los privilegios de los usuarios con respecto a la creación tanto de procedimientos como de funciones (veremos qué es una función más adelante en esta misma sección).

Debemos saber que los privilegios para ejecutar o modificar procedimientos o funciones son dados automáticamente a su creador.

Para crear un procedimiento, MySQL nos ofrece la directiva **CREATE PROCEDURE**. Al crearlo éste es ligado o relacionado con la base de datos que se está usando, tal como cuando creamos una tabla, por ejemplo.

Para llamar a un procedimiento lo hacemos mediante la instrucción **CALL**. Desde un procedimiento podemos invocar a su vez a otros procedimientos o funciones.

Un procedimiento almacenado, al igual cualquiera de los procedimientos que podamos programar en nuestras aplicaciones utilizando cualquier lenguaje, tiene:

- Un nombre.
- Puede tener una lista de parámetros.
- Tiene un contenido (sección también llamada definición del procedimiento: aquí se especifica qué es lo que va a hacer y cómo).
- Ese contenido puede estar compuesto por instrucciones sql, estructuras de control, declaración de variables locales, control de errores, etcétera.

En resumen, la sintaxis de un procedimiento almacenado es la siguiente:

```
CREATE PROCEDURE nombre (parámetro)
    [características] definición
```

Puede haber más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya nada dentro). Los parámetros tienen la siguiente estructura:

modo nombre tipo

Donde:

- **modo:** es opcional y puede ser **IN** (el valor por defecto, son los parámetros que el procedimiento recibirá), **OUT** (son los parámetros que el procedimiento podrá modificar) o **INOUT** (mezcla de los dos anteriores).
- **nombre:** es el nombre del parámetro.
- **tipo:** es cualquier tipo de dato de los provistos por MySQL.
- Dentro de **características** es posible incluir comentarios o definir si el procedimiento obtendrá los mismos resultados ante entradas iguales, entre otras cosas.
- **definición:** es el cuerpo del procedimiento y está compuesto por el procedimiento en sí: aquí se define qué hace, cómo lo hace y bajo qué circunstancias lo hace.

Así como existen los procedimientos, también existen las funciones. Para crear una función, MySQL nos ofrece la directiva **CREATE FUNCTION**.

Como ya vimos, la diferencia entre una función y un procedimiento es que la función devuelve valores. Estos valores pueden ser utilizados como argumentos para instrucciones SQL, tal como lo hacemos normalmente con otras funciones como son, por ejemplo, **MAX()** o **COUNT()**. Utilizar la cláusula **RETURNS** es obligatorio al momento de definir una función y sirve para especificar el tipo de dato que será devuelto (sólo el tipo de dato, no el dato). Su sintaxis es:

```
CREATE FUNCTION nombre (parámetro)
RETURNS tipo
[características] definición
```

Puede haber más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya nada dentro). Los parámetros tienen la siguiente estructura:

nombre tipo

Donde:

- **nombre:** es el nombre del parámetro.
- **tipo:** es cualquier tipo de dato de los provistos por MySQL.
- Dentro de **características** es posible incluir comentarios o definir si la función devolverá los mismos resultados ante entradas iguales, entre otras cosas.
- **definición:** es el cuerpo del procedimiento y está compuesto por el procedimiento en sí: aquí se define qué hace, cómo lo hace y cuándo lo hace.

Para llamar a una función lo hacemos simplemente invocando su nombre, como se hace en muchos lenguajes de programación.

Desde una función podemos invocar a su vez a otras funciones o procedimientos. En la bibliografía asociada a estos temas quizás se encuentre que tanto a los procedimientos almacenados como a las funciones se los llama "rutinas". También puede realizar búsquedas por **stored procedures**, que es la traducción al inglés de *procedimientos almacenados*. Tal vez no sea fácil encontrar información sobre procedimientos almacenados orientados específicamente hacia MySQL, y esto es por una razón lógica: MySQL implementa esta característica desde hace relativamente poco tiempo, y todavía no transcurrió el lapso necesario para que existan usuarios experimentados y expertos que desarrollen la documentación. A no desanimarse, todavía existe la opción de buscar información sobre procedimientos almacenados implementados en otros gestores de bases de datos: la teoría en estos casos es muy similar y lo mismo sucede con la forma de implementarlos, es decir con la sintaxis que se utiliza.

```
mysql> delimiter //
```

```
mysql> CREATE PROCEDURE procedimiento (IN cod INT)
-> BEGIN
->   SELECT * FROM tabla WHERE cod_t = cod;
-> END
-> //
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> delimiter ;
```

```
mysql> CALL procedimiento(4);
```

En el código anterior lo primero que hacemos es fijar un delimitador. Al utilizar la línea de comandos de MySQL vimos que el delimitador por defecto es el punto y coma (;): en los procedimientos almacenados podemos definirlo nosotros.

III SERVICIOS DE ALOJAMIENTO WEB

Dependiendo de las características de las instrucciones SQL que vayamos a utilizar en nuestros proyectos, será de importancia verificar que versión está instalada en el hosting para garantizar el correcto funcionamiento de nuestros scripts: evaluar los requerimientos de nuestras aplicaciones es un punto importante en las distintas etapas de un desarrollo.

Lo interesante de esto es que podemos escribir el delimitador anterior; sin que el procedimiento termine. Más adelante, en este mismo código volveremos al delimitador clásico. Luego creamos el procedimiento con la sintaxis vista anteriormente y ubicamos el contenido entre las palabras reservadas **BEGIN** y **END**.

El procedimiento recibe un parámetro para luego trabajar con él, por eso ese parámetro es de tipo **IN**. Definimos el parámetro como **OUT** cuando en él se va a guardar la salida del procedimiento. Si el parámetro hubiera sido de entrada y salida a la vez, sería de tipo denominado **INOUT**.

El procedimiento termina y es llamado luego mediante la siguiente instrucción:

```
mysql> CALL procedimiento(4);
```

Otro ejemplo:

```
CREATE PROCEDURE procedimiento2 (IN a INTEGER)
BEGIN
  DECLARE variable CHAR(20);
  IF a > 10 THEN
    SET variable = 'mayor a 10';
  ELSE
    SET variable = 'menor o igual a 10';
  END IF;
  INSERT INTO tabla VALUES (variable);
END
```

- El procedimiento recibe un parámetro llamado **a** que es de tipo entero.
- Se declara una variable para uso interno que se llama **variable** y es de tipo char.
- Se implementa una estructura de control y si **a** es mayor a 10 se asigna a **variable** un valor. Si no lo es se le asigna otro.
- Se utiliza el valor final de **variable** en una instrucción SQL.

TIPEAR O NO TIPEAR

Trabajar utilizando la línea de comandos de MySQL es una experiencia que no deberíamos dejar pasar: si bien hay muchas aplicaciones que nos permiten acceder a las funciones provistas por MySQL de forma gráfica, se aprenden muchas cosas y se afianzan otras utilizando esta opción que no por ser la más antigua deja de ser útil y muy empleada hoy en día.

Recordemos que para implementar el ultimo ejemplo se deberán usar nuevos delimitadores, como se vio anteriormente.

Observemos ahora un ejemplo de funciones:

```
mysql> delimiter //
```

```
mysql> CREATE FUNCTION cuadrado (s SMALLINT) RETURNS SMALLINT
-> RETURN s*s;
-> //
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> delimiter ;
```

```
mysql> SELECT cuadrado(2);
```

Otras bases de datos como PostgreSQL implementan procedimientos almacenados y brindan la posibilidad de programarlos utilizando lenguajes como PHP.

En MySQL hay intenciones de implementar lo mismo y seguramente en las próximas versiones lo veremos, pero más importante que utilizar un lenguaje u otro es entender para qué podrían servirnos los procedimientos almacenados.

Es por eso mismo que hemos decidido incluir esta introducción en el libro: para que el lector empiece a acostumbrarse a utilizar este tipo de herramientas y pueda sacar ventaja de su uso. Más adelante podrá cambiar la implementación de los procedimientos almacenados pero no lo harán con sus principios fundamentales.

En definitiva hemos dado un recorrido por el nuevo mundo de la programación de procedimientos almacenados en MySQL. Es importante que se trata de un mundo que está en pleno desarrollo y que promete evolucionar.

EL USO DEL MONITOR DE MySQL

Típicamente 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 consigo 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, o simplemente MySQL.

Este programa nos servirá para dar los primeros pasos con la base de datos, conocer a fondo su sintaxis y familiarizarnos con los mensajes de error del sistema.

Para ponerlo en marcha (primero asegúrese 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 sistemas Windows), ir hasta el directorio donde instalamos MySQL y entrar en la carpeta **bin**. Ahora debemos tipear según el caso:

- Es importante saber que si no creamos ningún usuario lo más probable es que podamos entrar poniendo simplemente lo siguiente:

```
MySQL <enter>
```

- Si ya creamos un usuario tenemos que entrar poniendo,

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

Atención: 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 y esperando recibir nuestras instrucciones.

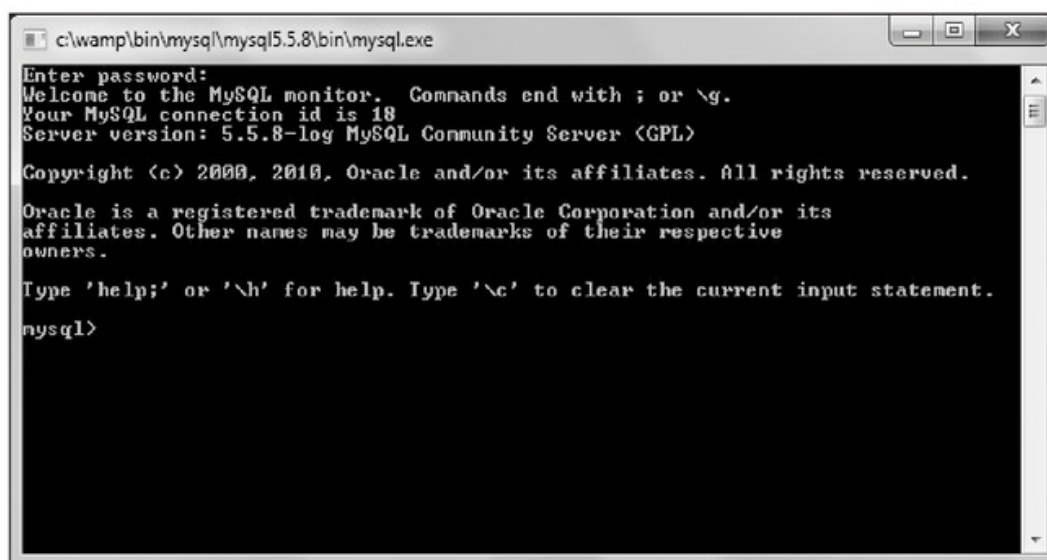


Figura 4. Pantalla de bienvenida del programa cliente MySQL.

Aquí lo que podemos hacer es ni más ni menos que ingresar sentencias SQL de cualquier tipo. Las sentencias terminan con ; (*punto y coma*), esto significa que aunque presionemos la tecla **<ENTER>** esto no producirá ningún cambio salvo el de introducir una nueva línea en la pantalla.

En definitiva, para ejecutar una sentencia tenemos que tipearla, finalizarla con ; (*punto y coma*) y presionar la tecla **<enter>**.

A medida que interactuemos con el programa cliente de MySQL, veremos que el prompt irá cambiando según lo que hagamos:

PROMPT	DESCRIPCIÓN
MySQL>	Listo para un nuevo comando.
->	Esperando una nueva línea.
'>	Esperando la siguiente línea, hay una cadena abierta con '.
">	Esperando la siguiente línea, hay una cadena abierta con ".

Tabla 7. Lista de posibles *prompt*.

La tercera y cuarta opción 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 otro par de comillas dobles; si abrimos la cadena con comillas simples, el cierre será otro par de comillas simples). 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;
```

III PASE Y NO VEA

Para ingresar al monitor de MySQL se puede usar la sintaxis **MySQL -u nombre_de_usuario -pcontraseña** o bien, **MySQL -u nombre_de_usuario -p**. Al ejecutar esto último, se nos pedirá la contraseña, con la diferencia de que esta vez, al tipearla, los demás no podrán verla, privilegiando así cuestiones de seguridad y evitando posibles espías.

```
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. Debemos saber que si quiere guardar estos resultados para imprimirlos o para leerlos más cómodamente, puede ingresar al monitor de MySQL poniendo:

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

O:

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

Con esto lograremos que todos los comandos y las salidas de los mismos se guarden en el archivo **uno.txt** (que en este caso se guardará en el mismo directorio donde se encuentra el programa MySQL, normalmente, el directorio **BIN**). El archivo no podrá ser abierto mientras la conexión esté activa.

Otra opción para ingresar al monitor es la siguiente:

```
MySQL -h servidor<enter>
```

Donde **servidor** es el servidor al que nos queremos conectar (allí estarán guardadas las bases de datos) y puede ser su nombre o bien su dirección **IP**. Si se omite esta opción se intentará conectar por defecto a la propia máquina desde donde se está trabajando. Lo anterior es muy importante porque pone de manifiesto la distinción entre MySQL

Cliente y MySQL Servidor, que podrían estar instaladas en diferentes máquinas, diferentes sistemas operativos, y tener diferentes versiones: más adelante veremos funciones que nos exigirán que sepamos distinguir entre clientes y servidores de MySQL. Por último, las opciones para ingresar al monitor son muchas y se pueden poner cuantas se quiera sin importar el orden de las mismas.

Para obtener un listado completo puede tipear:

```
MySQL -?<enter>
```

O bien:

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

Es importante que ambas hacen lo mismo, sólo que la primera muestra las opciones por pantalla, mientras que la segunda las guarda en este caso en el archivo **opciones.txt**, dentro de la misma carpeta **bin**.

... RESUMEN

En este capítulo hemos visto una introducción al gestor de bases de datos MySQL y las posibilidades que puede brindarnos en nuestras tareas de desarrollo. Algunas de estas opciones son: las plataformas soportadas, las diferentes versiones disponibles, los tipos de tablas (una para cada necesidad) y los tipos de datos soportados para definir nuestros campos. También repasamos las funciones que nos ofrece el lenguaje SQL y cómo ejecutar consultas a través de la línea de comandos de MySQL.



TEST DE AUTOEVALUACIÓN

- 1 ¿Desde qué sitio se puede descargar la última versión de MySQL?

- 2 ¿MySQL sólo puede trabajar sobre sistemas operativos MS Windows?

- 3 ¿Cuál es la ventaja de obtener una licencia comercial de MySQL?

- 4 ¿Se puede obtener y utilizar de forma libre MySQL?

- 5 Nombre tres tipos de tablas y sus características principales.

- 6 ¿Cuáles son los principales tipos de datos provistos por MySQL?

- 7 ¿Cuál es la diferencia entre los subtipos de datos DATETIME y DATE?

- 8 ¿Qué valor representa el timestamp de Unix? ¿Con qué función podemos saber el timestamp actual? ¿Esta función está disponible sólo en Unix o también en los demás sistemas?

- 9 Escriba una instrucción SQL para crear una tabla que guarde nombre, apellido, edad y fotografía de alumnos de una universidad (preste atención a los tipos de datos que utiliza, plantee las posibles opciones y elija la que le parezca más conveniente).

EJERCICIOS PRÁCTICOS

- 1 ¿Cuál es su versión de MySQL? Responda utilizando la función provista por MySQL para tal fin.

- 2 Utilizando las funciones DATE_FORMAT() y NOW() obtener la siguiente salida:

'el año xxxx será horrible para todos (los que no aprendan a usar mysql) !!!!'

donde xxxx es el año actual en 4 dígitos.

- 3 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, claro, las funciones matemáticas de MySQL.

- 4 Dada la cadena ' Carlos Marx ' obtener 'Groucho Marx'. Todo esto utilizando, claro, las funciones para manejo de cadenas de caracteres de MySQL.

- 5 ¿Cuál es el timestamp actual?

PHP y MySQL

En este capítulo hallaremos las distintas alternativas que nos ofrece PHP para acceder, y definir estructuras de bases de datos MySQL: sintaxis de las funciones, diferencias, ejemplos y comentarios acerca de cada una de ellas.

Cómo conectar PHP con MySQL	130
mysql_connect	130
mysql_pconnect	132
Ejecución de sentencias	133
mysql_db_query	133
mysql_query	134
Creación de base de datos a través de PHP	135
mysql_create_db	135
Selección de una base de datos	137
mysql_select_db	137
Creación de tablas a través de PHP	138
Obtener listados de bases de datos y tablas a través de PHP	140
mysql_list_dbs	141
mysql_list_tables	143
Utilización de múltiples bases de datos	144
Cuándo hacerlo	144
Cómo hacerlo	144
Cerrar conexión con la base	146
mysql_close	146
Ejemplo práctico: autos.php	147
Resumen	149
Actividades	150

CÓMO 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 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 en donde se encuentra instalado el servidor de bases de datos MySQL. Si estamos trabajando en forma local podemos utilizar como nombre de servidor **localhost** o **127.0.0.1** como dirección IP.
Opcionalmente se puede agregar el puerto de conexión (desde PHP 3), de la siguiente forma: **servidor:puerto** (por ejemplo, **localhost:80**).
- **NOMBRE DE USUARIO.** Se trata de nombre de usuario válido para poder tener acceso a la base de datos en cuestión.
- **PASSWORD.** Corresponde a la contraseña que corresponde al nombre de usuario que se ha ingresado anteriormente.

Como se dijo, los argumentos son opcionales y si no se especifica alguno se asumen los valores por defecto que se encuentran en el archivo **php.ini** (o **php3.ini** según la versión de PHP). 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 contenidas en el **php.ini** simplemente tenemos que escribir la información a continuación del signo igual. Por ejemplo, para completar sólo 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 **php.ini** 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, donde podemos incluir:

- Ningún argumento.
- Sólo el nombre del servidor.
- Sólo el nombre del servidor y el nombre de usuario.
- El nombre del servidor, el nombre de usuario y el password.

Debemos saber que esto significa que no se puede, por citar algún ejemplo, ingresar el password y dejar vacíos los demás argumentos.

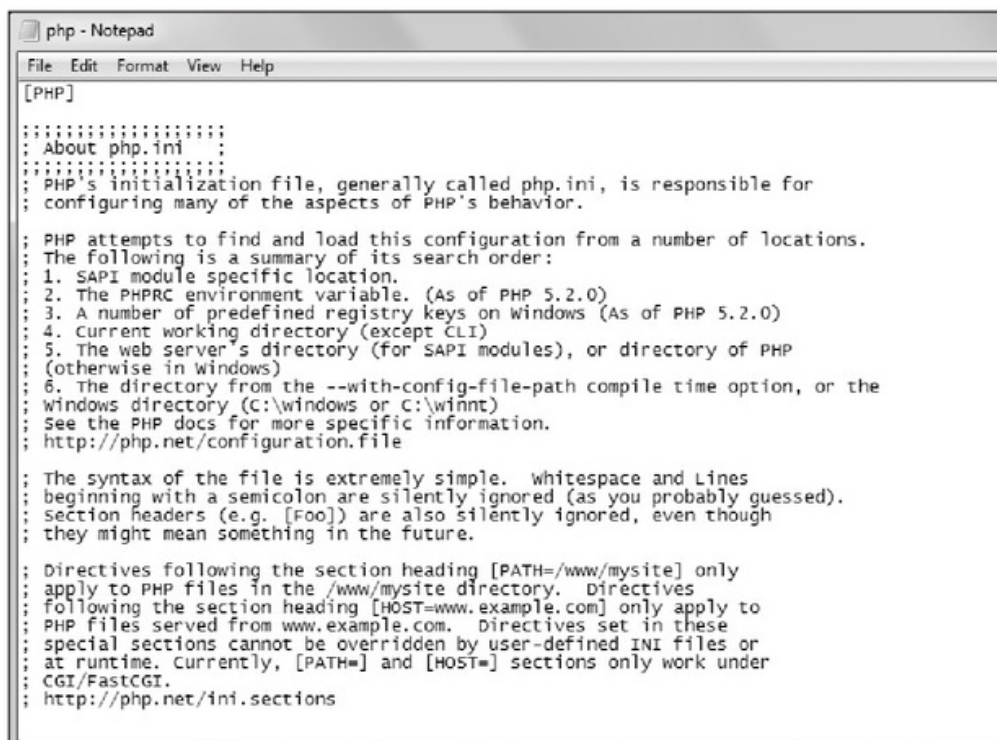


Figura 1. Configurando las opciones de MySQL en el archivo *php.ini*.

PHP tiene valores por defecto (no los incluidos en el **php.ini**, otros) que se usan cuando un determinado 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:** el del propietario del proceso del servidor.
- **Password:** vacía.

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. Si se logra la conexión devuelve, además, un **identificador de conexión**. Por ejemplo, en la conexión que vemos a continuación:

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

`$id_conexion` sería el identificador.

Nota importante: 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

La sintaxis de esta función es la siguiente:

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



TRABAJAR CON BASES DE DATOS

Recordemos que para empezar a interactuar con una base de datos (esto es, 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 llevado a cabo con éxito (mediante una de las funciones `mysql_connect` o `mysql_pconnect`).

Funciona de manera similar a **mysql_connect** con la diferencia de que las conexiones abiertas con **mysql_pconnect** (P de persistente) no se cierran –persisten– cuando termina la ejecución del archivo PHP (con **mysql_connect** sucede esto. Ver **mysql_close**).

Si en el momento en que se llama a esta función ya existe una conexión abierta idéntica, se usa; y en caso de que no exista, se la crea.

Cuando se dice “idéntica” se quiere decir que los argumentos “**servidor**”, “**nombre de usuario**” y “**password**” son iguales en las dos conexiones, tanto la conexión que se intenta abrir como la que se encuentra abierta.

Esta función, al igual de lo que sucede con la función **mysql_connect** que vimos anteriormente, también devuelve un identificador de conexión.

Ejecución de sentencias

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

mysql_db_query

Los argumentos que espera son:

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

- **Base de datos.** Aquí debemos ingresar el nombre de la base de datos sobre la cual queremos llevar a cabo 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:

```
<?php

$consulta_sql = "select * from alumnos";
mysql_db_query("base_ejemplo", $consulta_sql);

?>
```



```
<?php

mysql_db_query("base_ejemplo", "select * from alumnos");

?>
```

```
<?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);

?>
```

Esta función devuelve **1** (verdadero) si la instrucción pudo ejecutarse y se encarga de devolver el valor **0** (falso) si algo falló durante el proceso.

Nota: antes de proceder a enviar una instrucción SQL debemos tener establecida una conexión con el servidor correspondiente.

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 (;).

¿QUÉ PASÓ?

Algunas instrucciones SQL no devuelven registro alguno y, por lo tanto, no podemos saber con tanta facilidad si se llevaron a cabo correctamente o no. Para saber si se han ejecutado con éxito hay que tener presente que tanto la función **mysql_db_query** como **mysql_query** devuelven valores **0** ó **1** según se haya podido concretar lo requerido en la instrucción o no.

- **Identificador.** Se trata de un identificador de conexión. Este argumento es opcional y si elige 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 el valor **1** (que corresponde a verdadero) si la instrucción pudo ejecutarse con éxito y **0** (falso) si algo falló.

Nota: antes de enviar una instrucción SQL, es necesario que tengamos establecida una conexión con el servidor correspondiente.

CREACIÓN DE BASE DE DATOS A TRAVÉS DE PHP

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

`mysql_create_db`

Su sintaxis es la siguiente:

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

- **Base de datos:** se trata del 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:** este argumento es opcional y si se elige 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, ver `mysql_connect` en este mismo capítulo.

NO ES EL ÚNICO

Debemos saber que si bien SQL (*Structured Query Language*) es el lenguaje de consulta a bases de datos más utilizado de la actualidad, también existen otras opciones (como **QBE** –*Query By Example*–, **QUEL**, etcétera) que intentan hacerle sombra aunque no lo logren, ni por casualidad, al menos por ahora. Ya veremos qué pasa en futuras versiones

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 **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");
$db = mysql_create_db("mibase");
//en este caso se crea la base utilizando el
//identificador $con, que fue el ultimo abierto.

?>
```

```
<?php

$con1 = mysql_connect("198.92.34.1", "u", "p");
$con2 = mysql_connect("localhost", "u", "p");
$db = 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");
}

?>
```


Cada vez que se crea una base, ésta se almacenará –normalmente– en el directorio **mysql\data\nombre_base**, donde **mysql** es el directorio en donde se instaló MySQL.

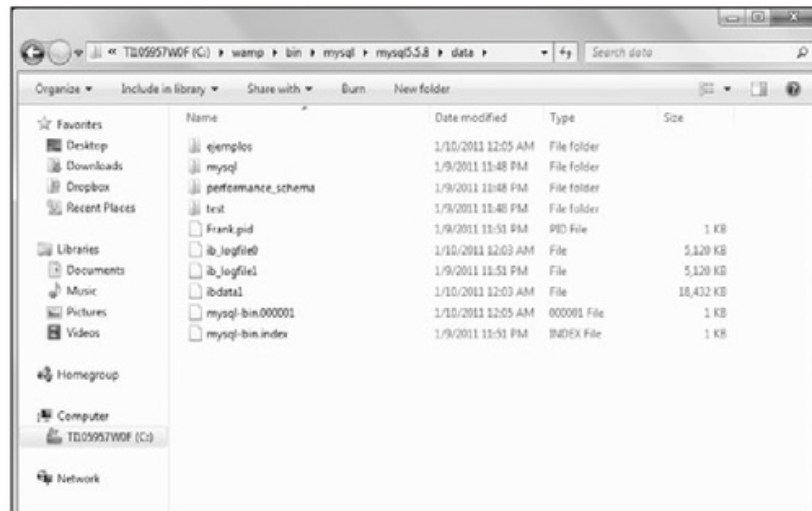


Figura 2. Bases de datos almacenadas en el servidor.

Para eliminar una base de datos desde PHP, podemos utilizar la función **mysql_drop_db** tal como lo vemos en el siguiente código:

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

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:

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

- **Base de datos:** nombre de la base de datos que queremos seleccionar y sobre la cual queremos realizar las consultas.
- **Identificador:** un identificador de conexión. Este argumento es opcional y si optamos por no incluirlo, la función asume la última conexión abierta con el servi-

dor MySQL. En caso de que no la encuentre, intentará crear una como si se llamara a **mysql_connect()** sin argumentos.

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

Algo importante es que esta función **liga** 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 base seleccionada y no habrá que volver a indicarla cada vez.

```
<?php

$link = mysql_connect("localhost", "u", "p");
$db = 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

No existe en PHP una función específica para crear tablas. Para hacerlo debemos utilizar la función **mysql_db_query** o la **mysql_query** ya explicadas anteriormente y escribir la instrucción SQL que corresponda. Veamos algunos ejemplos:

```
<?php

//cargamos en una variable toda la consulta
$sql = "create table t1 (";
$sql .= "c1 int(4) unsigned not null,";
$sql .= "c2 varchar(100)) ";
$sql .= "ENGINE=myisam";
//la ejecutamos
mysql_query($sql);

?>
```

```

<?php
//el mismo ejemplo anterior, pero con mensaje de confirmacion
//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";
}

?>

```



VALIDACIÓN DE INSTRUCCIONES

Al momento de ejecutar instrucciones SQL desde una página PHP, normalmente, no se suele prestar atención al hecho de validar si la instrucción se llevó a cabo con éxito o no. Este tipo de validaciones –explicadas en este mismo capítulo– serán de mucha ayuda para evitar e, incluso, solucionar algunos problemas que surgen a partir de errores no detectados a tiempo.

Para modificar la estructura de la tabla podemos hacer 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";
}

?>
```

Generalmente 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.

Por lo general, veremos en nuestros programas una gran cantidad de consultas a una base de datos por lo que, aunque sea en las primeras aplicaciones, sería lo más conveniente trabajar de la misma forma que en el último ejemplo. Esto significa, utilizar mensajes de confirmación de tareas. Esto deriva en el beneficio de no sólo saber dónde está el error sino, y a consecuencia de lo anterior, resolverlo de forma rápida.

OBTENER LISTADOS DE BASES DE DATOS Y TABLAS A TRAVÉS DE PHP

Para saber qué bases de datos existen en el servidor, PHP nos ofrece la función **mysql_list_dbs**. Para saber qué tablas se encuentran en una base de datos, PHP nos brinda otra función que nos será de utilidad: **mysql_list_tables**.



PHPMYADMIN

Debemos tener en cuenta que para darnos una idea de hasta dónde se puede llegar utilizando de manera conjunta **PHP** y **MySQL**, podemos poner como ejemplo una aplicación como **PHPMyAdmin** que, independientemente de ser utilizada por un gran número de usuarios, es realmente muy completa y detallada en cuanto a sus funcionalidades.

mysql_list_dbs

Sintaxis:

```
mysql_list_dbs(identificador);
```

Identificador se refiere a 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 con los ejemplos pertinentes, será de suma utilidad explicar la función denominada **mysql_tablename**.

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

Esta función toma dos argumentos: el primero es el puntero que apunta al 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 así 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).

El siguiente código merece una explicación:

```
<?php
$res = mysql_list_dbs ($link);
```

* LA VIDA DESPUÉS DE INTERNET

Las bases de datos no son propiedad exclusiva de Internet: su ámbito depende de qué proyectos queramos desarrollar 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, ya que existen otras opciones disponibles que podemos aprovechar.

```
$i = 0;
while ($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 de bases de datos MySQL, utilizando la conexión **\$link** (definida previamente).
- La cuarta línea inicializa la variable **\$i**.
- La quinta línea se encarga de recorrer la lista de bases de datos utilizando la función conocida como **mysql_fetch_row**.
- La sexta línea, utilizando 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**, como vemos a continuación:

```
<?php

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

?>
```

CREACIÓN DE BASES Y TABLAS I

Entonces, ¿por qué crear bases de datos o tablas desde PHP? Tal vez quisiéramos distribuir nuestra aplicación a usuarios finales y que ésta se encargue automáticamente de crear las diferentes estructuras para su correcto funcionamiento, y no dejar esta tarea en manos del usuario. Éste es sólo un ejemplo; hay otras ocasiones en las que quizá precisemos usar esta posibilidad de PHP.

Aquí, cada una de las veces 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 manipula como si se tratase de un objeto gracias a **mysql_fetch_object**) tomamos la propiedad **Database** (que representa el nombre) y la mostramos.

mysql_list_tables

Es importante saber que hace lo mismo 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 podemos utilizar la función **mysql_tablename**, que se encuentra explicada detalladamente en **mysql_list_dbs**.

Sintaxis:

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

- **Base de datos.** Corresponde al nombre de la base de datos de la cual queremos extraer los nombres de las tablas que necesitamos.
- **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. Debemos saber que si no la encuentra, intentará crear una como si se llamara a **mysql_connect()** sin argumentos.

Para obtener más información acerca de los identificadores de conexión y de cómo conectarse al servidor, podremos encontrar la función **mysql_connect** en el transcurso de este mismo capítulo.

Ejemplo:

```
<?php

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

?>
```

UTILIZACIÓN DE MÚLTIPLES BASES DE DATOS

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. A continuación damos algunos comentarios acerca de este 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 normalmente con mucha frecuencia, más de la que uno supondría.

Cómo hacerlo

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

```
<?php

//ejemplo con dos bases de datos que estan 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 estan guardadas en el mismo servidor
```

```
//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);
```

```
?>
```

La forma de trabajo es parecida a la que usamos al trabajar con sólo una base: lo que hay que tener presente es utilizar **mysql_select_db** cada vez que cambiemos de base.



TRABAJAR EN FORMA LOCAL

Para desarrollar aplicaciones en PHP y MySQL no es necesario montar una red: podemos trabajar en forma local aun teniendo una sola máquina. Lo que sí precisamos es tener una placa de red instalada y funcionando correctamente. De este modo, deberemos realizar la conexión con la base de datos configurando el nombre del servidor como **localhost** o la dirección IP como **127.0.0.1**.

CERRAR CONEXIÓN CON LA BASE

Para cerrar una conexión utilizamos la función **mysql_close()**. Tengamos en cuenta que se cerrará la conexión o no, según cómo 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 hay que decir que este tipo de conexiones se cierran automáticamente al terminar de ejecutarse cada página PHP en la cual hayamos abierto una conexión previamente. En cambio **mysql_close** no cerrará la conexión si ésta fue abierta utilizando la función **mysql_pconnect**. Su sintaxis es la siguiente:

```
mysql_close(identificador);
```

Identificador. Refiere a un identificador de conexión. Debemos saber que este argumento es opcional y si se opta por no incluirlo, la función se encarga de asumir 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) {
```

CREACIÓN DE BASES Y TABLAS II

Si bien es posible crear bases de datos y tablas desde PHP, también es posible hacerlo desde aplicaciones especialmente dedicadas a trabajar con MySQL, como **MySqlfront** o **mysql Control Center**. Esto supone una ventaja, porque estas aplicaciones ofrecen funciones avanzadas y estandarizadas para mantener bases de datos. Además, pueden ejecutarse sin tener instalado PHP.

```

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

?>

```

Para más información acerca de identificadores de conexión encontraremos **mysql_connect** en este mismo capítulo. 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

```

<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<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 instruccion 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 al crear la tabla \'modelo\'';

//se cierra la conexión
mysql_close();

?>
</body>
</html>
```

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

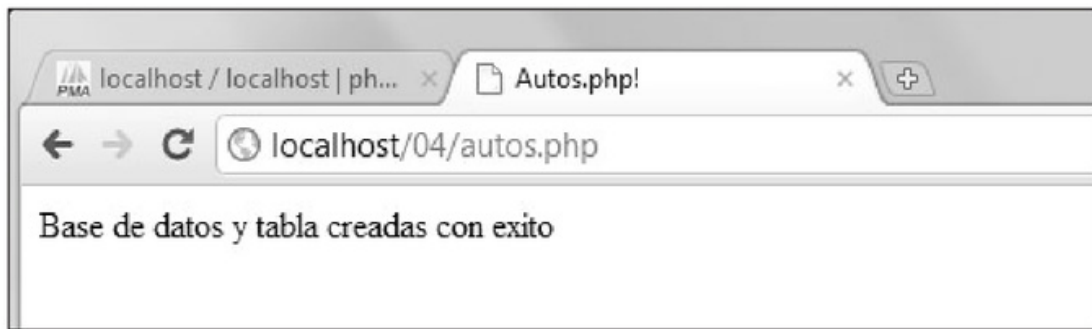


Figura 3. Salida de *autos.php*.

Y puede ver la nueva carpeta **autos**:

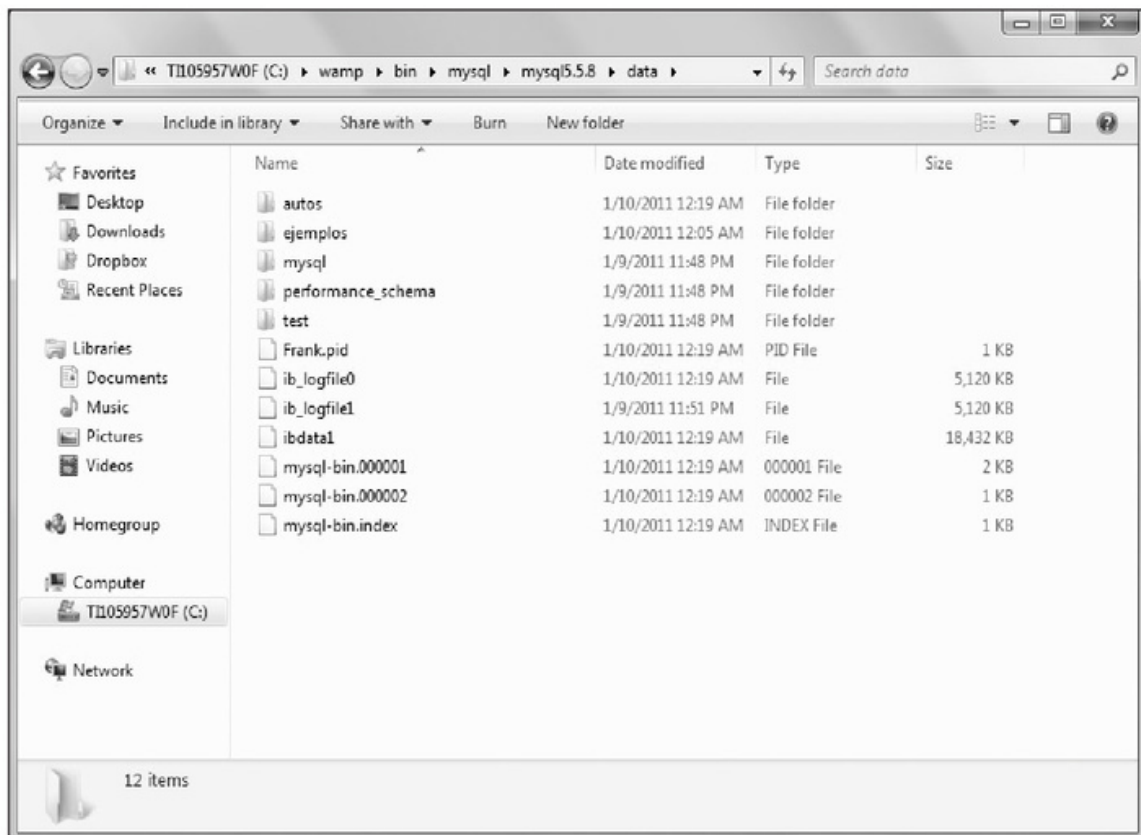


Figura 4. Bases de datos *autos* almacenada en el servidor.

... RESUMEN

Acabamos de aprender cómo ejecutar sentencias SQL para definir la estructura de una base de datos MySQL: creación y eliminación de la base de datos, creación y eliminación de tablas, establecimiento de conexiones a la base, desconexión de la base, cómo listar tablas, cómo listar bases de datos disponibles en el servidor, etc. En el próximo capítulo nos alejamos de las estructuras y las conexiones, y empezamos a manipular los datos de la base.



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?
- 4 ¿Mencione cuál es el inconveniente con `mysql_create_db`? ¿Cuál es la alternativa?
- 5 ¿Para que sirve la función `mysql_select_db`?
- 6 ¿Para que sirve PHPMyAdmin?
- 7 ¿Es posible utilizar la función denominada `mysql_connect` más de una vez dentro de un mismo script?
- 8 ¿Cuál es la utilidad de la función llamada `mysql_close`?
- 9 ¿Es posible configurar MySQL a través del archivo `php.ini`?
- 10 ¿Mencione en que casos crearía una base de datos a través de scripts PHP?

EJERCICIOS PRÁCTICOS

- 1 En el siguiente código:

```
<?php
$con1 = mysql_connect
      ("198.92.34.1", "u", "p");
$con2 = mysql_pconnect
      ("localhost", "u", "p");
mysql_close();
?>
```

¿Qué conexiones quedan abiertas?

- 2 Se quiere crear la tabla `t1`. En el siguiente código,

```
<?php
$con = mysql_connect
      ("localhost", "u", "p");
$dbas = mysql_create_db("mibase");
mysql_select_db("mibase", $con);
$tabla = "create table t1 ("
$tabla .= "c1 int(4) unsigned
          not null,";
$tabla .= "c2 varchar(100)) ";
$tabla .= "ENGINE=myisam";
mysql_close();
?>
```

Y suponiendo que la conexión con el servidor sea válida y la base se haya creado exitosamente ¿qué error se comete? ¿qué falta?

SQL en PHP

Una vez creada la estructura de la base de datos (ya sea a través de PHP o de algún programa cliente de MySQL), y establecida una conexión, debemos empezar a interactuar con los datos para poder realizar consultas de selección y de modificación (ingreso, baja y actualización de registros). He aquí un recorrido por las distintas funciones que PHP nos ofrece para cumplir con estas tareas.

Consultas de selección	152
Selección simple	152
Consultas multitabla	152
Subconsultas	152
Recorrer las filas devueltas de una consulta	153
mysql_fetch_array	153
mysql_fetch_row	158
mysql_fetch_object	158
mysql_fetch_assoc	160
mysql_fetch_field	163
Moverse entre registros	164
mysql_data_seek	164
Número de registros y campos devueltos	166
mysql_num_rows	166
mysql_affected_rows	167
mysql_num_fields	169
Insert	170
Campos autoincrementables	170
mysql_insert_id	171
Delete	172
Update	173
Ejemplo práctico: libros.php	174
Resumen	187
Actividades	188

CONSULTAS DE SELECCIÓN

El lenguaje de consulta SQL es a la vez sintético y muy potente. Cuando queremos realizar consultas de selección (esto es recuperar registros de una o más tablas) podemos recuperar millones de registros con tan sólo una línea de código. A continuación, nos dedicaremos a analizar las distintas posibilidades que existen para extraer datos y luego trabajar con ellos.

Selección simple

Con el término **selección simple** hacemos referencia a consultas a una sola tabla por vez. Este tipo de consultas no son frecuentemente utilizadas en aquellos sistemas que poseen un diseño complejo.

```
SELECT * FROM TABLA1;
```

Consultas multitable

Consisten en realizar consultas que implican varias tablas a la vez, esto significa que se recuperan datos de varias fuentes.

Son bastante comunes y más adelante, en el transcurso de este mismo capítulo, veremos en forma detallada cómo solucionar el problema que representa tener dos o más columnas de varias tablas diferentes que poseen el mismo nombre.

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

Subconsultas

Las subconsultas se implementaron en MySQL a partir de la versión 4.1.

```
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 la versión de MySQL que poseemos no incluye soporte para subconsultas, no hay de qué preocuparse: son perfectamente reemplazables, sólo 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, normalmente obtendremos un conjunto de registros. PHP permite almacenarlos en forma de objetos o, también, de arrays (matrices) con el objetivo de acceder a ellos utilizando índices ya sea numéricos o alfanuméricos. Haremos a continuación un recorrido por las distintas posibilidades.

mysql_fetch_array

Debemos saber que esta función nos permite recuperar filas y acceder a los valores de cada campo como si de una matriz se tratase.

Su sintaxis es la siguiente:

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

id_consulta: se envía 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**.

Para recuperar los datos devueltos por la consulta (si es que los hay) una de las funciones que se puede 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, ésta se posiciona en la primera fila (siempre y cuando exista una primera fila). En el ejemplo que podemos observar en la siguiente página, cada vez que se llama a la función **mysql_fetch_array**, se asigna a la matriz **\$fila** los campos resultantes de la consulta **\$sql**. Cuando no haya más filas, la matriz **\$fila** tomará el valor **FALSE** y el bucle **WHILE** termina.



DEPURAR EL LENGUAJE

Es importante verificar de manera detallada la sintaxis de las instrucciones SQL que utilizamos en las aplicaciones, puesto que 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.

```
<?php

$sql = "select * from tabla";
$res = mysql_query($sql);
while ($fila = mysql_fetch_array($res))    {
    .....
}

?>
```

tipo_índice: esta función trata a cada fila como si fuera una matriz. Este argumento permite definir el tipo de índice. Las opciones son **MYSQL_ASSOC**, **MYSQL_NUM**, y **MYSQL_BOTH**. Este argumento es opcional y si se opta por 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>";
}

?>
```



MATRIZ DEVUELTA POR MYSQL_FETCH_ARRAY

La función conocida como **mysql_fetch_array** se encarga de entregar un array que contiene los valores de cada registro devuelto por una consulta SQL (tal como se detalla a lo largo de este capítulo). Este array es tratado por PHP como cualquier otro, y es posible aplicar sobre él funciones tales como **count**, **in_array**, **sort**, y otras más.


```

<?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 hacen lo mismo.

Como podemos observar, los nombres de campos son sensibles a mayúsculas y minúsculas: no es lo mismo `$fila["campo1"]` que `$fila["Campo1"]`. La forma correcta para escribir los nombres de los campos dependerá exclusivamente de cómo los hayamos definido al momento de crear las tablas.

MYSQL_NUM

Se utiliza 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>";
    }

?>
```

Es importante saber 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 poner `$fila[0]` que poner `$fila["0"]` o `$fila['0']`.

MYSQL_BOTH

Esta función nos brinda la posibilidad de utilizar el número o el nombre del campo (columna) como índice. Es como utilizar `MYSQL_ASSOC` y `MYSQL_NUM` al mismo tiempo. Observemos el 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 son sensibles a mayúsculas y minúsculas: no es lo mismo `$fila["campo1"]` que `$fila["Campo1"]`. La forma correcta para escribir los nombres de los campos dependerá de cómo los definimos al crear las tablas. Si consultamos sobre más de una tabla, y por lo menos dos de los campos recuperados tienen el mismo nombre, sólo tendrá validez la última columna. Algunos ejemplos de esta situación y las posibles soluciones, pueden encontrarse en `mysql_fetch_assoc`, en este capítulo.



MYSQL_FETCH_ARRAY Y REGISTROS DEVUELTOS

En algunos ejemplos hemos usado la función `mysql_fetch_array` junto con la estructura de control **while** para recorrer las filas devueltas por una consulta. Si sabemos de antemano que la consulta devolverá sólo un registro, no será necesario utilizar un **while**; sólo deberemos proceder de este modo: `$fila = mysql_fetch_array($res)`; para luego trabajar sobre el array `$fila`.

mysql_fetch_row

Esta función es como utilizar **mysql_fetch_array** con el argumento **tipo_indice** igual a **MYSQL_NUM**, es decir que 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 se encarga de tratar 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);
```

id_consulta: se envía una consulta al servidor (con **mysql_query** o con **mysql_db_query**, por ejemplo) de bases de datos. Debemos saber que 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 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**).

tipo_indice: este argumento permite definir el tipo de índice.

Las opciones disponibles para este argumento son **MYSQL_ASSOC**, **MYSQL_NUM**, y **MYSQL_BOTH**. Es opcional y si elegimos no incluirlo se asume por defecto **MYSQL_ASSOC**. Debemos escribirlo con letras mayúsculas. Para mayor información, recomendamos ver **mysql_fetch_array**.

Por ejemplo, si tenemos la tabla:

Nombre Tabla: NBA			
Codigo	Nombre_y_Apellido	Nacionalidad	Estado_civil
001	Allen Iverson	EEUU	Casado
004	Emanuel Ginobili	Argentina	Casado
005	Eduardo Nájera	Mexico	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
```

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>";
}

?>
```

Los nombres de campos son sensibles a mayúsculas y minúsculas: no es lo mismo `$fila["campo1"]` que `$fila["Campo1"]`. La forma correcta para escribir los nombres de los campos dependerá de cómo los definimos cuando creamos las tablas.

Si consultamos sobre más de una tabla, y por lo menos dos de los campos recuperados tienen el mismo nombre, sólo tendrá validez la última columna. Ejemplos de esto y posibles soluciones pueden encontrarse en `mysql_fetch_assoc`, en este capítulo.

mysql_fetch_assoc

Esta función se comportará del mismo modo que al utilizar la denominada `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, sólo tendrá validez la última columna. Por ejemplo, si tenemos las siguientes tablas:

Nombre Tabla: TABLA1		
Código1	Nombre	Edad
1	Homero	38
4	Marge	37
5	Lisa	7
6	Bart	8

Nombre Tabla: TABLA2		
Código2	Nombre	Habitantes
1	Philadelphia	1200000
4	New York	4000000
5	Phoenix	2000000
6	Seattle	300000

```
<?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 - Philadelphia
//      4 - New York
//      5 - Phoenix
//      6 - Seattle

?>
```

```
<?php

$sql = "selectCodigo1, TABLA2.Nombre,
        Tabla1.Nombre, Edad from TABLA1, TABLA2 whereCodigo1=Codigo2";

$res = mysql_query($sql);

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

// la salida sera:
//      1 - Homero
//      4 - Marge
//      5 - Lisa
//      6 - Bart

?>
```

Para poder acceder a dos columnas o más que posean el mismo nombre, contamos con por lo menos dos posibilidades, tal como podemos observar en la siguiente página:

ÍNDICE DE LAS COLUMNAS

Es un error frecuente 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.

- 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 sera:
//      1 - Homero - Philadelphia
//      4 - Marge - New York
//      5 - Lisa - Phoenix
//      6 - Bart - Seattle

?>
```

Los nombres de campos son sensibles a mayúsculas y minúsculas: no es lo mismo escribir **\$fila["campo1"]** que **\$fila["Campo1"]**. La forma correcta para escribir los nombres de los campos dependerá exclusivamente de la manera en cómo los hayamos definido al momento de crear las tablas.

Si necesitamos trabajar sólo con índices numéricos o con índices numéricos y asociativos, utilizaremos la función **mysql_fetch_row** o bien, **mysql_fetch_array**.

{ } ELECCIONES

Al revisar las funciones provistas por PHP para recuperar y recorrer las filas devueltas por una consulta de selección, habremos notado que algunas realizan la misma tarea o que en principio las diferencias son mínimas. Al ganar experiencia desarrollando aplicaciones, deberemos implementar unas u otras de acuerdo con el problema que tengamos que resolver.

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 forma:

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

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

PROPIEDAD	DESCRIPCIÓN
name	nombre de la columna
table	name de la tabla a la que pertenece la columna
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 unica, si no 0
numeric	1 si la columna es numérica, si no 0
blob	1 si la columna es un BLOB, sino 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, y otros) en el **Capítulo 3**; e información sobre primary_key, unique_key, multiple_key, nombres de columnas, nombres de tablas y demás, en el **Capítulo 2**.

Forma de uso:

```
<?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 destinadas a recorrer las filas devueltas por una instrucción **SELECT** (esto es, **mysql_fetch_array**, **mysql_fetch_object**, **mysql_fetch_assoc**, **mysql_fetch_row**), observamos que ante cada llamada a estas funciones, se avanza al siguiente registro (si es que lo hay).

Para volver hacia atrás o bien para posicionarnos en algún registro específico existe la función **mysql_data_seek**.

mysql_data_seek

Su sintaxis es:

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

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

numero_fila: se encarga de indicar 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, si tenemos la siguiente tabla:

Nombre Tabla: TABLA1	
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, y 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 sólo sirve para sentencias tipo SELECT. Devuelve el número de filas de un resultado. Su sintaxis es la siguiente:

```
mysql_num_rows(id_consulta);
```

id_consulta: se envía 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**.

Por ejemplo:

```
<?php

$sql = "select * from tabla1";

$res = mysql_query($sql);

$numero_filas = mysql_num_rows($res);

echo "Numero de filas devueltas: ".$numero_filas;

?>
```

MANTENER LOS DATOS EN MEMORIA

mysql_data_seek 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**, mantiene los datos devueltos por ella, pudiendo acceder a ellos con **mysql_data_seek** o **mysql_fetch**.

Si bien podríamos obtener el mismo resultado realizando una consulta como:

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

en muchos casos nos la ahorraríamos utilizando `mysql_num_rows`.

mysql_affected_rows

Esta función sólo sirve para sentencias tipo INSERT, UPDATE o DELETE. Devuelve el número de filas involucradas en la última instrucción SQL (siempre y cuando sean INSERT, UPDATE o DELETE). En el código que se encuentra debajo podemos observar la sintaxis de esta función.

```
mysql_affected_rows(identificador);
```

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. Para obtener mayor información acerca de los identificadores de conexión y cómo conectarse al servidor, podremos consultar **mysql_connect**, que se encuentra explicado en el **Capítulo 4**.

Por ejemplo, si tenemos la siguiente tabla:

Nombre Tabla: TABLA1	
cod_s	nom_s
1	Batman
2	Robin
3	Acertijo
4	Gatúbela
5	Alfred

* ¿CUÁL USAR?

Es importante saber que la mayoría de las veces la diferencia entre las funciones denominadas **mysql_fetch_array**, **mysql_fetch_object**, **mysql_fetch_assoc** y **mysql_fetch_row** es prácticamente ínfima, y se recomienda usar la que más cómoda nos resulte, aunque **mysql_fetch_array** puede suplantar el funcionamiento de casi todas las demás.

```
<?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 ultima instruccion)

?>
```

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 devolver 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);
```

id_consulta: Se envía 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**. Por ejemplo:

```
<?php

//Devuelve el numero 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 una función que puede ser muy útil cuando se trabaja con campos de auto incremento.

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 al 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);

$sql = "INSERT INTO T1 (descripcion) VALUES ('lindo')";
mysql_query($sql);

$sql = "INSERT INTO T1 (descripcion) VALUES ('feo')";
mysql_query($sql);

$sql = "INSERT INTO T1 (descripcion) VALUES ('mas o menos')";
mysql_query($sql);
```



VALOR MÁXIMO DE ALMACENAMIENTO DE UNA COLUMNA

La función `mysql_fetch_field` permite obtener, entre otros datos, la longitud máxima de una columna. Esto puede ser particularmente interesante al 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.

```
//      la tabla T1 quedaria asi:
//      1      lindo
//      2      feo
//      3      mas o menos

?>
```

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);

$sql = "INSERT INTO T1 (descripcion) VALUES ('horrible')";
mysql_query($sql);

//      la tabla T1 quedaria asi:
//      1      lindo
//      2      feo
//      4      horrible

?>
```

mysql_insert_id

Esta función devuelve el valor (no el número de fila) del último campo autoincrementable ingresado y debe llamarse inmediatamente después del ingreso. Su sintaxis es:

```
mysql_insert_id(identificador);
```

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. Para más información acerca de identificadores de conexión y de cómo conectarse al servidor, consultemos **mysql_connect** en el **Capítulo 4**.

Por ejemplo, teniendo en cuenta el ejemplo anterior:

```
<?php

$sql = "INSERT INTO T1 (descripcion) VALUES ('pasable')";
mysql_query($sql);

//      la tabla T1 quedaria asi:
//      1      lindo
//      2      feo
//      4      horrible
//      5      pasable

echo "Ultimo valor de codigo : ".mysql_insert_id();

//devuelve 5

?>
```

DELETE

No existen funciones específicas para borrar tablas o registros de una tabla o columnas de una tabla: todo esto se hace mediante la función **mysql_query** o la **mysql_db_query**, vistas en el **Capítulo 4**.

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);

?>
```


Existe la función **mysql_drop_db** para borrar bases de datos completas. 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";

?>
```

UPDATE

Al igual que para borrar datos o estructuras, no existen funciones específicas para modificar o actualizar estructuras de tablas o registros: todo esto se hace mediante la función **mysql_query** o la **mysql_db_query**, vistas en el **Capítulo 4**. Por ejemplo:

```
<?php

$sql = "update tabla1 set campo = 10";
mysql_query($sql);

$sql = "alter table tabla1 modify campo BIGINT NOT NULL";
mysql_query($sql);

?>
```



ELIMINACIONES

Es muy importante que notemos la diferencia entre utilizar instrucciones para borrar tablas e instrucciones para borrar todos los datos (filas) de una tabla. En el primer caso se borra la estructura completa y todos los datos que contiene, mientras que en el segundo caso se mantiene la estructura y solamente se eliminan los datos correspondientes.

EJEMPLO PRÁCTICO: LIBROS.PHP

Para el ejemplo **libros.php**, hemos definido la estructura para la base de datos que podemos observar a continuación:

NOMBRE TABLA: 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

NOMBRE TABLA: AUTOR_LIBRO		
#	COD_A	CÓDIGO DEL AUTOR
#	COD_L	CÓDIGO DEL LIBRO

NOMBRE TABLA: 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

NOMBRE TABLA: NACIONALIDAD		
#	COD_N	CÓDIGO DE NACIONALIDAD
	DESC_N	DESCRIPCIÓN DE NACIONALIDAD

En esta estructura de base de datos, estamos en condiciones de visualizar las relaciones que vemos en la página siguiente:



DISEÑAR LAS APLICACIONES

Recordemos que antes de empezar a escribir un programa es conveniente dividirlo en funciones. Esto, junto con los comentarios, nos ayudará a generar aplicaciones claras, legibles y entendibles, tanto para nosotros como para otros desarrolladores a lo largo del tiempo. Además, podremos sacar notable provecho del código escrito y reutilizarlo.

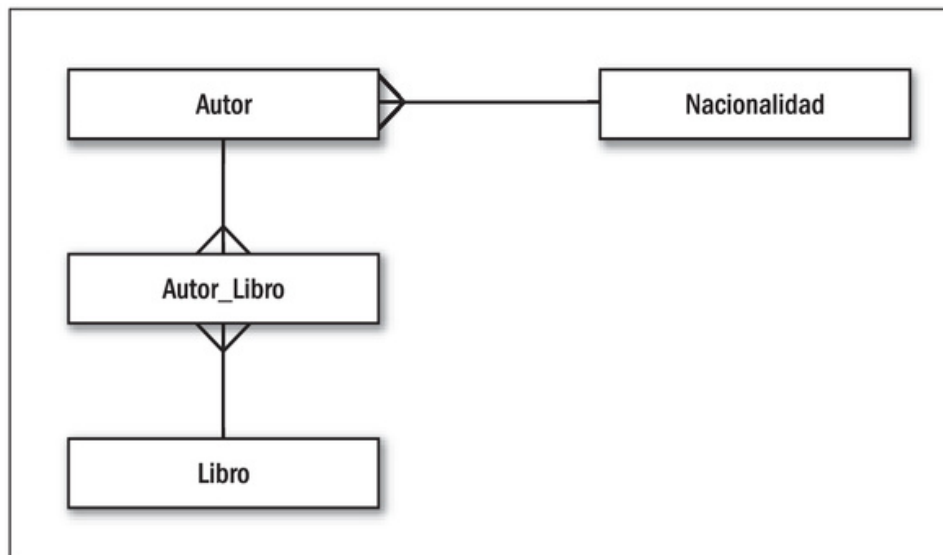


Figura 1. Relaciones entre tablas.

Como primer punto vamos a definir el archivo **config.inc.php**, el cual contendrá las directivas para conectar a la base de datos:

```

<?php

//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 selecciona la base
mysql_select_db($nombre_base, $link);

?>

```

Luego vamos a definir el archivo **crear_estructura.php**, que nos va a permitir 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**, o sea que se creará otra base de datos (si optamos por esto último el nuevo nombre de la base se deberá actualizar en el archivo **config.inc**).

Podríamos modificar el archivo **crear_estructura.php** para que cada vez que éste se ejecute, borre la base de datos llamada “libros” y la cree otra vez. Podemos intentarlo.

```
<?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_1 INT UNSIGNED NOT NULL,";
$sql .= "nom_1 VARCHAR(100),";
$sql .= "cod_a INT UNSIGNED NOT NULL,";
$sql .= "cant_pag_1 INT UNSIGNED,";
$sql .= "isbn_1 VARCHAR(20),";
$sql .= "foto_1 VARCHAR(100),";
```

LÍNEA DE COMANDOS

Desde PHP es posible no sólo programar aplicaciones con interfaz web (es decir, aquellas que únicamente pueden ser visualizadas desde un navegador), sino también otras sin interfaz, que sean invocadas a través de la línea de comandos. Ésta es una opción interesante para desarrollar programas que realicen una tarea específica y que privilegien su funcionalidad por sobre su diseño.

```

$sql .= "precio_1 FLOAT (5, 2),";
$sql .= "PRIMARY KEY (cod_1)";
$sql .= ") ENGINE=MYISAM";
mysql_query($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);

```



MYSQL_FETCH_ARRAY

En algunos ejemplos de este capítulo hemos utilizado 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 un registro, no será necesario utilizar un **while**; simplemente procedemos así: **\$fila = mysql_fetch_array(\$res)**; para luego trabajar sobre el **array \$fila**.

```
//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';

?>
```

Debemos tener en cuenta que en el archivo anterior nos encargamos de insertar datos en las tablas. Lo hicimos desde PHP, pero podemos hacerlo por otros medios (el monitor del gestor MySQL, por ejemplo).

Si todo funcionó bien, luego de ejecutar esta página en nuestro navegador deberíamos obtener los siguientes resultados en el monitor:

ACCESO A DATOS EN MEMORIA

mysql_data_seek nos permite posicionarnos sobre algún registro particular del array de datos. Lo interesante es que 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 y se evitan consultas extra hacia el servidor.


```

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:

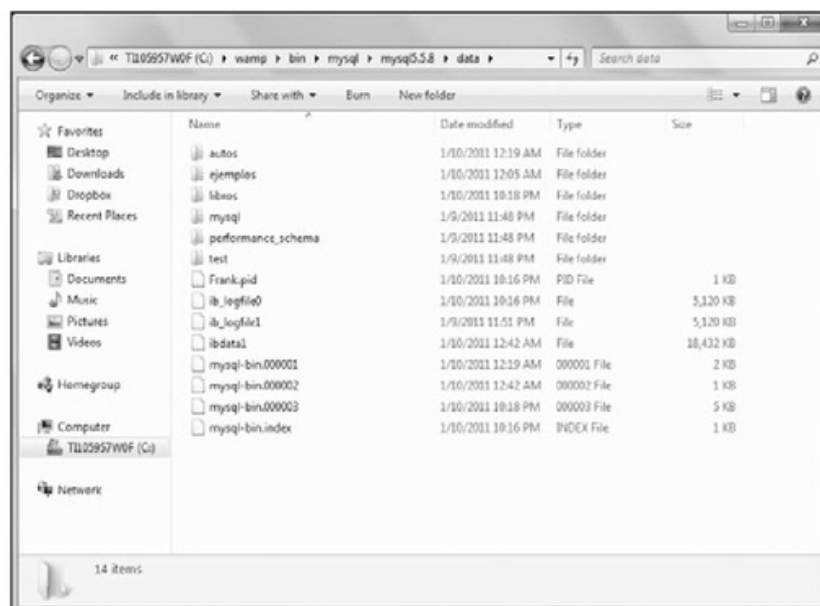


Figura 3. La base de datos creada, en el sistema de archivos.

A continuación definimos el archivo denominado **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 nacionalidad order
by desc_n asc");
$select_nacionalidad = '<select id="cbo_nacionalidad"
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>&nbsp;</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>&nbsp;</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 boton
    &quot;Buscar&quot;</td>
</tr>
</table>

</body>
</html>

```

Y por último, el archivo denominado **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'])
    $sql .= " and isbn_1 = '".$_POST['txt_isbn']."'";

if ($_POST['txt_palabra_clave'] != '')    {
    $sql .= " and (";
    $sql .= " nom_1 like '".$_POST['txt_palabra_clave']."'";
    $sql .= " or cant_pag_1 like '".$_POST['txt_palabra_clave']."'";
    $sql .= " or isbn_1 like '".$_POST['txt_palabra_clave']."'";
    $sql .= " or precio_1 like '".$_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_1";
$sql .= " order by nom_1";
$res = mysql_query($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($res).' libros encontrados';
        echo 'Resultado de la Búsqueda: '.$temp.':';

      ?>

    </td>
  </tr>
  <tr>
    <td>

<table cellpadding="4" cellspacing="0">

<?php

```



PLAN

Segun Oracle (propietaria de MySQL desde 2009), el plan es posicionar a MySQL como un competidor directo en relacion a SQL Server, el servidor de bases de datos de Microsoft. Podemos encontrar mas informacion si visitamos el sitio web que se encuentra en la dirección www.oracle.com/us/products/mysql/index.html.

```

if (!mysql_num_rows($res)) {
    echo '<td bgcolor="#COCOCO">&nbsp;No se encontraron resultados.</td>';
} else {

    $c=0;
    while ($f = mysql_fetch_array($res)) {

        $col = $c % 2 ? "#F0F3F8" : "#COCOCO";
        echo '<tr bgcolor='.$col.'>';
        echo '<td>'.ucwords(strtolower($f['nom_1'])).'</td>';
        echo '<td>'.ucwords(strtolower($f['ape_a']).',
'. $f['nom_a'])).'</td>';
        echo '<td>'. $f['cant_pag_1'].'</td>';
        echo '<td>'. $f['isbn_1'].'</td>';
        echo '<td>u$ $f['precio_1'].'</td>';
        echo '<tr>';

        $c++;
    }
}

?>

</table>

</td>
</tr>
<tr>
    <td colspan="2"><a href="buscar.php">Volver al buscador</a></td>
</tr>
</table>

</body>

</html>

```

Para continuar, debemos proceder a guardar los archivos ya creados; esto lo realizamos en una carpeta accesible dentro del servidor que estamos utilizando. Es importante tener en cuenta que debería quedar algo parecido al listado de archivos que vemos en la imagen presentada a continuación:

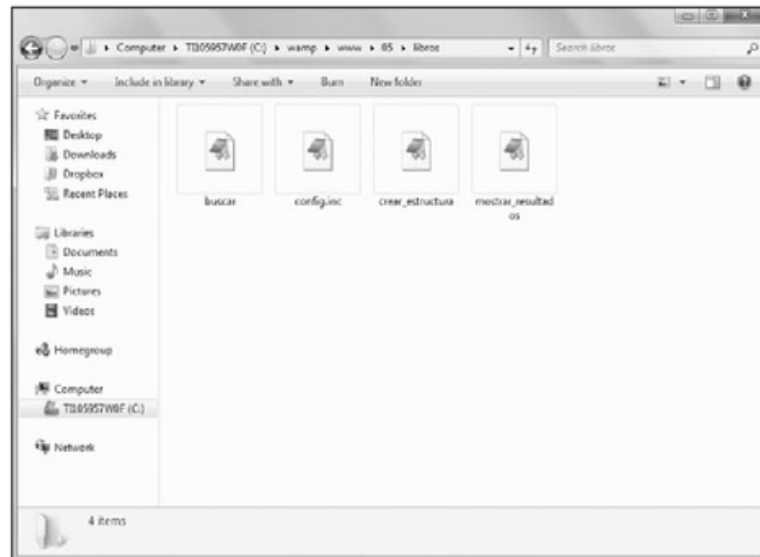


Figura 4. Listado de archivos del proyecto.

Recordemos que, si estamos trabajando de forma local, para acceder a una de estas páginas debemos tipear en nuestro navegador:

`http://127.0.0.1/nombre_pagina.php`

Esto, siempre y cuando hayamos guardado las páginas en el directorio raíz de nuestro servidor (normalmente HTDOCS). Si no las guardamos allí, debemos incluir en la dirección la ruta hasta **nombre_pagina.php**, por ejemplo:

`http://127.0.0.1/ejemplo1/paginas/nombre_pagina.php`

Tengamos presente que, antes de acceder a la página **buscar.php** se debe ejecutar una vez la página llamada **crear_estructura.php**.

... RESUMEN

La información que se guarda en una base de datos está, por lo general, sufriendo continuos cambios. A lo largo de este capítulo pudimos analizar cómo recuperar filas a través de consultas de selección, cómo insertar datos, cómo es posible modificarlos y cómo debemos borrarlos. Además, consideramos las distintas funciones para recorrer el conjunto de filas devueltas por una consulta, y vimos cómo acceder a cada campo en particular, terminando con un ejemplo completo de todo lo aprendido.



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 ¿Su versión de MySQL incluye soporte para subconsultas?
- 5 ¿Qué aplicación tienen las constantes `MYSQL_ASSOC`, `MYSQL_NUM` y `MYSQL_BOTH` en la función `mysql_fetch_array`?
- 6 ¿Qué función deberíamos usar si quisiéramos saber cuál es el nombre de una columna de una tabla?
- 7 ¿Qué hace la función `mysql_num_fields`?
- 8 ¿Qué es un campo autoincrementable? ¿En qué situaciones otorgaría esta cualidad a un campo de una tabla?

EJERCICIOS PRÁCTICOS

- 1 Supongamos la tabla y el código siguientes. ¿Cuál es la salida?

Nombre Tabla: TABLA1	
Cod	Nom
1	Viena
2	Edimburgo
3	Sevilla
6	Auxerre
5	Kiev

```
<?php
$con = mysql_connect("168.22.22.3",
    "u", "p");
mysql_select_db("base1");
$sql = "select * from TABLA1
    where Cod > 2";
$res = mysql_query($sql);
echo mysql_num_rows($res);
?>
```

Errores

Tener un conocimiento detallado de los errores cometidos nos da la posibilidad inmejorable de 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 nuestras aplicaciones.

Reportes de error en PHP	190
error_reporting	190
display_errors	193
display_startup_errors	194
log_errors	194
error_log	196
log_errors_max_len	196
ignore_repeated_errors	197
ignore_repeated_source	199
track_errors	200
html_errors	201
error_prepend_string	203
error_append_string	204
warn_plus_overloading	205
mysql_error	206
mysql_errno	207
register_globals	208
Códigos de error en MySQL	212
perro	212
Errores en INNodb	212
Manejo de excepciones	213
Resumen	213
Actividades	214

REPORTES DE ERROR EN PHP

PHP nos ofrece distintas funciones y directivas de configuración que nos permiten obtener diferentes niveles de reportes de error. A continuación, podemos observar un listado y ejemplos de cada una de ellas.

error_reporting

Para definir qué tipo de errores mostrar en nuestras páginas. Opciones:

CONSTANTE	DESCRIPCIÓN
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 / errores no fatales.
E_PARSE	Errores de compilación internos.
E_NOTICE	Errores no críticos, normalmente producto de fallas en nuestro código, como variables no inicializadas, por ejemplo.
E_CORE_ERROR	Errores fatales producidos durante el inicio de PHP. Disponible a partir de PHP 4.
E_CORE_WARNING	Advertencias –no errores fatales– de posibles errores producidos durante el inicio de PHP. Disponible a partir de PHP 4.
E_COMPILE_ERROR	Errores fatales en tiempo de compilación. Disponible a partir de PHP 4.
E_COMPILE_WARNING	Advertencias –no errores fatales– de posibles errores en tiempo de compilación. Disponible a partir de PHP 4.
E_USER_ERROR	Mensaje de error generado por el usuario que se asemeja a E_ERROR . Disponible a partir de PHP 4.
E_USER_WARNING	Mensaje de error generado por el usuario que se asemeja a E_WARNING . Disponible a partir de PHP 4.
E_USER_NOTICE	Mensaje de error generado por el usuario que se asemeja a E_NOTICE . Disponible a partir de PHP 4.
E_STRICT	Noticias de tiempo de ejecución para hacer que PHP sugiera cambios para mantener la compatibilidad de su código. Disponible a partir de PHP 5.
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, sólo que los mensajes de error pueden ser generados por el usuario.

Tabla 1. Tabla de errores de *error_reporting*.



SISTEMAS OPERATIVOS

Es muy importante saber que los mensajes de error que se presentan pueden variar de un sistema operativo a otro. Por esta razón, se recomienda tener cuidados especiales en este aspecto y tratar de conocer en profundidad las características técnicas de cada sistema operativo para no ser sorprendidos por las posibles incompatibilidades.

Si estamos desarrollando sitios, es conveniente tener habilitadas todas las opciones referidas al informe de errores. Todo lo contrario a si se está en etapa de producción, o sea, si nuestro sitio está en funcionamiento y es visitado por sus usuarios.

A partir de PHP 6, se incluye el nivel de errores **E_RECOVERABLE_ERROR**, que podremos usar en lugar de **E_ERROR** para identificar errores que requieren control específico, pero que produzcan una inestabilidad no definitiva. Además, el nivel de errores **E_ALL** contendrá al nivel **E_STRICT**.

Desde PHP versión 4 se puede configurar el reporte de errores a través de constantes simbólicas que deben ser deshabilitadas explícitamente mediante el uso de algunos operadores. Éstos son:

OPERADOR	SIGNIFICADO
~	Negación
	Or (O)
&	And (Y)

Tabla 2. Operadores.

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
```



VARIABLES

Al intervenir en la programación de aplicaciones complejas, es muy probable que debamos interactuar con una gran cantidad de funciones y de variables. Mantener el orden en estos casos es imprescindible, y ubicar la información en módulos y archivos separados utilizando una cierta lógica será una costumbre que valoraremos en el futuro.

Esta forma de definir el nivel de comunicación correspondiente a los errores producidos difiere con respecto a PHP en su versión 3, donde el nivel de reporte se establecía mediante el uso de números, y de esta forma, si había más de un nivel, los números se sumaban.

En PHP podemos encontrar una función con el mismo nombre que la directiva vista anteriormente (**error_reporting**) que posee la ventaja de permitirnos pasar por encima de lo que se encuentra escrito en el archivo **php.ini** y utilizar el nivel de reporte de errores que le otorguemos a la función. Este valor permanecerá únicamente durante nuestra sesión, luego se perderá y se seguirá usando el valor que figura en el archivo **php.ini**.

La sintaxis de la función es la siguiente:

```
error_reporting(nivel);
```

Donde nivel puede ser uno de los siguientes valores enteros:

VALOR ENTERO	VALOR CONSTANTE
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
32	E_CORE_WARNING

Tabla 3. Valores de la función **error_reporting**.

Para seleccionar varias opciones a la vez, debemos sumar los valores enteros de las opciones. Por ejemplo, si quisiéramos utilizar las opciones denominadas **E_ERROR**, **E_WARNING** y **E_PARSE**, deberíamos escribir:

```
error_reporting(7);
```

La función llamada **error_reporting** se encarga de devolver el valor anterior, en forma de valor entero, que contenía la directiva.

También se puede utilizar la función **ini_set** para configurar el nivel de error, pero que tendrá validez sólo en tiempo de ejecución, de la siguiente manera:

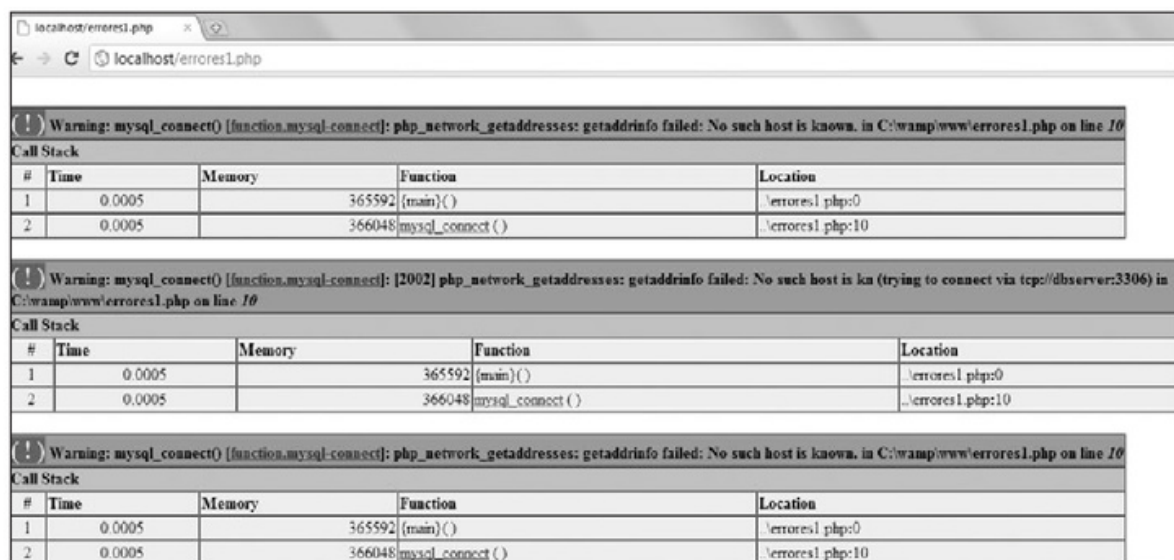
```
ini_set ("error_reporting", opcion);
```

Donde opción puede ser **E_ALL**, **E_WARNING**, etcétera.

Por medio del uso de la función **ini_set**, estamos en condiciones de modificar valores de otras directivas en tiempo de ejecución.

display_errors

Se pueden tomar dos valores: **1** o **0**. Si se toma el valor verdadero (**1**), al cometerse un error se mostrará el mensaje correspondiente por pantalla. Por ejemplo, si intentamos conectarnos a una base de datos inexistente obtendremos el siguiente mensaje a través de nuestro navegador:



The screenshot shows a web browser window with the address bar displaying 'localhost/errores1.php'. The page content shows three identical error messages and call stacks. Each error message is a warning from the MySQL extension, stating: 'Warning: mysql_connect() [function.mysql-connect]: php_network_getaddresses: getaddrinfo failed: No such host is known. in C:/wamp/www/errores1.php on line 10'. Below each warning is a 'Call Stack' table with columns: #, Time, Memory, Function, and Location. The first call stack shows two entries: entry 1 is the main function at line 0, and entry 2 is the mysql_connect function at line 10. The second and third call stacks are identical to the first.

#	Time	Memory	Function	Location
1	0.0005		{main}()	./errores1.php:0
2	0.0005		mysql_connect()	./errores1.php:10

Figura 1. Directiva *display_errors* habilitada.

El valor por defecto de esta directiva es **1**.

Sin dudas, tener activada esta directiva es muy útil durante el período de desarrollo de un sitio web, ya que nos notificará los errores que cometamos.

Si, en cambio, nuestro sitio está publicado y en etapa de producción, es conveniente no mostrar los mensajes de error, por más que estos ocurrieran, a los usuarios. Esto tiene que ver tal vez con una cuestión relacionada al diseño, a lo amigable que necesita ser siempre un sitio web.

* RECURRAMOS AL MANUAL

Es una buena costumbre tratar de resolver por cuenta propia los errores que cometemos. Pero en ocasiones, nos será beneficioso consultar la ayuda de PHP o de MySQL, y no quedarnos estancados perdiendo el tiempo para intentar solucionar algún inconveniente que se presente en el desarrollo de nuestras aplicaciones.

Incluso en ocasiones se producen errores que no modifican el buen funcionamiento del sistema pero que igualmente emiten mensajes de error al producirse.

De hecho, formalmente, podríamos decir que los errores deben solucionarse en la etapa de desarrollo del sitio, y sólo cuando no los haya podremos pasar a la publicación definitiva del proyecto correspondiente.

display_startup_errors

Pueden llegar a producirse errores durante el arranque de PHP, y estos errores no están contenidos entre los que se muestran o no configurando la directiva **display_errors**. Para mostrar o no los errores que se producen durante la carga de PHP debemos configurar la directiva **display_startup_errors**, que puede tomar los valores **1** (verdadero) o **0** (falso). Si está puesta a verdadero, mostrará los errores ocurridos, y si está a falso no los mostrará. Su valor por defecto es 0.

Nuevamente, recomendamos no tener habilitada esta directiva durante la etapa de producción de un sitio web, y en cambio sí durante la etapa de desarrollo.

Un error típico que se encuentra dentro de la categoría de errores de arranque, se da cuando PHP no encuentra directorios especificados en el archivo **php.ini**, como es el de las extensiones, o los definidos en la directiva **include_path**.

log_errors

Cada servidor web, generalmente, mantiene en un archivo un registro de todos los errores ocurridos. Estos errores pueden ser de cualquier tipo, desde errores al cargar módulos hasta problemas de configuración. Si queremos agregar a este archivo los errores que se producen en nuestras páginas PHP, deberemos configurar esta directiva. Esta directiva 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, o no, los errores producidos al mismo tiempo que los guarda –o no– en el archivo de errores del servidor, es decir que ambas directivas no son privativas ni están relacionadas de ninguna manera.



AYUDA EN PHP

Algunos lenguajes de programación traen junto a la ayuda del lenguaje sus entornos de desarrollo. Como PHP no posee un entorno de desarrollo único, es posible que tengamos que descargar la ayuda que se ofrece en el sitio web de PHP, ya que tampoco se incluye junto con la versión oficial. La ayuda viene en muchos formatos y para muchos sistemas operativos.

Si tenemos esta directiva habilitada, pronto –al cometer un error– podremos ver en el archivo de errores líneas como las siguientes:

```
[Thu Mar 10 22:19:55 2005] [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
```

Éstos nos marcan errores cometidos 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 se dan cuando escribimos de forma incorrecta líneas de código (por ejemplo, cuando no cerramos correctamente una cadena o cuando nos sobra o nos falta alguna llave para cerrar o abrir módulos de código). Veamos un 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
```

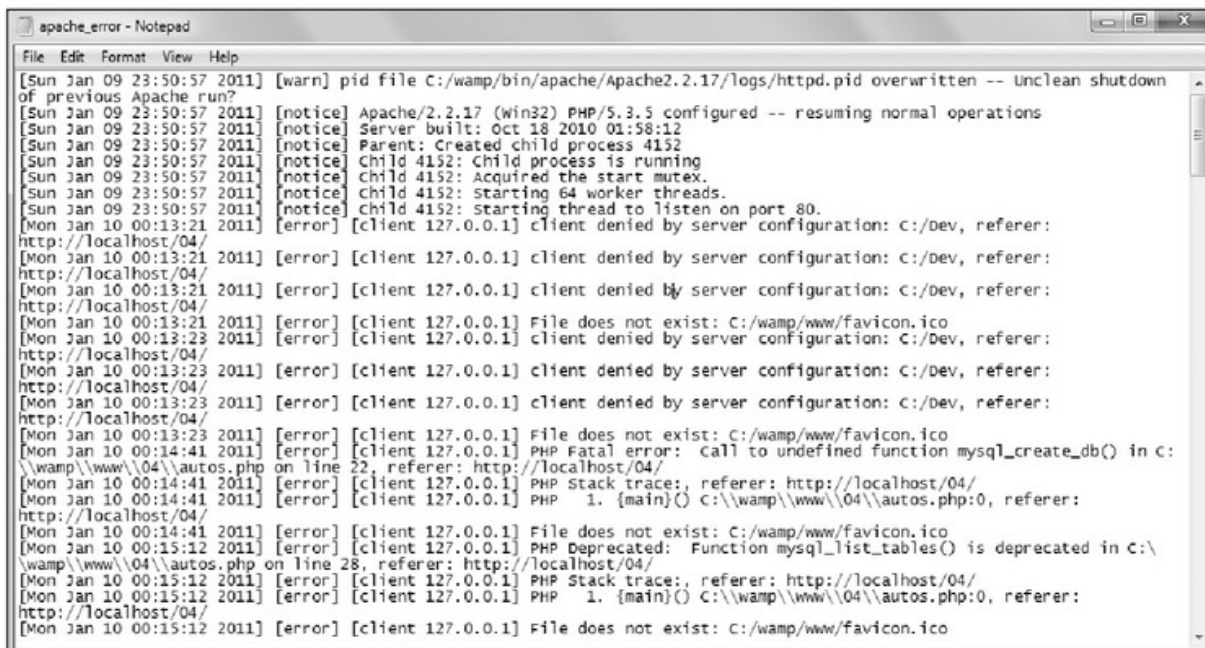


Figura 2. Archivo de errores.

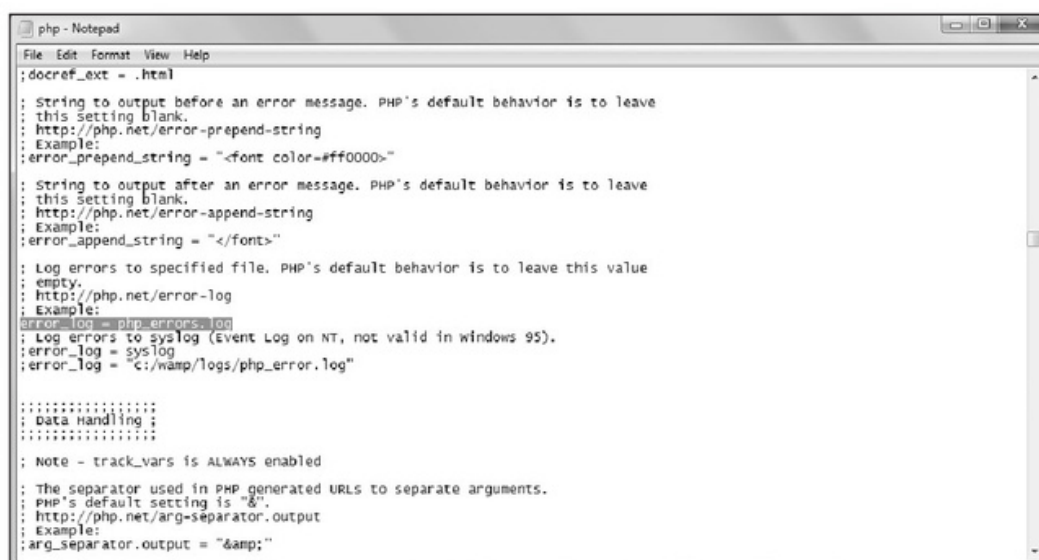
Si estamos utilizando el servidor web **Apache**, podremos ubicar este archivo en la carpeta llamada **LOGS**, dentro del directorio Apache.

El archivo de errores normalmente se llamará **error.log** y al ser un archivo de texto plano se puede abrir con cualquier editor de textos.

Si el servidor web Apache está activo, se encontrará imposibilitado para acceder al archivo **error.log**. Esto es lógico porque Apache necesita acceso permanente a este archivo debido a que lo modifica –en realidad sólo agrega líneas– de forma regular. Si se quiere leer este archivo se puede o bien realizar una copia y trabajar sobre ella o bien apagar el servidor web.

error_log

Desde esta directiva podemos configurar el nombre bajo el cual se guardará el archivo que almacenará los errores cometidos. Podemos especificar también la ruta hacia el archivo, de manera de poder ubicarlo donde queramos.



```

php - Notepad
File Edit Format View Help
;docref_ext = .html

; String to output before an error message. PHP's default behavior is to leave
; this setting blank.
; http://php.net/error-prepend-string
; Example:
; error_prepend_string = "<font color=#ff0000>"

; String to output after an error message. PHP's default behavior is to leave
; this setting blank.
; http://php.net/error-append-string
; Example:
; error_append_string = "</font>"

; Log errors to specified file. PHP's default behavior is to leave this value
; empty.
; http://php.net/error-log
; Example:
; error_log = php_errors.log
; Log errors to syslog (Event Log on NT, not valid in Windows 95).
; error_log = syslog
; error_log = "c:/wamp/logs/php_error.log"

;
; Data Handling
;
; Note - track_vars is ALWAYS enabled

; The separator used in PHP generated URLs to separate arguments.
; PHP's default setting is "&".
; http://php.net/arg-separator.output
; Example:
; arg_separator.output = "&"

```

Figura 3. Directiva **error_log**.

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.



AYUDA EN MYSQL

MySQL trae, junto a su distribución, una ayuda en formato HTML, que es realmente muy extensa y completa. Proporciona no sólo una referencia de las funciones y las características de la base de datos, también una serie de ejemplos y recomendaciones que facilitan mucho las cosas. Se puede descargar la ayuda en otros formatos desde el sitio web de MySQL (www.mysql.com).

Esta capacidad deberá ser definida en bytes y el valor por defecto de esta directiva es 1024 bytes, es decir, un MegaByte.

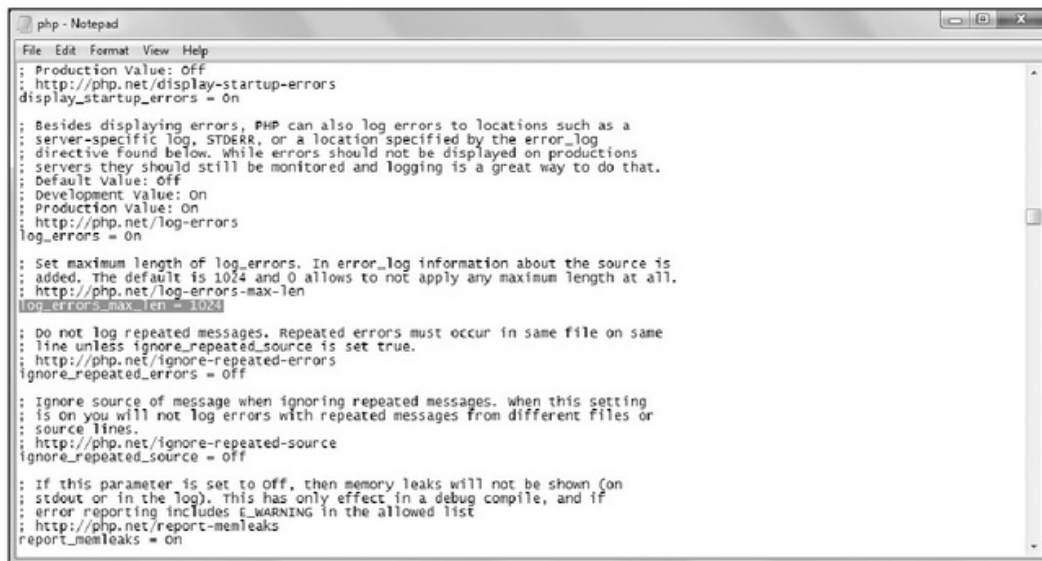


Figura 4. Directiva `log_errors_max_len`.

Si no queremos imponer límite alguno al tamaño del archivo, le asignamos 0 simplemente. Recordemos este hecho que normalmente se presta a confusión y que sucede también en otras directivas: puede que el valor **0** indique valores ilimitados, o sea que represente un valor infinito, no acotado. Si quisiéramos no utilizar el archivo de errores del servidor web para almacenar los errores relacionados con PHP, deberíamos configurar la directiva **log_errors**, vista antes.

ignore_repeated_errors

Si ponemos esta opción a 1, PHP mostrará los errores que se repiten pero sólo lo hará una vez. Se asume que los errores, para considerarse repetidos, deben ocurrir en la misma línea y en el mismo archivo.

Veamos un ejemplo:

```
<!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** puesto a 0, la salida será:

The screenshot shows a web browser window with the address bar displaying 'localhost/errores1.php'. The page content consists of three identical warning messages, each followed by a call stack table.

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 5. *ignore_repeated_errors* deshabilitada.

Y con **ignore_repeated_errors** puesto a 1, la salida será:

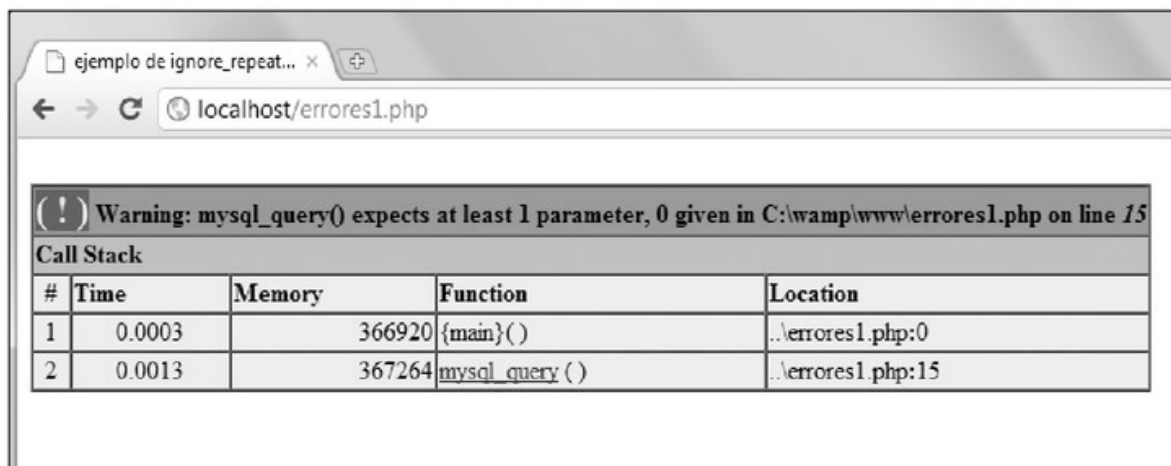


Figura 6. *ignore_repeated_errors* habilitada.

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**, etcétera).

Puede resultar más cómodo tener esta directiva habilitada, puesto que nos facilitaría y nos aclararía de manera considerable la visión de los errores cometidos al momento de verlos en la pantalla. Al acudir al código fuente de la aplicación y resolverlo, estaríamos solucionando a la vez todas las instancias del mismo.

El comportamiento de **ignore_repeated_errors** cambia según el valor de otra directiva llamada **ignore_repeated_source** que veremos a continuacións.

ignore_repeated_source

Cuando se explicó la forma de funcionar de la directiva **ignore_repeated_errors**, se dijo que para considerar un conjunto de errores como repetidos, éstos debían figurar en la misma línea del mismo archivo.

Debemos saber que la directiva denominada **ignore_repeated_source** lo que hace es modificar la definición de lo que es en sí un error repetido con respecto a como lo hace la directiva llamada **ignore_repeated_errors**: **ignore_repeated_source**; si es que 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, teniendo activada la directiva **ignore_repeated_source**, sólo veremos un mensaje de error correspondiente a la falla encontrada.

Debemos tener en cuenta que si mantenemos **ignore_repeated_source** a un valor 0 y mantenemos **ignore_repeated_errors** a un valor 1, el lenguaje PHP se comportará como se expuso en la sección **ignore_repeated_errors**.

Si mantenemos **ignore_repeated_source** a 1 e **ignore_repeated_errors** a 1, el lenguaje PHP se comportará como se expuso anteriormente.

track_errors

En PHP existe una variable llamada **php_errormsg** (**\$php_errormsg**) que nos permite acceder al mensaje correspondiente al último error cometido, siempre que la directiva **track_errors** esté habilitada. Suponiendo esto último, veamos el 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>
```

Cuya salida será la que se muestra en la figura:



Figura 7. Directiva **track_errors** habilitada.

La directiva denominada **track_errors** se encuentra deshabilitada por defecto. Para habilitarla, debemos asignarle el valor **On**.

html_errors

Teniendo esta directiva deshabilitada, los errores se muestran como mensajes en formato HTML. Ahora 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>
```



Figura 8. Directiva `html_errors`.

El cambio que se produce al habilitar esta directiva, asignándole el valor **On**, es que a los mensajes de error se les agregará un enlace que derivará en una mejor

descripción del problema surgido, que en general estará dentro del manual que se entrega junto al lenguaje de programación PHP.

Si no tenemos a nuestra disposición el manual de PHP, podremos descargarlo desde el sitio web **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.

Ahora, nos encargamos de abrir el archivo llamado **php.ini**, vamos a la línea que comienza con **html_errors** y cambiamos su valor a **On**:

```
html_errors = On
```

Luego, vamos a la línea que comienza con el texto **docref_root** y procedemos a escribir la dirección en donde se encuentra el manual:

```
docref_root = "/manualPHP/"
```

Aquí podemos usar también referencias a sitios de Internet. Por ejemplo:

```
docref_root = "http://www.direccion.com/manual"
```

Por último, nos ocupamos de indicar la extensión que tienen los archivos del manual obtenido (HTML, TXT, etcétera):

```
docref_ext = .html
```

En la **Figura 9** podemos ver la página en la cual encontramos el manual oficial del lenguaje de programación PHP:



REVISAR PERIÓDICAMENTE EL ARCHIVO DE ERRORES

Es importante saber que, como se dijo en este capítulo, los servidores web guardan en general un archivo que contiene los errores producidos desde un tiempo a esta parte. Es recomendable revisar este archivo periódicamente para observar posibles comportamientos irregulares, no sólo de PHP sino también del propio servidor.

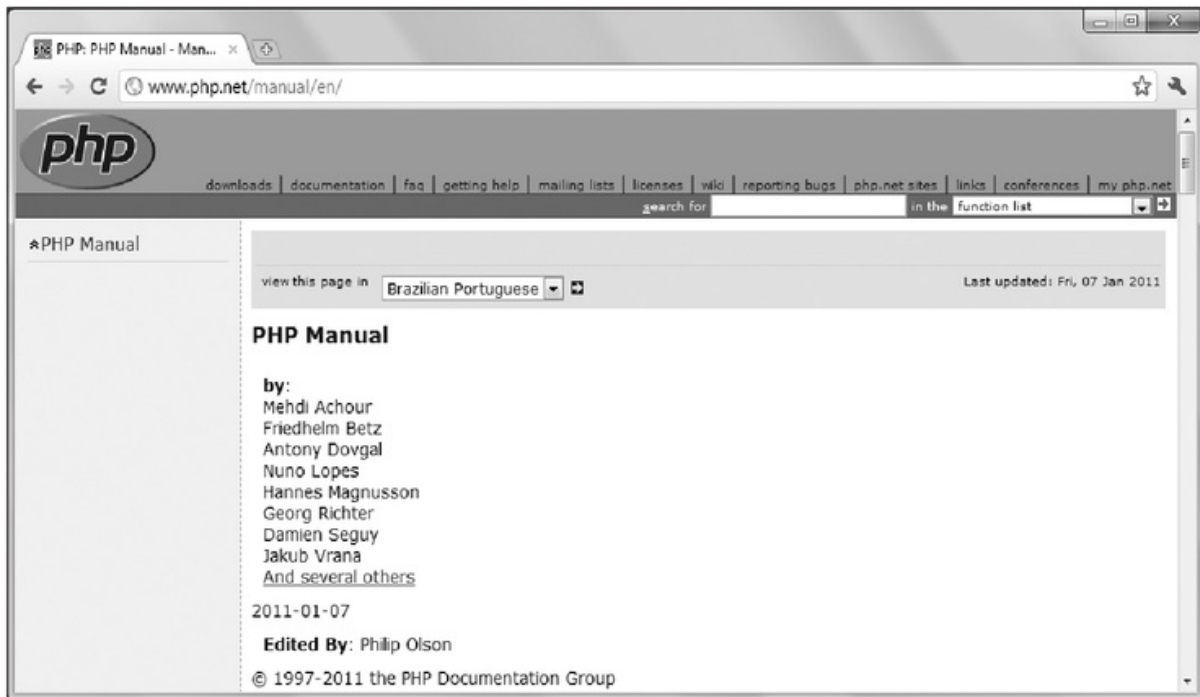


Figura 9. *html_errors* nos permite enlazar directamente al manual oficial de PHP.

error_prepend_string

Esta directiva nos servirá para mostrar contenidos que se verán antes de los mensajes de error. Si escribimos textos en formato HTML, esta directiva lo soportará y mostrará la salida correspondiente.

Es una directiva interesante en el sentido que nos permite personalizar el formato o la presentación de los mensajes de error, para adecuarlos a nuestras necesidades. No hace falta habilitar o deshabilitar expresamente esta directiva: bastará con asignarle valores o no. En el caso de asignarle algún texto, se considerará habilitada, y en el caso de no asignarle nada, se considerará deshabilitada.

Un ejemplo de esta directiva es el siguiente:

error_prepend_string = "Error!
"

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 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>

```



Figura 10. Directiva `error_prepend_string`.

error_append_string

Esta directiva es en extremo similar a `error_prepend_string` con la siguiente diferencia: su contenido se imprimirá luego del mensaje de error.

Supongamos que la directiva contiene lo siguiente:

```
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>

```

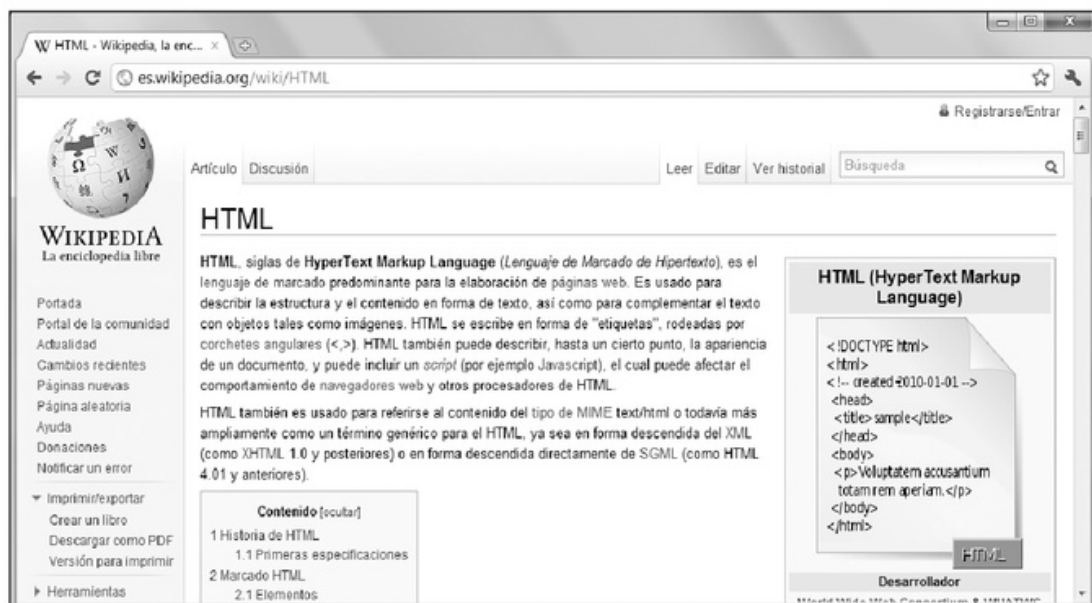


Figura 11. Podemos incluir código html a través de la directiva `error_append_string`.

warn_plus_overloading

Esta curiosa directiva, en caso de estar habilitada, nos informa a través de una advertencia, cuando usamos el signo 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 suma (+) 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, por supuesto. Su sintaxis es la siguiente:

```
mysql_error(identificador);
```

Identificador: es un identificador de conexión. Este argumento es opcional y si se elige no incluirlo, la función intentará encontrar 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.

Esta función no nos da mensajes de alerta (**WARNINGS**) acerca de posibles errores, sino que sólo trabaja con los errores ya consumados.

A continuación, damos algunos ejemplos acerca de cómo utilizar esta función.

```
<?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, hemos utilizado la construcción **DIE**. La diferencia entre usar **DIE** y usar **mysql_error** es que con la primera podemos personalizar los mensajes de error (además, **DIE** finaliza la ejecución del script) y con la segunda podemos obtener mensajes de error **oficiales** por parte de MySQL.



ERRORES CONOCIDOS

En las distribuciones de PHP, MySQL y Apache, suelen incluirse archivos con información acerca de los errores reportados por usuarios y todavía no solucionados. Incluso, si nosotros encontramos algún error hasta el momento desconocido, podemos reportarlo a través de un formulario, o a una dirección de correo electrónico que figura en la misma distribución.

Otra diferencia es que **DIE** puede usarse en cualquier ámbito, no sólo 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 de nuestro script se produjo el error, podremos solucionarlo sin mayores inconvenientes.

No obstante ello, siempre es una buena costumbre utilizar esta clase de funciones. En un gran número de oportunidades pueden sacarnos de ciertas situaciones que lo único que hacen es provocar pérdidas de tiempo y que, en la mayoría de los casos, no son tan difíciles de resolver.

mysql_errno

En MySQL –y posiblemente en cualquier lenguaje de programación o aplicación– los errores están catalogados y es posible identificarlos a través de un código. Se puede conocer este código utilizando la función **mysql_errno**, cuya sintaxis es absolutamente similar a **mysql_error**, vista antes:

```
mysql_errno(identificador);
```

El identificador de conexión es un argumento opcional, y si se elige no incluirlo, la función intenta encontrar la última conexión abierta activa al servidor MySQL. Si no la encuentra, tratará de crear una como si se llamara a la función denominada **mysql_connect()** sin argumentos.

Un ejemplo de esta función:

```
<?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

Quizás la mayor fuente de errores en las aplicaciones escritas en PHP haya sido el cambio producido a partir de la versión 4.2.0, que consistió en el cambio del valor por defecto de la directiva **register_globals**.

En rigor, siempre existió la posibilidad de configurar esta directiva tal como lo podemos hacer hoy en día; el cambio se produjo a partir de que los desarrolladores de PHP y la comunidad de usuarios que lo utilizan notaron que el valor por defecto que tenía esta directiva podía traer inconvenientes relacionados con la seguridad de los sitios web. Debemos saber que el paso siguiente fue cambiar el valor por defecto y también recomendar a los usuarios mantenerlo así.

Ahora bien, este cambio tuvo y tiene repercusiones en la forma de programar las aplicaciones, por lo que existió la necesidad de modificar algunas cosas del código programado. Veamos cuáles son estos aspectos y cómo modificarlos.

Utilizando **register_globals = On**, PHP asume todas las variables como globales y no podrá diferenciar entre los diferentes tipos de variables (si usa una variable en su script que tiene el mismo nombre de una variable de sesión, PHP no podrá diferenciarlas y las tratará como a una sola).

Para solucionar estos problemas y trabajar con **register_globals = Off**, PHP nos ofrece una serie de arrays que son los siguientes:

- **\$_SERVER**
- **\$_GET**
- **\$_POST**
- **\$_COOKIE**
- **\$_FILES**
- **\$_ENV**
- **\$_REQUEST**
- **\$_SESSION**

Estos arrays fueron implementados en PHP durante la versión 4.1.0 y si se posee una versión anterior, se pueden usar los arrays que mostramos a continuación, que trabajan de forma similar a los anteriores:

- **\$HTTP_SERVER_VARS**
- **\$HTTP_GET_VARS**
- **\$HTTP_POST_VARS**
- **\$HTTP_COOKIE_VARS**
- **\$HTTP_POST_FILES**
- **\$HTTP_ENV_VARS**
- **\$HTTP_SESSION_VARS**

No existe el equivalente al array **\$_REQUEST** en las versiones precedentes a la 4.1.0.

La idea es identificar el método por el cual recibimos las variables y acceder a ellas por medio del array correspondiente. Por ejemplo, si enviamos variables a través de un formulario a una página, debemos recuperar las variables mediante el 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];

?>

```

A continuación, haremos una breve reseña de los distintos arrays que provee el lenguaje PHP para almacenar distintas variables.

- **\$_SERVER**. Contiene una serie de variables que son generadas por el servidor web. Algunas de las variables contenidas en este array –hay varias– 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 forma clara cuál es el problema de seguridad que supone usar **register_globals = On**.

Supongamos que tenemos un script que posee la función de autenticar usuarios. Luego de finalizado un determinado proceso, si el usuario se encuentra registrado, almacenamos en la variable **usuario** el valor **registrado**.

Si pasamos la variable por url de la siguiente manera:

http://www.misitioweb.com?usuario=registrado

Con **register_globals = On** cualquiera estaría registrado, evitando el proceso de registro (recuerde que PHP no necesita declarar las variables). Con **register_globals = Off** podríamos especificar la forma en la que recibiríamos la variable. Si especificamos que la variable será de tipo **SESSION**, no nos importa qué valor tenga **\$_GET[usuario]**, lo que nos importa es el valor que tenga **\$_SESSION[usuario]**.

- **\$_POST**. Idéntica en su uso a **\$_GET**, almacena las variables recibidas por el método **POST**; por ejemplo, a través de formularios.
- **\$_COOKIE**. Debemos saber que se trata de una matriz asociativa de variables pasadas al script actual a través de cookies HTTP.
- **\$_FILES**. Es importante destacar que 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. Es importante que otras variables de entorno incluyen las variables **CGI**.
- **\$_REQUEST**. Se trata de una matriz asociativa que guarda los contenidos de **\$_GET**, **\$_POST** y también los contenidos de **\$_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**. Se puede observar qué valor tiene la directiva denominada **register_globals** en su sistema, de dos maneras: la primera es accediendo al archivo **php.ini** buscando la línea que contenga algo parecido a esto:

```

; Leaving this value empty will cause PHP to use the value set in the
; variables_order directive. It does not mean it will leave the super globals
; array REQUEST empty.
; Default value: None
; Development value: "GP"
; Production value: "GP"
; http://php.net/request-order
request_order = "GP"

; whether or not to register the EGPCS variables as global variables. You may
; want to turn this off if you don't want to clutter your scripts' global scope
; with user data. This makes most sense when coupled with track_vars - in which
; case you can access all of the GPC variables through the $HTTP_*_VARS[]
; variables.
; You should do your best to write your scripts so that they do not require
; register_globals to be on; Using form variables as globals can easily lead
; to possible security problems, if the code is not very well thought of.
; http://php.net/register-globals
register_globals = Off

; Determines whether the deprecated long $HTTP_*_VARS type predefined variables
; are registered by PHP or not. As they are deprecated, we obviously don't
; recommend you use them. They are on by default for compatibility reasons but
; they are not recommended on production servers.
; Default value: on
; Development value: off
; Production value: off
; http://php.net/register-long-arrays
register_long_arrays = off

; This directive determines whether PHP registers $argv & $argc each time it
; runs. $argv contains an array of all the arguments passed to PHP when a script
; is invoked. $argc contains an integer representing the number of arguments
; that were passed when the script was invoked. These arrays are extremely
; useful when running scripts from the command line. When this directive is

```

Figura 12. Directiva **register_globals** en el archivo **php.ini**.

O también a través de la función **phpinfo**, como se ve en la imagen:



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
safe_mode_gid	Off	Off
safe_mode_include_dir	no value	no value
sendmail_from	you@yourdomain	you@yourdomain
sendmail_path	no value	no value

Figura 13. 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 anteriormente 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 sólo mantienen el funcionamiento de las aplicaciones que están programadas con **register_globals**.

* REGISTER_GLOBALS

Habilitar la directiva **register_globals** puede convertirse en una tentación, ya que bajo esa circunstancia, la forma de programar aplicaciones se vuelve más fácil; pero recordemos que, tanto por las cuestiones de seguridad referidas en este capítulo como porque en estos momentos casi nadie habilita esta opción, se recomienda tenerla deshabilitada si es que ya no lo está.

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.

CÓDIGOS DE ERROR EN MYSQL

A continuación analizamos un programa de MySQL, el cual nos servirá para interpretar mejor los mensajes de error: **perror**.

perror

MySQL exhibe mensajes de error y el código del 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.

Su sintaxis es la siguiente:

```
C:\> perror codigo [codigo] [codigo] [...]
```

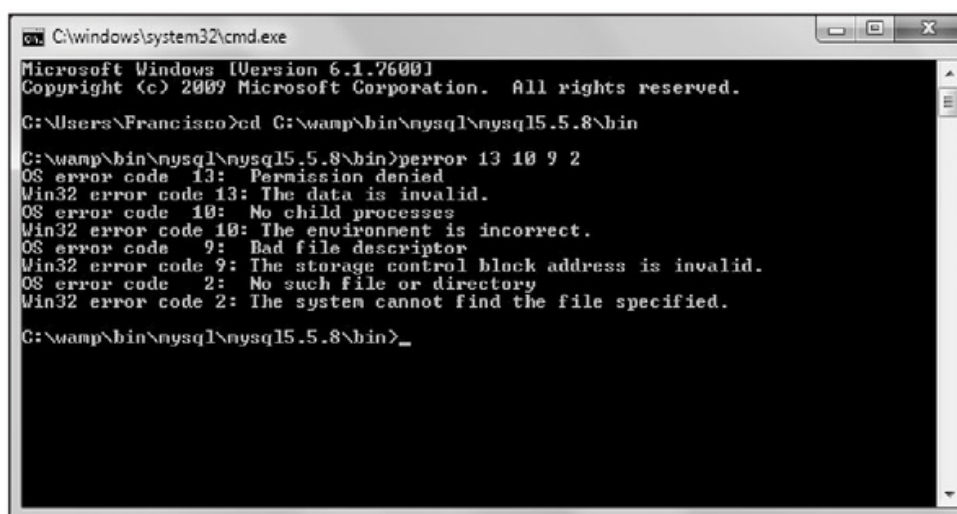


Figura 14. Salida del **perror**.

Las descripciones de los errores pueden variar de un sistema operativo a otro.

Errores en INNObd

Tal como se mencionara en los capítulos anteriores, el tipo de tabla **INNObd** se está convirtiendo rápidamente en un estándar para todo tipo de desarrollo que se quiera emprender, y en este capítulo uno de los temas principales donde focalizaremos será la forma que tiene de manejar errores.

INNodb no siempre maneja errores de la forma como lo hacen las tablas que adoptan lo especificado en el estándar SQL. De acuerdo a este estándar, cualquier error producido durante la ejecución de una instrucción SQL causará una vuelta atrás (**ROLLBACK**, que trataremos más adelante).

En cambio, **INNodb** en ocasiones sólo deshace parte de las instrucciones.

Si se producen referencias circulares (la tabla A espera que la tabla B se desocupe y a su vez la tabla B está esperando lo mismo de la tabla A) o se sobrepasa el tiempo límite durante un bloqueo, **INNodb** hará un **rollback** sobre la transacción completa. Si se ingresan claves duplicadas se hará un **rollback** sobre la instrucción SQL.

MANEJO DE EXCEPCIONES

Debemos saber que una excepción es un suceso o evento que ocurre y que interrumpe la ejecución normal de un programa. Pueden ser 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, éste crea un objeto que guarda información acerca de la excepción ocurrida. PHP puede lanzar, intentar o capturar excepciones, por medio de los bloques **try**, **catch** y **throw**.

Si la excepción no es capturada se producirá un error fatal y PHP mostrará un mensaje refiriendo esta situación. Las excepciones son comunes en lenguajes como Java o C++ y se implementaron en PHP versión 5.

... RESUMEN

En este capítulo nos hemos dedicado a realizar un análisis de una serie de funciones, programas y opciones de configuración que PHP y MySQL ponen a nuestra disposición para mantener conocimiento sobre los errores cometidos y poder administrarlos. Vimos también cómo programar nuestras aplicaciones ante la disposición por parte de PHP del nuevo valor por defecto que se le da a la directiva de configuración de `register_globals`, además de una introducción al manejo de excepciones en PHP versión 5.



TEST DE AUTOEVALUACIÓN

- 1 ¿Qué tipo de variables se guardan en el array \$_POST? ¿Y en \$_GET? ¿Y en \$_SESSION?
- 2 ¿Qué valor tiene la directiva error_reporting en su sistema? Responda:
a- Verificando el valor en el archivo php.ini.
b- A través de la función phpinfo.
c- A través de la función 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 las constantes E_ALL y E_WARNING?
- 5 Habilite la directiva log_errors en su sistema. ¿Qué valores contiene su archivo error.log?
- 6 Modifique el valor de las directivas error_prepend_string y error_append_string de tal forma que se muestre su nombre antes del mensaje de error y una línea horizontal (<HR>) después del mismo. Visualice los resultados obtenidos.
- 7 ¿Está habilitada en su sistema la directiva register_globals?
- 8 Brevemente, ¿qué es una excepción?

EJERCICIOS PRÁCTICOS

- 1 ¿Qué error cometimos si cometimos el error número 3 en MySQL?
- 2 ¿Cuál es la salida del siguiente código si usamos register_globals = Off? ¿Y si usamos register_globals = On?

```
<!--pagina1.php-->

<form action=pagina2.php
      method=POST>
  <input type=text name=nombre
        value='no hay salida'>
  <input type=submit>
</form>
```

```
<?php

//pagina2.php

echo $_POST[nombre];

?>
```


Transacciones

Las transacciones son algo relativamente nuevo en MySQL pero no en el modelo relacional. En este capítulo, haremos un recorrido por las ventajas que brinda usar transacciones, y su funcionamiento en los distintos tipos de tablas.

¿Qué es una transacción?	216
Casos de aplicación	216
Implementación de transacciones con MySQL	217
La variable autocommit	220
Tablas transaccionales	221
BDB	222
INNODB	222
Bloqueos	223
Bloqueos en INNODB: FOR UPDATE	224
Bloqueos en INNODB: LOCK IN SHARE MODE	227
Resumen	227
Actividades	228

¿QUÉ ES UNA TRANSACCIÓN?

Las transacciones consisten en un conjunto de instrucciones SQL que tienen la cualidad de ejecutarse como si de una unidad se tratara: o se ejecutan todas con éxito o, en caso contrario, no se ejecuta ninguna.

Un tipo de tabla que soporta transacciones se denomina **ACID** (**A**tomicidad, **C**onsistencia, **S**eparación, **D**urabilidad).

- **Atomicidad:** consultas tratadas como una sola, de tal forma que sólo se ejecutan cuando todas ellas tienen éxito; en caso de que alguna falle no se ejecuta ninguna. Algo atómico es algo que no puede ser dividido.
- **Consistencia:** sólo datos válidos pueden ser escritos en la base de datos. Se debe respetar la integridad referencial.
- **Separación:** las transacciones que tengan lugar simultáneamente deben ejecutarse aisladas y de forma transparente unas de otras hasta que finalizan. Recién cuando finalizan, tengan éxito o no, son visibles para los otros usuarios.
- **Durabilidad:** cuando una transacción se completa exitosamente, los cambios son permanentes y no se podrá volver atrás.

Casos de aplicación

Veamos el ejemplo para apreciar la importancia de las transacciones. Supongamos que dos clientes acceden a una base de datos y realizan las siguientes operaciones:

```
Cliente 1: SELECT precio FROM articulos;  
Cliente 2: SELECT precio FROM articulos;  
Cliente 1: UPDATE articulos SET precio = precio + (precio * 10/100);  
Cliente 2: DELETE FROM articulos WHERE precio = 300;
```

El **Cliente 1** y el **Cliente 2** obtienen el listado de precios de los artículos. Luego el **Cliente 1** actualiza los precios aumentándolos un 10%.

III EL USO DE TRANSACCIONES

Quizás, el uso de transacciones no esté del todo popularizado en lo que a pequeños y medianos sistemas se refiere, ya sea porque la cantidad de clientes que acceden de forma simultánea son pocos, o no se necesitan controles tan estrictos en cuanto a la consistencia de los datos. Sin embargo, es de gran ayuda empezar a utilizarlas aun en esta clase de emprendimientos.

El **Cliente 2**, que no sabe que los precios han aumentado, borra los artículos cuyo precio sea igual a 300. La visión que tiene el **Cliente 2** de los precios de los artículos no es real; si se hubieran utilizado transacciones, la escena anterior sería la siguiente:

```

Cliente 1: SELECT precio FROM articulos;
Cliente 1: UPDATE articulos SET precio = precio + (precio * 10/100);
Cliente 2: SELECT precio FROM articulos;
Cliente 2: DELETE FROM articulos WHERE precio = 300;
  
```

Es decir que las operaciones se hubieran hecho de forma ordenada permitiendo la consistencia de los datos correspondientes.

Otro ejemplo: supongamos que tenemos un sistema de venta de algún artículo. Cuando se concreta una venta debemos actualizar el stock y es evidente que necesitamos imperiosamente que se hagan las dos cosas o en su defecto ninguna, pero no una sola. Aquí entrarían a jugar las transacciones.

IMPLEMENTACIÓN DE TRANSACCIONES CON MYSQL

Para que el servidor SQL pueda diferenciar lo que es una transacción de lo que no es una transacción, éstas deben delimitarse con las siguientes palabras clave:

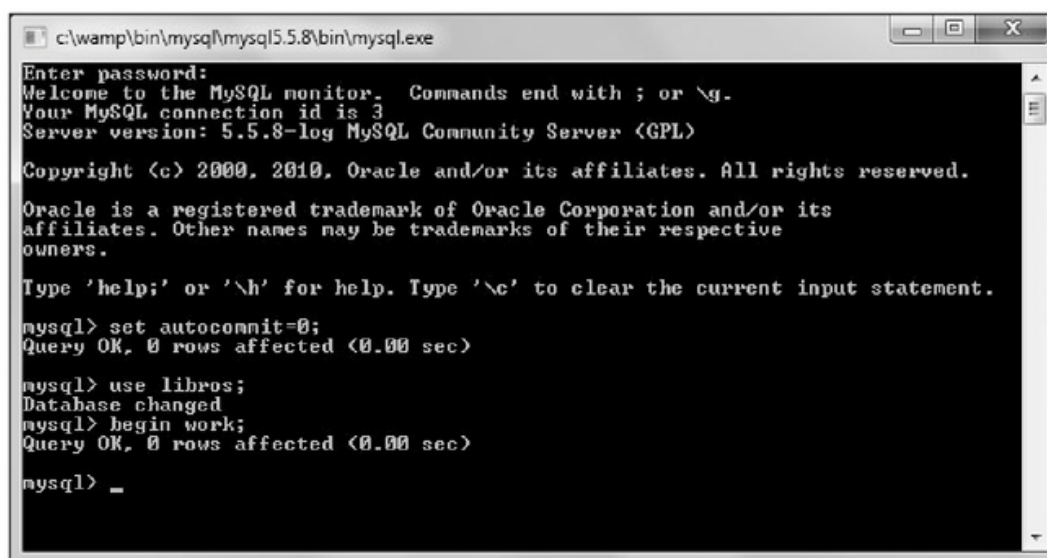
PALABRA CLAVE	DESCRIPCIÓN
BEGIN WORK	Comienzo de una transacción que finalizará con la palabra COMMIT o ROLLBACK.
COMMIT	Finalización de una transacción. Las instrucciones SQL que forman parte de la transacción se ejecutan y realizan los cambios pertinentes sobre la base de datos.
ROLLBACK	Finalización de una transacción. Las instrucciones SQL que forman parte de la transacción no se ejecutan.

Tabla 1. Palabras clave soportadas en las transacciones.

III EL FUTURO

De más está decir que MySQL no es la única base de datos que maneja transacciones: otras lo hacen desde hace más tiempo y, quizás por eso, el soporte que brindan es más completo.

Todavía queda mucho camino por recorrer para MySQL en éste y otros temas. También hay que decir que MySQL viene caminando rápido desde sus inicios.



```

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 3
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> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> use libros;
Database changed
mysql> begin work;
Query OK, 0 rows affected (0.00 sec)

mysql> _

```

Figura 1. Inicio de transacción con **BEGIN WORK**.

La transacción no se considera completa hasta que no se encuentre un **COMMIT** o un **ROLLBACK**. En el primer caso, la transacción se habrá ejecutado con éxito y en el segundo se habrá abortado. Si sucede algún inconveniente que afecta al servidor en medio de la transacción y no se llega a ejecutar ni **COMMIT** ni **ROLLBACK**, cuando se reinicia el servidor se asume **ROLLBACK** y la transacción no se lleva a cabo.

```

BEGIN WORK;
SELECT precio FROM articulos;
UPDATE articulos SET precio = precio + (precio * 10/100);
COMMIT;

```

```

BEGIN WORK;
SELECT precio FROM articulos;
DELETE FROM articulos WHERE precio = 300;
COMMIT;

```



TRANSACCIONES EXITOSAS

Al momento de programar aplicaciones que trabajen con transacciones, tengamos en cuenta que las palabras clave que delimitan las transacciones –tanto **BEGIN WORK** como **ROLLBACK**, y **COMMIT**– no devuelven registros, pero sí podemos llegar a saber si se ejecutaron con éxito o no, como se explicó en el **Capítulo 4**.

```

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 1
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> begin work;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from autor lock in share mode;
+----+-----+-----+----+-----+-----+
| cod_a | nom_a | ape_a | cod_n | fec_nac_a | fec_dec_a |
+----+-----+-----+----+-----+-----+
| 1 | julio | cortazar | 2 | 1912-11-12 | 1984-01-10 |
| 2 | jorge luis | borges | 1 | 1912-11-12 | 1912-11-12 |
| 3 | ernesto | sabato | 1 | 1912-11-12 | 1912-11-12 |
+----+-----+-----+----+-----+-----+
3 rows in set (0.36 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>

```

Figura 2. Finalización de una transacción a través de **COMMIT**.

```

DELETE FROM articulos WHERE precio=10;
BEGIN WORK;
DELETE FROM articulos;
ROLLBACK;

```

En el ultimo ejemplo, la transacción se aborta y la instrucción

```
DELETE FROM articulos;
```

no se ejecuta, pero sí se ejecuta la instrucción

```
DELETE FROM articulos WHERE precio=10;
```

porque está fuera de la transacción.

* ¡A ANIMARSE!!

Las transacciones parecen un tema lejano y fuera de nuestro alcance al principio, pero recomendamos ponerlas en práctica, y sobre todo, tratar de entender por lo menos los conceptos principales para tenerlas en cuenta al momento de desarrollar aplicaciones que involucren a muchos usuarios accediendo a los mismos datos simultáneamente.

Debemos saber que hay ciertos comandos, sin embargo, que finalizan una transacción por sí mismos, y son los que vemos a continuación:

COMANDOS
ALTER TABLE
BEGIN
CREATE INDEX
DROP DATABASE
DROP TABLE
RENAME TABLE
TRUNCATE

Tabla 2. Comandos para finalizar una transacción.

Utilizar uno de los comandos anteriores es igual a escribir un **COMMIT** antes de ellos. Desde el lenguaje de programación PHP podemos ejecutar los comandos para administrar transacciones de la siguiente forma:

```
mysql_query("COMMIT");
mysql_query("ROLLBACK");
mysql_query("BEGIN WORK");
```

La variable autocommit

En MySQL existe una variable llamada autocommit: si toma el valor 1, MySQL asumirá que cada una de las instrucciones SQL que no estén dentro de una transacción finalizan con **COMMIT**, aunque no lo escribamos. Si toma el valor 0, deberemos escribir específicamente alguno de los comandos **COMMIT.ROLLBACK** para realizar la acción que creamos conveniente.

Por defecto, MySQL trabaja en **autocommit=1**, lo que implica que inmediatamente al ejecutar una instrucción SQL que no forme parte de una transacción, se actualizará la base de datos. Debemos saber que lo mismo pasará con las transacciones cuando escribamos la palabra **COMMIT**.

Para las tablas transaccionales existe la posibilidad de cambiar este comportamiento de alguna de las siguientes maneras,

Ubicando al comienzo de cada página PHP lo siguiente:

```
mysql_query("SET AUTOCOMMIT=0");
```

O si estamos trabajando desde el prompt de MySQL, escribir:


```
mysql> SET AUTOCOMMIT=0;
```

De ahora en más será necesario utilizar los comandos denominados **BEGIN WORK**, **COMMIT** y **ROLLBACK** explícitamente según corresponda.

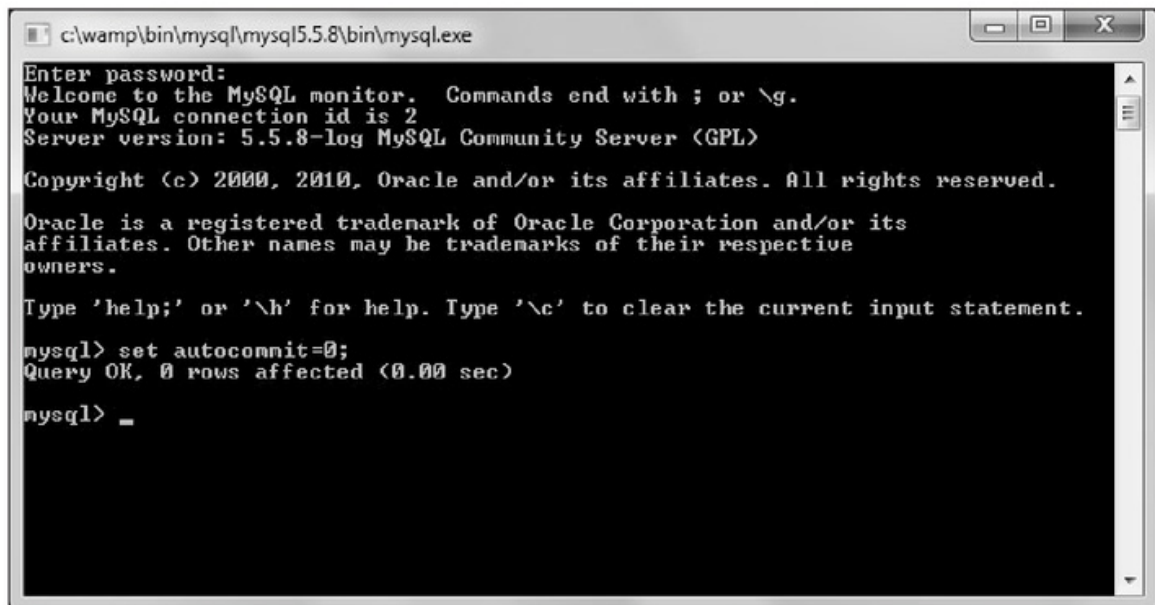


Figura 3. Cambiando el valor de Autocommit en el prompt de MySQL.

Tablas transaccionales

En sus comienzos, MySQL ofrecía únicamente un tipo de tabla: las **ISAM**, que luego derivaron en las **MyISAM**. Estos dos tipos de tablas mencionados, no incluían soporte para transacciones y hubo que esperar hasta la introducción en MySQL de las tablas **BDB** para que, finalmente, se diera este paso fundamental necesario para cualquier servidor SQL. Luego de que se crearan las **INNODB**, se sumaron a **BDB** en el sentido de proveer soporte para transacciones.

Si utilizamos los comandos denominados **BEGIN WORK**, **COMMIT**, **ROLLBACK**, autocommit o cualquier comando o variable que se aplique a tablas transaccionales sobre una tabla que no soporte transacciones –por ejemplo, las **ISAM** o **MyISAM**– recibiremos mensajes de advertencia o simplemente ningún mensaje, en cuyo caso los cambios no surtirán efecto alguno.

Tal como se había mencionado anteriormente, MySQL cuenta con varios tipos de tablas que soportan transacciones y cada una de ellas las implementa de una manera totalmente distinta. En las páginas siguientes, nos dedicaremos a dar un vistazo a las distintas formas de hacerlo de cada una para, de esta manera, poder observar y comprender sus ventajas y desventajas.

BDB

Estas tablas ofrecen bloqueos a nivel de página. Una página consiste en un conjunto de filas cuyo número se define a partir de varios factores, como son la cantidad de registros de la tabla, la versión de MySQL, el sistema operativo, etc.; pero lo importante es que, como se dijo anteriormente a propósito de las tablas transaccionales, se nos permite bloquear sólo algunos registros sin la necesidad de bloquear la tabla completa, lo que se reduce a que eventualmente muchos clientes puedan acceder simultáneamente a distintas partes de la tabla.

Si un cliente comienza una transacción y trabaja sobre determinada/s fila/s, las páginas que contengan esas filas se bloquearán. Si en ese momento otro cliente inicia su propia transacción e intenta realizar operaciones sobre filas pertenecientes a la misma página, la acción quedará suspendida hasta que el primero termine, momento en el cual la/s página/s se libera/n.

Nótese que si usamos **autocommit=0**, todas las instrucciones que ejecutemos serán visibles a los demás sólo cuando realice un **COMMIT**. Mientras no lo haga, las acciones de otros usuarios sobre esas páginas quedarán suspendidas.

En estas tablas no existe la opción de bloqueos para lectura o para escritura, sino que se bloquean ambas posibilidades a la vez.

INNODB

Estas tablas ofrecen bloqueos a nivel de fila. Los bloqueos a nivel fila suponen que en medio de una transacción sólo las filas afectadas por un **INSERT**, **UPDATE** o **DELETE** serán bloqueadas hasta tanto se ejecute el comando **COMMIT**.

Tienen algunas particularidades que las diferencian de las tablas **BDB** a la hora de realizar transacciones. Cuando un cliente comienza una transacción, con **BEGIN WORK**, toma una **instantánea** de los datos. Si en medio de la transacción otro cliente termina la suya y modifica algún dato, el primer cliente no verá estos cambios hasta que finalice su propia transacción, lectura consistente.

En el caso de que dos o más clientes modifiquen los mismos datos en transacciones simultáneas, persistirán los cambios realizados por la última transacción, utilizando el comando denominado **COMMIT**.



ARCHIVO DE CONFIGURACIÓN

En capítulos anteriores se recomendó guardar en un solo archivo todas las opciones de configuración y requerir este archivo mediante una de las funciones **include** o **require** provistas por PHP. Podríamos incluir en este archivo la instrucción SQL para definir el valor de la variable **autocommit**, explicada en este capítulo, dándole el valor que corresponda según lo que queramos hacer.

Si quisiéramos obtener los datos actualizados, existe la alternativa de ejecutar las sentencias **SELECT** agregando al final la opción **LOCK IN SHARE MODE** (para establecer un bloqueo compartido, ya trataremos esto más adelante).

Si otra transacción estuviera en ese momento trabajando sobre los datos, la consulta **SELECT** se detendría hasta finalizar la segunda transacción.

Estas tablas utilizan hilos, **Threads**, para implementar transacciones.

Aquí también, al igual que en las tablas **MyISAM**, se permiten bloqueos a nivel fila, tanto para lectura como para escritura.

Bloqueos

Es importante que uno de los inconvenientes de no usar transacciones es que ocurre mientras se realiza una tarea que requiere un cierto orden y los datos cambian a través de la acción de otros usuarios.

Cuando en su momento se habló de **Separación**, se dijo que las transacciones que tengan lugar simultáneamente deben ejecutarse aisladas unas de otras hasta que finalizan. Recién cuando finalizan, tengan éxito o no, son visibles para los otros usuarios. Esto resolvería el problema inicial. Pero, ¿cómo se logra solucionar el problema que ocurre cuando dos o más transacciones intentan actualizar los mismos datos? Esto supone la necesidad de bloquear la información sobre la que se está trabajando.

Las tablas **MyISAM** soportan bloqueos a nivel tabla, tanto para lectura como para escritura. Las tablas transaccionales van más allá y permiten bloquear datos en sectores específicos de las tablas de dos maneras: bloqueos exclusivos –si existe un bloqueo de este tipo sobre una tabla, los demás usuarios no podrán realizar actualizaciones **INSERT** / **UPDATE** / **DELETE** ni lecturas **SELECT** de datos–; y bloqueos compartidos –si existe un bloqueo de este tipo sobre una tabla, los demás usuarios no podrán realizar actualizaciones **INSERT** / **UPDATE** / **DELETE** pero sí lecturas **SELECT** de datos.

En las tablas transaccionales estos bloqueos son realizados internamente por el motor de MySQL y no se necesita casi ninguna acción por parte de los usuarios. Esto supone la gran ventaja de no tener que programar manualmente los chequeos de error y las acciones pertinentes a partir de los mismos, algo que seguramente habrán experimentado o experimentan los usuarios de las tablas ISAM o las MyISAM.

* SOLO O ACOMPAÑADO

Uno de los requisitos básicos para implementar transacciones es que la aplicación en cuestión sea utilizada por más de un usuario al mismo tiempo, y que éstos accedan a los mismos datos. Esto no sólo parece totalmente normal, sino que cuesta creer que exista la situación inversa. Recordemos que cada aplicación tiene un fin específico y tal situación puede darse perfectamente.

Bloqueos en INNODB: FOR UPDATE

Ahora pensemos el ejemplo y supongamos que el campo **cod_a** puede admitir duplicados. Buscamos el máximo de una tabla y luego insertamos un registro.

Tabla Alumnos	
cod_a	nom_a
1	Julian
2	Victor
3	Ema
4	Marta

```
//conexión 1
begin work;
select max(cod_a) from alumnos;

//-----> 4
```

```
//conexión 2
begin work;
select max(cod_a) from alumnos;

//-----> 4
```

```
//conexión 1
insert into alumnos values (5, "Jacobo");
```

```
//conexión 2
insert into alumnos values (5, "Florencia");
```

III PRUEBAS

Debemos saber que en éste y otros capítulos se explican determinados conceptos con ejemplos que involucran a varios clientes conectándose a una base de datos simultáneamente. Tengamos presente que podemos probar estos ejemplos en nuestra propia máquina, estableciendo dos o más conexiones desde ventanas DOS o terminales de Linux distintas.

```
//conexión 1  
commit;
```

```
//conexión 2  
commit;
```

La tabla quedaría de la siguiente manera:

Tabla Alumnos	
cod_a	nom_a
1	Julian
2	Victor
3	Ema
4	Marta
5	Jacobo
5	Florencia

Pero lo ideal hubiera sido:

Tabla Alumnos	
cod_a	nom_a
1	Julian
2	Victor
3	Ema
4	Marta
5	Jacobo
6	Florencia

Con la única finalidad de lograr esto último, utilizaremos la cláusula **FOR UPDATE** según la cual ninguna otra conexión estará posibilitada de leer o modificar datos



DOCUMENTEMOS POR FAVOR

Para tener un control sobre las situaciones en las que implementaremos transacciones, recomendamos realizar notas acerca de los contextos en los que lo haremos, luego de diseñar el sistema; es decir, documentar las transacciones. Todo esto, claro, después de estudiar cuándo amerita hacerlo y cuándo no corresponde.

hasta que la primera haya finalizado su transacción. La cláusula **FOR UPDATE** permite obtener los datos actualizados de la base. Supongamos la situación inicial tal como lo observamos en el código que se encuentra en la siguiente página:

```
//conexión 1
begin work;
select max(cod_a) from alumnos FOR UPDATE;

//-----> 4
```

```
//conexión 2
begin work;
select max(cod_a) from alumnos FOR UPDATE;

//-----> se queda esperando hasta que finalice la transaccion de la conexión 1
```

```
//conexión 1
insert into alumnos values (5, "Jacobo");
commit;
```

```
//conexión 2
//-----> 5 [resultado del select]
insert into alumnos values (6, "Florencia");
commit;
```

```
c:\wamp\bin\mysql\mysql5.5.8\bin\mysql.exe

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> begin work;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from autor for update;
+----+-----+-----+-----+-----+-----+
| cod_a | non_a | ape_a | cod_n | fec_nac_a | fec_dec_a |
+----+-----+-----+-----+-----+-----+
| 1 | julio | cortazar | 2 | 1912-11-12 | 1984-01-10 |
| 2 | jorge luis | borges | 1 | 1912-11-12 | 1912-11-12 |
| 3 | ernesto | sabato | 1 | 1912-11-12 | 1912-11-12 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Figura 4. Sintaxis de la cláusula **FOR UPDATE**.

Bloqueos en INNODB: LOCK IN SHARE MODE

Realizando un bloqueo de este tipo podemos obtener la última información disponible, y durante la transacción bloqueamos las filas que leemos, o sea las que devuelve la instrucción **select**, y sólo permitimos a los demás usuarios leer los datos.

Ejemplo de utilización:

```
select max(cod_a) from alumnos LOCK IN SHARE MODE;
```

LOCK IN SHARE MODE es similar a **FOR UPDATE** con la diferencia de que el primero permite, en una transacción a otros clientes, realizar lecturas y el segundo, no.

```

c:\wamp\bin\mysql\mysql5.5.8\bin\mysql.exe
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> begin work;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from autor lock in share mode;
+----+-----+-----+----+-----+-----+
| cod_a | nom_a | ape_a | cod_n | fec_nac_a | fec_dec_a |
+----+-----+-----+----+-----+-----+
| 1 | julio | cortazar | 2 | 1912-11-12 | 1984-01-10 |
| 2 | jorge luis | borges | 1 | 1912-11-12 | 1912-11-12 |
| 3 | ernesto | sabato | 1 | 1912-11-12 | 1912-11-12 |
+----+-----+-----+----+-----+-----+
3 rows in set (0.00 sec)

mysql> _

```

Figura 5. Sintaxis de la cláusula **LOCK IN SHARE MODE**.

... RESUMEN

En este capítulo hemos experimentado la importancia de las transacciones a partir de ejemplos que clarifican esta situación. Además, vimos cómo implementar transacciones en distintos tipos de tablas y las diferencias que existen en cuanto a cómo tratan a cada una de ellas.



TEST DE AUTOEVALUACIÓN

- 1** ¿Qué es una transacción?

- 2** ¿Qué requisitos deben cumplir una serie de instrucciones para ser consideradas todas ellas como parte de una misma transacción?

- 3** ¿Qué significa ACID?

- 4** ¿Cuándo se puede considerar a un tipo de tabla como transaccional? Nombre un tipo de tabla que entre en esta categoría y otra que no.

- 5** ¿Las transacciones son propiedad exclusiva de MySQL o existe la posibilidad de implementarlas en otras bases de datos?

- 6** ¿Qué sucede cuando la variable autocommit es igual a 1? ¿Y cuando es igual a 0? ¿Cómo podemos cambiar el valor de esta variable?

- 7** ¿Cómo sabemos cuándo empieza y termina una transacción?

- 8** Brevemente, ¿qué es un bloqueo?

EJERCICIOS PRÁCTICOS

- 1** Dé tres ejemplos de casos en los cuales para usted sería fundamental la implementación de transacciones.

- 2** Modifique el valor de la variable autocommit. Hágalo desde PHP y desde el prompt de MySQL.

Seguridad en MySQL

Hoy en día la información es la base para la toma de decisiones, de ahí su condición de imprescindible.

Como uno de los objetivos primordiales de las bases de datos es el de almacenar grandes cantidades de información, necesariamente se debe tener en cuenta el resguardo de los datos y el mantenimiento de la consistencia de los mismos.

Backup de Bases de Datos	230
BACKUP	230
MySQLDump	231
Recuperación ante fallos	235
Desde la línea de comandos	235
Desde el prompt de MySQL	236
Seguridad a nivel tabla	237
Reparación de tablas de una Base de Datos	237
Optimización de tablas	241
Introducción a la replicación	242
Seguridad física	243
Resumen	243
Actividades	244

BACKUP DE BASES DE DATOS

MySQL admite varias opciones a la hora de realizar copias de seguridad. A continuación haremos un recorrido por las más utilizadas.

BACKUP

Este comando provisto por MySQL permite hacer una copia de los archivos de las tablas sobre las cuales queremos hacer un backup.

Actualmente, sólo admite trabajar con tablas de tipo **MyISAM**.

Debemos recordar que la forma de utilizar este comando es escribiéndolo, desde el prompt de MySQL se detalla en el siguiente código:

```
mysql > backup table nombre_tabla to ruta;
```

Donde **nombre_tabla** es el nombre de la tabla (podemos ingresar más de una, separando los nombres con comas) y **ruta** es el directorio en donde se guardará la copia. Al terminar de ejecutar el comando podremos ver una tabla que nos dará distintas informaciones acerca del estado de la operación (si falló, si se realizó con éxito, advertencias, y demás datos importantes).

Habrá un registro por cada tabla que hayamos dado como argumento al comando. Este comando bloquea la tabla sobre la que se está haciendo una copia de seguridad. Aquí aparece el problema potencial que podría surgir con respecto a la consistencia de los datos, ya que cuando, por ejemplo, el comando se dispone a realizar el backup de la última tabla, las anteriores podrían haber cambiado.

Ejemplo:

```
mysql > backup table alumnos to 'c:\bak';
```



PREVISIÓN

Realizar copias de seguridad de nuestros sitios es una costumbre saludable y es recomendable hacerlo periódicamente, dependiendo de la asiduidad de las actualizaciones. No guardemos las copias en el mismo lugar en el que se encuentra el equipo. Si lo hacemos, ante desastres o robos, las copias correrían la misma suerte que la máquina.

MySQLDump

Si bien existe una gran cantidad de aplicaciones que nos permiten administrar el servidor de bases de datos y, entre otras tareas, realizar copias de seguridad de la información (también llamados *backups*, su traducción al inglés), MySQL trae consigo un comando llamado **MySQLDump**.

El aspecto de una copia de seguridad con este comando, a diferencia en este punto del comando **BACKUP** visto anteriormente, es el de un archivo de texto que contiene todas las instrucciones necesarias para restaurar una o varias bases de datos. Estas instrucciones están escritas en lenguaje **SQL** y son las ya por nosotros conocidas (**CREATE DATABASE**, **INSERT**, **DROP TABLE**, etcétera).

El archivo resultante podrá ser ejecutado sin problemas desde cualquier programa que nos permita gestionar nuestro servidor de bases de datos (**PHPMyAdmin**, **MySQL Control Center**, **la línea de comandos de MySQL**, etc.), ya que el lenguaje SQL es independiente de las aplicaciones.

Tengamos en cuenta que **mysqldump** es un programa, y para utilizarlo debemos abrir una consola en DOS o una terminal en Linux y posicionarnos en el directorio **BIN** de MySQL. Una vez allí, y estando el servidor correspondiente activo, podremos utilizar este programa de la forma:

```
C:\mysql\bin> mysqldump opciones
```

Existe infinidad de [**opciones**] a la hora de realizar copias de seguridad con **MySQLDump**, aquí trataremos algunas de ellas:

OPCIÓN	DESCRIPCIÓN
--add-locks	Mientras realiza la copia de seguridad de una tabla, la bloquea.
--add-drop-table	Inserta una sentencia DROP TABLE antes de crear cada tabla, de manera que cada vez que restauremos la base desde la copia de seguridad, borraremos las tablas existentes.
-A, --all-databases	Ambas opciones hacen lo mismo: realizar una copia de seguridad de todas las bases de datos en el servidor. Al comienzo de la copia agrega la línea: CREATE DATABASE /*!32312 IF NOT EXISTS*/ nombre_base; que sirve para crear la base si ésta no existe. ►

III SISTEMAS OPERATIVOS

Una de las razones por las cuales los sistemas operativos **Unix/Linux** son de los más frecuentemente utilizados en cuanto a servidores se refiere, es su orientación definida a mantener la seguridad del sistema: la elección de un sistema operativo también juega de manera decisiva a la hora de planificar la seguridad de un sistema informático.

OPCIÓN	DESCRIPCIÓN
--allow-keywords	Permite la creación de nombres de columnas que son palabras clave; esto se realiza poniendo de prefijo a cada nombre de columna, el nombre de la tabla.
-C, --compress	Si estamos realizando la copia desde un equipo remoto, esta opción nos permite comprimir los datos para que la comunicación sea más rápida.
-B, --databases	Para realizar copias de varias bases de datos. Al comienzo de la copia agrega la línea: CREATE DATABASE /*!32312 IF NOT EXISTS*/ nombre_base; que sirve para crear la base si ésta no existe.
-f, --force	Continúa con la copia aun cuando ocurriera algún error.
-h, --host=nombre_host	Especifica el host en donde está instalado el servidor MySQL. Por defecto es localhost.
-l, --lock-tables	Bloquea todas las tablas antes de iniciar la copia (a diferencia de --add-locks que bloquea una tabla por vez). Si está realizando copias de múltiples bases de datos, bloqueará las tablas de cada base en forma separada.
-n, --no-create-db	No agrega la línea: CREATE DATABASE /*!32312 IF NOT EXISTS*/ nombre_base; al comienzo de la copia. Esta línea crea la base si ésta no existe.
-t, --no-create-info	No realiza una copia de la estructura de las tablas.
-d, --no-data	No realiza una copia de los datos contenidos en las tablas de la base.
--single-transaction	Realiza la copia como si fuera una transacción. Sólo para tipos de tablas que soporten transacciones (MyISAM o HEAP, por ejemplo). Añadida en MySQL 4.0.2. Es mutuamente exclusiva con respecto a la opción --lock-tables.
-T, --tab=ruta	Crea un archivo de nombre nombre_tabla.sql, que contiene las instrucciones de creación, y un archivo nombre_tabla.txt, que contiene los datos de cada tabla. El formato del archivo .txt se realiza de acuerdo con las opciones --fields-xxx y --lines-xxx. Esta opción sólo funciona si el comando mysqldump se ejecuta en la misma máquina que mysqld. El usuario deberá tener permisos para crear y escribir los archivos en la ubicación especificada (ruta).
-X, --xml	Guarda una copia de la base en formato XML (eXtensible Markup Language)
-w, --where='where-condition'	Si utilizamos mysqldump sobre una sola tabla, podemos aplicar una restricción sobre los registros que queremos obtener. Ver ejemplo más adelante.

Tabla 1. Opciones de MySQLDump.

Es importante que podemos utilizar más de una opción en forma simultánea, tal como podemos apreciar en el siguiente código:

```
mysqldump --opcion1 --opcion2 ...
```

Ejemplos:

- Copia de seguridad de la tabla llamada **T1** de la base de datos **B1**. Almacenamos la salida en el archivo **tablaT.txt**. De esta manera, se resguardan tanto la estructura de la tabla como sus datos.


```
mysqldump B1 T1 > c:\windows\escritorio\tablaT.txt
```

- Copia de seguridad de las tablas **T1** y **T2** de la base de datos **B1**. Guardamos la salida en el archivo **tablaT.txt**. Se resguardan tanto la estructura de la tabla como sus datos.

```
mysqldump B1 T1 T2 > c:\windows\tablaT.txt
```

- Copia de seguridad de las tablas **T1** y **T2** de la base de datos **B1**. Guardamos la salida en el archivo **tablaT.txt**. No guardamos los datos, sólo las estructuras.

```
mysqldump --no-data B1 T1 T2 > c:\bak\tablaT.txt
```

- Copia de seguridad de la tabla llamada **T1** de la base de datos **B1**. Almacenamos la salida en el archivo denominado **tablaT.txt**. Guardamos sólo los datos, sin incluir la estructura de la tabla.

```
mysqldump --no-create-info B1 T1 > c:\tablaT.txt
```

- Lo mismo que en el ejemplo anterior, pero esta vez la base de datos **B1** está ubicada en el host llamado **servidor23**.

```
mysqldump --host=servidor23 --no-create-info B1 T1 > c:\tablaT.txt
```

- Copia de seguridad de la tabla llamada **T1** de la base de datos **B1**. Guardamos la salida en el archivo **tablaT.txt**. Se resguarda tanto la estructura de la tabla como sus datos. Sólo se toman en cuenta los registros que tengan el campo **c** mayor a 2. Equivale a hacer un backup de la consulta **select * from T1 where T1.c > 2**.

```
mysqldump -w= " c > 2" B1 T1 > c:\tablaT.txt
```

- La opción **-w** es útil también cuando queremos realizar una copia de un número concreto de registros, por ejemplo:

```
mysqldump -w= "1=1 limit 55" B1 T1 > c:\tablaT.txt
```

De esta manera, copiaríamos los primeros 55 registros de la tabla **T1** valiéndonos del operador **LIMIT** y de que la condición **1=1** se cumple siempre.

- Copia de seguridad de todas las tablas contenidos en la base de datos **B1**. Guardamos la salida en el archivo **baseB1.txt**. Se resguardan tanto las estructuras de las tablas como así también sus datos.

```
mysqldump B1 > c:\windows\escritorio\baseB1.txt
```

- Copia de seguridad de todas las bases de datos en el servidor. Se almacenan todas las bases en el archivo **basesB1.txt**. De esta manera, se resguardan tanto las estructuras de las tablas como sus datos.

```
mysqldump -A > c:\windows\escritorio\baseB1.txt
```

- Copia de seguridad de las bases de datos denominada **B1** y **B2**. Se almacenan las bases de datos en el archivo **basesB1B2.txt**. De esta forma, Se resguardan tanto las estructuras de las tablas como sus datos.

```
mysqldump -B B1 B2 > z:\pip\baseB1B2.txt
```

En los ejemplos que vimos anteriormente no se utilizó la opción **--add-drop-table**, motivo por el cual, al intentar restaurar nuestras bases o tablas desde las copias realizadas, en la mayoría de los casos obtendremos un error, debido a que MySQL intentará crear tablas con nombres ya existentes.

En cambio, en el caso de que hubiéramos usado esta opción, esto no sucedería ya que al restaurar la copia primero se borrarían las tablas y luego, se las crearía. Modifiquemos los ejemplos anteriores agregándoles la opción **--add-drop-table**.

BACKUPS INTERNACIONALES

Es importante tener en cuenta que aunque parezca extraño o poco común, existen algunas empresas –generalmente, grandes– que se encargan de contratar, alquilar o, incluso, son propietarias de servidores ubicados en países distintos de aquél en el que ellas se encuentran, para guardar allí sus copias de seguridad con mayores niveles de confianza.

RECUPERACIÓN ANTE FALLOS

Hemos visto cómo crear copias de seguridad de nuestras bases de datos y ahora veremos cómo restaurar el normal funcionamiento de nuestro sistema valiéndonos de esas copias y de algunas herramientas que nos provee MySQL.

Como se mencionó, también en secciones anteriores, se pueden restaurar copias de seguridad desde la mayoría de las aplicaciones existentes para administrar MySQL. Aquí sólo nos limitaremos a explicar aquéllas que se encuentran incluidas en la instalación básica de MySQL.

Cuando ocurre alguna irregularidad que haga fallar nuestra base de datos debemos recurrir a las copias de seguridad (que deberíamos tener hechas) y desde ellas poder solucionar el problema en forma adecuada.

Si realizamos la copia utilizando el programa **mysqldump**, tendremos disponible un archivo –normalmente se le da la extensión **.SQL**, pero no tiene importancia– y lo que necesitamos hacer ahora es remitirlo a MySQL para que lo pueda procesar. MySQL puede leer archivos de texto con instrucciones SQL de dos formas:

- Desde la línea de comandos.
- Desde el **prompt** de MySQL.

Desde la línea de comandos

De esta forma, para restaurar nuestra base de datos debemos ingresar al directorio denominado **BIN** de MySQL y escribir lo siguiente:

```
C:\> mysql nombre_base < nombre_archivo
```

Donde **nombre_base** es el nombre de la base de datos y **nombre_archivo** es la ruta al archivo que contiene las instrucciones SQL, el que fuera creado con **mysqldump**.

Por ejemplo, si queremos restaurar la base llamada **B1** cuya copia de seguridad se encuentra guardada en el archivo denominado **baseB1.txt**:

```
C:\> mysql B1 < c:\windows\escritorio\baseB1.txt
```

Algunos detalles para tener en cuenta:

- Es importante que el ejemplo anterior funcionará si tenemos creada la base **B1**. Si no es así, y tampoco tenemos la instrucción **CREATE DATABASE** en el archivo **baseB1.txt**, obtendremos un mensaje de error.

- Si tenemos lo anterior pero no tenemos las instrucciones **DROP TABLE** correspondientes a cada tabla, obtendremos un mensaje de error. Recordemos que se puede forzar la inclusión de estas instrucciones a través de la opción **--add-drop-table**.
- Si la base de datos existe, si tenemos las instrucciones **DROP TABLE** y si el archivo que contiene la copia de seguridad tiene la instrucción **USE nombre_base**, podemos restaurarla sin mencionar el nombre de la base de datos:

```
C:\> mysql < c:\windows\escritorio\baseB1.txt
```

- Podemos ingresar las distintas opciones disponibles para el programa **MySQL** (como el nombre de usuario y la contraseña):

```
C:\> mysql -u nombre -pcontraseña B1 < c:\windows\escritorio\baseB1.txt
```

Desde el prompt de MySQL

Aquí podemos utilizar el comando **source**:

```
mysql> source 'c:\windows\escritorio\baseB1.txt'
```

Si realizamos la copia de seguridad utilizando el comando **BACKUP**, podemos restaurar esa copia con el comando **RESTORE**, cuya sintaxis es la siguiente:

```
mysql > restore table nombre_tabla from ruta;
```

Donde **nombre_tabla** es el nombre de la tabla (podemos ingresar más de una separando los nombres con comas), y **ruta** es el directorio en donde se guardara la copia realizada anteriormente con el comando **BACKUP**.



¿CADA CUÁNTO TIEMPO HAY QUE HACER BACKUPS?

La periodicidad con que se realizan los backups suele ser directamente proporcional a los cambios producidos en el material que se va a resguardar. Pero esto no siempre es así, porque pueden ocurrir modificaciones que, aunque no sean muchas, sí sean importantes para el desarrollo de las actividades de la empresa en cuestión.

En caso de que tratemos de restaurar tablas ya existentes en el directorio ruta, obtendremos un error. Tanto el comando **BACKUP** como **RESTORE** trabajan exclusivamente con tablas de tipo MyISAM.

SEGURIDAD A NIVEL TABLA

Los fallos en las tablas se hacen notorios a partir de la ejecución de instrucciones SQL que devuelven errores inesperados. Los comandos para mantenimiento, reparación y optimización de tablas son esencialmente seguros, pero es una buena costumbre realizar copias de seguridad antes de ejecutarlos.

Reparación de tablas de una Base de Datos

Se recomienda usar los comandos **REPAIR TABLE** y **CHECK TABLE** por sobre **MyISAMchk**, ya que los primeros están más actualizados y tienen automatizados ciertos procedimientos que requieren más instrucciones en **MyISAMchk**.

MyISAMchk

Este programa sirve tanto para mantenimiento, reparación, optimización y obtención de información de tablas MyISAM. Los comandos **CHECK TABLE** y **REPAIR TABLE** –vis-
tos en este mismo capítulo– utilizan algunas de las funcionalidades de **MyISAMchk**.

Existe un programa llamado **ISAMchk** que se utiliza sobre tablas ISAM. Todo lo que se expresa en este capítulo acerca de **MyISAMchk** será aplicable a **ISAMchk**. Su sintaxis es:

```
myisamchk opciones archivo_tabla
```

Donde **archivo_tabla** es la ruta al archivo **.MYI** (la mayoría de los problemas ocurren en los archivos de índices). Si ejecutamos este comando desde un lugar distinto del que está alojada la base de datos, deberemos escribir la ruta completa al archivo **.MYI**. Este comando admite comodines (*), por ejemplo:

```
C:\mysql\bin> myisamchk c:\mysql\data\MIBASE\*.MYI
```

El código anterior, realiza un chequeo de todas las tablas de la base de datos **MIBASE**.

```
C:\mysql\bin> myisamchk c:\mysql\data\*\*.MYI
```

Realiza un chequeo de todas las tablas de todas las bases de datos en el servidor.

Este programa trabaja creando una copia del archivo **.MYD**, fila por fila, mientras realiza el chequeo. Cuando termina, borra el archivo **.MYD** original.

Nótese que si algún cliente está usando la tabla que deseamos chequear, recibiremos una advertencia por parte del programa. Por esta razón, es conveniente en lo posible apagar el servidor antes de ejecutar este programa.

Se puede apagar el servidor, por ejemplo, utilizando el programa **MySQLAdmin**:

```
C:\mysql\bin> mysqladmin shutdown
```

Veamos un listado de algunas de las opciones de **MyISAMchk**:

OPCIÓN	DESCRIPCIÓN
-s , --silent	No imprime nada en pantalla, salvo los errores si es que los hay.
-v , --verbose	Imprime información extra.
-c , --check	Chequea la tabla en busca de errores. Si no se le dan opciones, ésta es la función por defecto.
-i , --information	Imprime información adicional acerca de la tabla en cuestión.
-m , --medium-check	Más completo que el chequeo normal, pero menos que el extendido.
-e , --extend-check	El chequeo más completo, también el que requiere más memoria.
-T , --read-only	No marca la tabla como "chequeada".
--update	Marca la tabla como "chequeada".

Tabla 2. Opciones de reparación de **MyISAMchk**.

Es importante saber que las siguientes opciones de reparación están disponibles si se ejecuta **myisamchk** con la opción **-r** o **-o**:

```
C:\mysql\bin> myisamchk -r opciones
```

```
C:\mysql\bin> myisamchk -o opciones
```

OPCIÓN	DESCRIPCIÓN
-e , --extend-check	Trata de recuperar hasta la última fila posible. Puede que muchas de las filas recuperadas contengan datos dañados.
-r , --recover	Repara todo, excepto las filas que tienen la clave repetida y que estaban definidas como únicas.
-q , --quick	Reparación rápida que no modifica el archivo de datos. Llamado por segunda si lo modifica en caso de claves duplicadas. Siempre recrea el archivo .MYI .
-d , --description	Descripción de la tabla.

Tabla 3. Opciones de reparación de **MyISAMchk** con **-r** y **-o**.

Puede obtener un listado completo de opciones ejecutando:

```
C:\mysql\bin> myisamchk -help
```

Si no se le da ninguna opción, el programa simplemente realiza un chequeo de la tabla.

CHECK TABLE

El comando conocido como **CHECK TABLE** trabaja exclusivamente con tablas **MyISAM** e **InnoDB** y se utiliza para buscar errores en tablas.

Para el caso de tablas MyISAM, este comando es idéntico a ejecutar:

```
myisamchk -m nombre_tabla
```

Su sintaxis es la siguiente:

```
CHECK TABLE nombre_tabla opcion
```

Donde **nombre_tabla** es el nombre de la tabla sobre la que se quiere realizar un chequeo (puede definirse más de una separando sus nombres con comas), y opción toma uno de los siguientes valores:

OPCIÓN	DESCRIPCIÓN
QUICK	Sólo intenta reparar los índices.
FAST	Sólo chequea las tablas que no fueron cerradas correctamente.
CHANGED	Sólo chequea las tablas que no fueron cerradas correctamente o que cambiaron desde el último chequeo.
MEDIUM	Más rápido que con EXTENDED, pero encuentra el 99.99% de los errores. Es, sin embargo, suficiente en la mayoría de los casos.
EXTENDED	Realiza un chequeo completo que garantiza la consistencia de la tabla. Normalmente, tarda más tiempo que los demás tipos de chequeos.

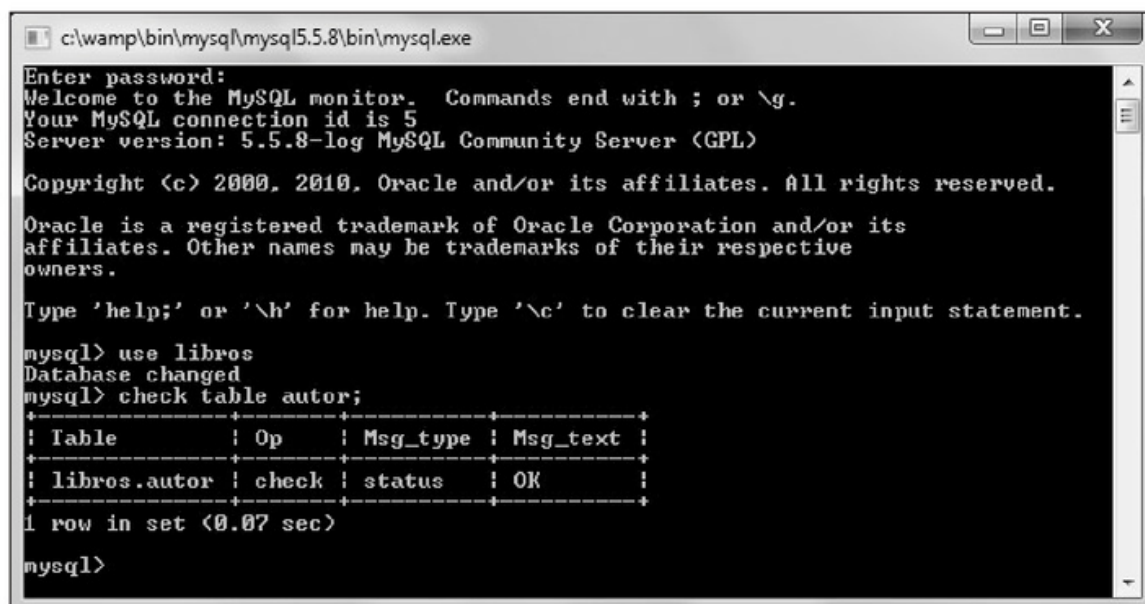
Tabla 4. Opciones de **CHECK TABLE**.

Si no se especifica la opción, se toma **MEDIUM** como valor por defecto. Puede combinar varias opciones a la vez, por ejemplo:

```
CHECK TABLE nombre_tabla FAST QUICK;
```

Si se quieren realizar chequeos rutinarios en tablas que asumimos que se encuentran en buen estado, conviene utilizar el comando, o bien, sin opciones (o sea **MEDIUM**) o con la opción **QUICK**. La opción **EXTENDED** se debería utilizar sólo en los casos en los que sospechamos, a causa del mal funcionamiento, errores.

El comando **CHECK TABLE** se encarga de devolver una tabla como la que se puede ver en la imagen que se encuentra a continuación:



```

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 5
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> check table autor;
+-----+-----+-----+-----+
| Table      | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| libros.autor | check   | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.07 sec)

mysql>

```

Figura 1. Salida del comando **CHECK TABLE**.

Si la última fila (**Msg_text**, mensaje que imprime el resultado de la operación) devuelve un valor distinto de **OK** o **Table is already up to date**, deberíamos reparar la tabla. El mensaje **OK** se encarga de avisarnos que se realizó un chequeo sobre la tabla y que no se ha encontrado ningún error.

El mensaje denominado **Table is already up to date** significa que directamente no hubo necesidad siquiera de chequear la tabla.

REPAIR TABLE

El comando **REPAIR TABLE** repara tablas posiblemente corruptas. Trabaja sólo con tablas MyISAM y es equivalente a ejecutar el comando `myisamchk -r nombre_tabla`. Su sintaxis es la siguiente:

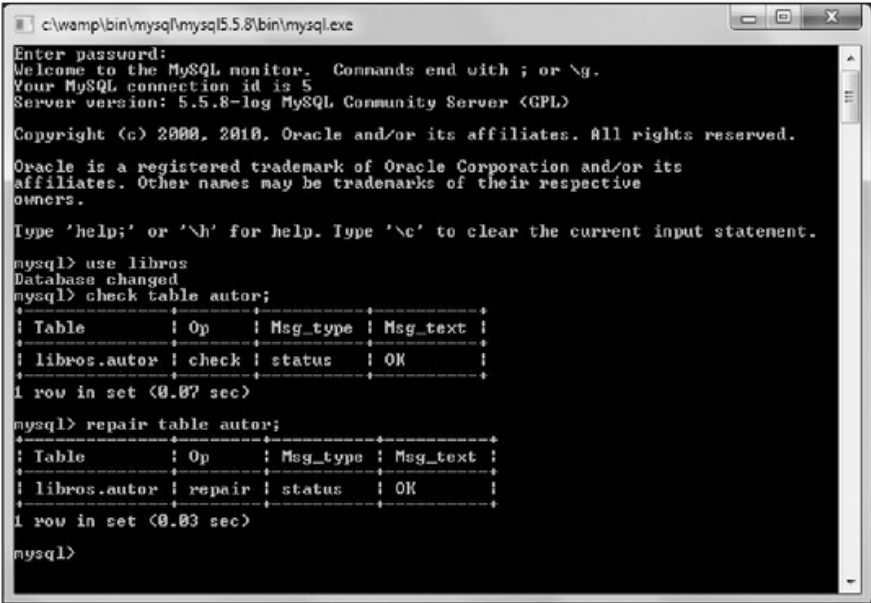
```
REPAIR TABLE nombre_tabla opcion
```

Donde **nombre_tabla** es el nombre de la tabla que se desea reparar (pueden definirse más de una, separando sus nombres con comas), y **opción** toma uno de los valores que apreciamos en la siguiente tabla:

OPCIÓN	DESCRIPCIÓN
QUICK	Intentará reparar sólo los índices.
EXTENDED	Esta opción crea el índice fila por fila y equivale a ejecutar el comando <code>myisamchk --safe-recover</code> .
USE_FRM	Disponible desde MySQL 4.0.2, esta opción se utiliza cuando el archivo <code>.MYI</code> no está disponible o está corrupto y obtiene información del archivo <code>.frm</code> para realizar la reparación.

Tabla 5. Opciones de **REPAIR TABLE**.

El comando **REPAIR TABLE** tiene como función devolver una tabla como la que se puede ver en la imagen siguiente.



```

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 5
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> check table autor;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| libros.autor | check | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.07 sec)

mysql> repair table autor;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| libros.autor | repair | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.03 sec)

mysql>

```

Figura 2. Salida del comando **REPAIR TABLE**.

La última fila devuelve el resultado de la operación, que debería ser OK. Si no lo es, debería ejecutar el comando `myisamchk -o nombre_tabla`.

Optimización de tablas

Para defragmentar los archivos de datos y eliminar el espacio que pudiere quedar después de la eliminación o actualización de registros (conviene ejecutar este comando después de borrar un gran número de filas, por ejemplo), podemos utilizar las distintas opciones que nos ofrece MySQL:

- Con **MyISAMChk**

```
myisamchk -r archivo_tabla
```

Donde `archivo_tabla` es la ruta al archivo `.MYI`.

Hallaremos más en la sección **MyISAMchk** en este mismo capítulo. Para emular el comando **OPTIMIZE TABLE**, que veremos a continuación, podemos ejecutar:

```
myisamchk --quick --check-only-changed --sort-index -analyze archivo_tabla
```

• Con **OPTIMIZE TABLE**

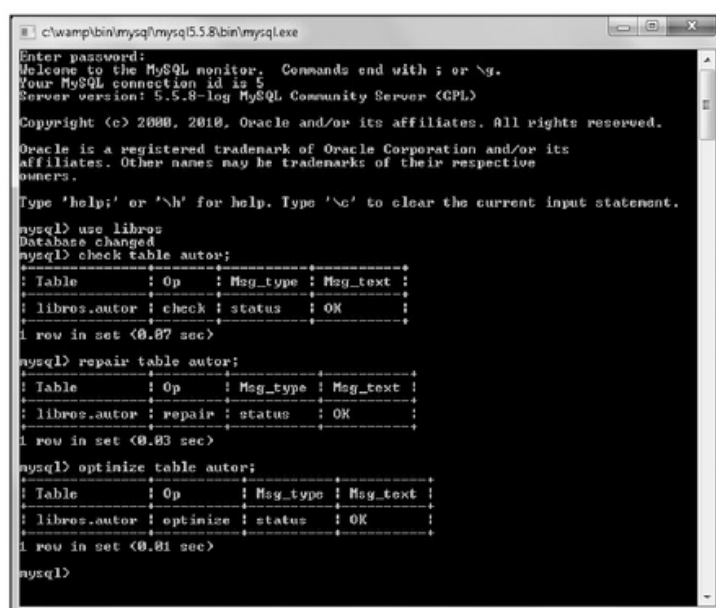
No sólo optimiza sino que repara la tabla. Su sintaxis es la siguiente:

```
OPTIMIZE TABLE nombre_tabla
```

Donde **nombre_tabla** es el nombre de la tabla que se quiere optimizar (puede definirse más de una separando sus nombres con comas).

Este comando trabaja actualmente sólo con tablas **MyISAM** y **BDB**.

La tabla que se está optimizando queda bloqueada durante el proceso.



```

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 5
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> check table autor;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| libros.autor | check | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.07 sec)

mysql> repair table autor;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| libros.autor | repair | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.03 sec)

mysql> optimize table autor;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| libros.autor | optimize | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql>

```

Figura 3. Salida del comando **OPTIMIZE TABLE**.

INTRODUCCIÓN A LA REPLICACIÓN

Existen diversas técnicas para manejar bases de datos de gran tamaño y mantener la información contenida. A continuación avanzaremos sobre una de ellas, la replicación. Esta técnica nos permite, en síntesis, realizar una copia exacta de una base de datos y alojarla en otro servidor. Cuando una de estas bases –a la que llamaremos **maestra**–, sufre algún cambio (se le insertan datos, se le modifican, se borran, se alteran las es-

estructuras de las tablas, etc.) la otra base —a la que llamaremos **esclava**— repetirá los cambios ocurridos en la otra. De esta forma ambas se mantienen sincronizadas.

La finalidad de la replicación de bases de datos no es la de realizar un simple **backup**, sino la de permitir dividir las cargas de trabajo en partes y no sólo a un servidor. Por ejemplo, podríamos derivar todas las consultas (**select**) al servidor esclavo. De esta manera se logra una mejora en el rendimiento.

MySQL nos provee ante cada nueva versión más y más mejoras con respecto a este tema central para los administradores.

SEGURIDAD FÍSICA

Cuando hablamos de seguridad física no hablamos sólo de la seguridad pertinente al hardware de un equipo, sino de la de todo un sistema informático, lo que abarca:

- Equipos informáticos (PC, Impresora, Scanner, etc.).
 - Copias de seguridad de la información.
 - Lugares en donde depositar las copias de seguridad/traslado de las mismas.
 - Mantenimiento/actualización periódica de los equipos.
- Dispositivos electrónicos.
- Redes. Cableado eléctrico/de telefonía/de redes.
- Sistema eléctrico del lugar en donde se encuentra instalado el sistema.
- Alarmas.
- Oficinas o edificios.
 - Cerraduras u otros sistemas para ingresar. Aberturas de ingreso y salida.
 - Suministro de energía eléctrica.
 - Previsión ante posibles incendios.
- Seguridad Privada. Cámaras de seguridad.
- Sistemas generadores de energía eléctrica.
- Personal habilitado a ingresar al edificio y a cada oficina.
 - Métodos de identificación segura.
 - Planes de evacuación ante la posibilidad de desastres.

... RESUMEN

Acabamos de tener un primer acercamiento al complejo y extenso tema de la seguridad en las bases de datos MySQL: cómo resguardarlas ante posibles fallos, cómo recuperarlas, cuáles son los peligros potenciales y cómo optimizarlas para obtener mejores resultados.



TEST DE AUTOEVALUACIÓN

- 1** ¿Cree que los backups son importantes aun para los usuarios finales?

- 2** ¿Cuáles serían para usted las tres medidas principales de seguridad para un sistema hogareño?

- 3** ¿Qué diferencia hay entre `myisamchk` y `check tables`?

- 4** ¿Cuáles son las formas que conoce para buscar errores en tablas?

- 5** ¿Cuáles son las formas que conoce para reparar errores en tablas?

- 6** ¿Cuáles son las formas que conoce para optimizar tablas?

- 7** Brevemente, ¿qué es la replicación de bases de datos?

- 8** ¿Recomendaría el uso de técnicas de replicación en grandes empresas? ¿Y en pequeñas? Justifique ambas respuestas.

EJERCICIOS PRÁCTICOS

- 1** Seleccione el método que prefiera y realice una copia de seguridad de una base de datos creada por usted. Luego elimine una tabla y restaure la base a partir de la copia de seguridad realizada.

- 2** Suponga que tenemos una base de datos llamada BD que contiene las tablas TA1, TA2 y TA3. ¿Cómo hacemos para -a través del programa `mysqldump`- realizar un backup sólo de la estructura de la tabla TA3? ¿Y uno de los datos y la estructura de las tablas TA1 y TA3?

Administración de MySQL

Mientras desarrollamos nuestras aplicaciones, cumplimos el rol de administrador del servidor web, del servidor de bases de datos, programador de aplicaciones, usuarios, y más. Pero cuando trabajamos en un sistema real, las tareas se vuelven más específicas. Aquí entenderemos la naturaleza y el rol de los administradores de bases de datos.

El rol del administrador de Bases de Datos	246
El rol del usuario: usuario y administradores del sistema	247
Sistema de privilegios	247
Tabla user	248
Tabla host	249
Tabla db	250
Tablas	
tables_priv y columns_priv	251
Jerarquías de acceso	252
Comandos grant y revoke	253
Gestión de privilegios con insert	257
Gestión de privilegios con select	257
Gestión de privilegios con update	257
Gestión de privilegios con delete	258
Baja de usuarios	258
mysql_change_user	258
Listar todos los usuarios mediante una consulta	258
Resumen	259
Actividades	260

EL ROL DEL ADMINISTRADOR DE BASES DE DATOS

Un **Administrador de Bases de Datos** (*DBA* por sus siglas en inglés) es la persona destinada a conocer cada detalle del servidor de bases de datos y responder a los usuarios preguntas técnicas acerca de su funcionamiento. Su tarea es tan importante y compleja que incluso puede llegar a haber más de uno por empresa.

El administrador no tiene ninguna injerencia en temas de diseño de la base (nótese que la diferencia entre una base de datos y un servidor de bases de datos radica en que un servidor puede contener varias bases).

Una de las tareas del administrador sería la de administrar usuarios –es decir crearlos, modificar sus permisos, eliminarlos.

Otra de las tareas adecuadas es mantener las bases de datos y el servidor protegido ante posibles ataques externos (de esta forma será necesario actualizar el servidor mediante parches de seguridad, firewalls, etcétera).

Otras tareas importantes:

- Instalación y actualización del servidor de bases de datos.
- Configuración del servidor (optimización, mejoras de rendimiento).
- Creación de bases de datos –aunque en algunos sistemas esto se delega a los usuarios.
- Es importante administrar los recursos correspondientes al sistema (prestar atención al crecimiento de los archivos de las bases).
- Utilizar copias de seguridad.
- Responder ante fallos en el sistema de manera rápida y segura.
- Mantenimiento de los datos.
- Monitoreo de las actividades de los usuarios. Control de acceso.
- Generar estadísticas. Generar informes.
- Probar nuevos productos que posiblemente se instalarán en el sistema.

Si bien en los gestores de bases de datos más avanzados existen herramientas de última generación que permiten (o por lo menos ayudan) a detectar, diagnosticar y resolver según el caso estas tareas de forma casi automática, la presencia y disponibilidad de un buen administrador es indispensable.



PARA TODAS LAS BASES DE DATOS

Debemos saber que si bien este libro está destinado al trabajo con bases de datos MySQL, cuando en este capítulo hablamos de las tareas fundamentales y a la vez comunes de los **Administradores de Bases de Datos**, no hacemos referencia específica a ninguna base de datos, puesto que las tareas son independientes de la elección.

EL ROL DEL USUARIO: USUARIOS Y ADMINISTRADORES DEL SISTEMA

Como se habrá notado en la sección anterior, el fin principal de la tarea de un administrador de bases de datos es **asegurar que la información esté disponible** para los usuarios en el tiempo y la forma en que éstos la necesiten.

Cuando se habla de usuarios, por lo general, nos estamos refiriendo a las personas que acceden a los servicios provistos por un sistema gestor de bases de datos a través de aplicaciones desarrolladas para tal fin. La conexión entre usuarios y administradores es con frecuencia establecida tanto en un sentido como en otro: podría ocurrir que un usuario se contactara con un administrador para pedir alguna clase de información, o que un administrador se contactara con un usuario al notar comportamientos indebidos.

SISTEMA DE PRIVILEGIOS

MySQL cuenta con un sistema de privilegios que tiene como fin principal validar la conexión de un usuario y otorgarle determinados permisos -operaciones- sobre bases de datos, tablas, y columnas.

Algunos de los permisos que pueden otorgársele a un usuario son la posibilidad de ejecutar determinadas instrucciones (**SELECT, INSERT, UPDATE, DELETE, ALTER**, otras) e incluso permitir la ejecución de algunas funciones provistas por MySQL.

Estos permisos están guardados en el mismo servidor, en una base de datos llamada **mysql**, que se crea automáticamente durante la instalación de MySQL. De esta manera, al establecerse una conexión se dan ciertos privilegios de acuerdo a la identidad de quien se conecta. Luego de establecida la conexión, cuando el usuario intente realizar operaciones que no le estén permitidas, no se le permitirá consumirlas.

Esta identidad estará definida por el nombre de usuario y la máquina desde la cual se efectúa la conexión, porque se asume que dos o más usuarios que se conectan desde distintos equipos pueden tener el mismo nombre de usuario.



EL SISTEMA DE PRIVILEGIOS Y SU COMPRENSIÓN

Es muy importante mencionar que el sistema de privilegios aquí expuesto puede provocar confusión en los primeros acercamientos; pero, si bien es complejo, también es completamente lógico y nos ofrece una amplia cantidad de opciones que nos serán de gran ayuda al simplificar un buen número de tareas en nuestro trabajo de desarrollo.

Tanto cuando un usuario intenta establecer una conexión como cuando intenta ejecutar una instrucción SQL, MySQL tomará la decisión de permitirlo o no de acuerdo a la información contenida en las tablas **DB**, **USER**, **HOST**, **tables_priv**, y **columns_priv**. A continuación, haremos un recorrido por cada una de ellas.

Tabla user

La tabla **user** contiene información sobre cada uno de los usuarios (desde qué máquina puede acceder al servidor MySQL, su clave y sus diferentes permisos).

En la siguiente tabla, podemos observar la estructura de la tabla **user** donde se detalla cada campo junto a la descripción del mismo:

CAMPO	DESCRIPCIÓN
Host	Nombre de la máquina desde la cual el usuario tiene permitido conectarse. Por ejemplo, host1.com.ar .
User	Nombre del usuario.
Password	Contraseña del usuario. Se almacena encriptada.
Select_priv	¿Permitir instrucciones SELECT?
Insert_priv	¿Permitir instrucciones INSERT?
Update_priv	¿Permitir instrucciones UPDATE?
Delete_priv	¿Permitir instrucciones DELETE?
Index_priv	¿Permitir la creación o eliminación de índices?
Alter_priv	¿Permitir la modificación de las estructuras de las tablas?
Create_priv	¿Permitir crear tablas?
Drop_priv	¿Permitir borrar tablas?
Grant_priv	¿Permitir otorgar permisos a otros usuarios?
References_priv	¿Permitir crear relaciones entre tablas?
Reload_priv	¿Permitir ejecutar los comandos reload, refresh, flush-privileges, flush-hosts, flush-logs, y flush-tables? (Permiten recargar el sistema.)
Shutdown_priv	¿Permitir ejecutar el comando shutdown? (Permite parar el servidor.)
Process_priv	¿Permitir ejecutar el comando processlist?
File_priv	¿Permitir leer y escribir archivos usando comandos como "select into outfile" y "load data infile"?

Tabla 1. Estructura de la tabla **USER**.

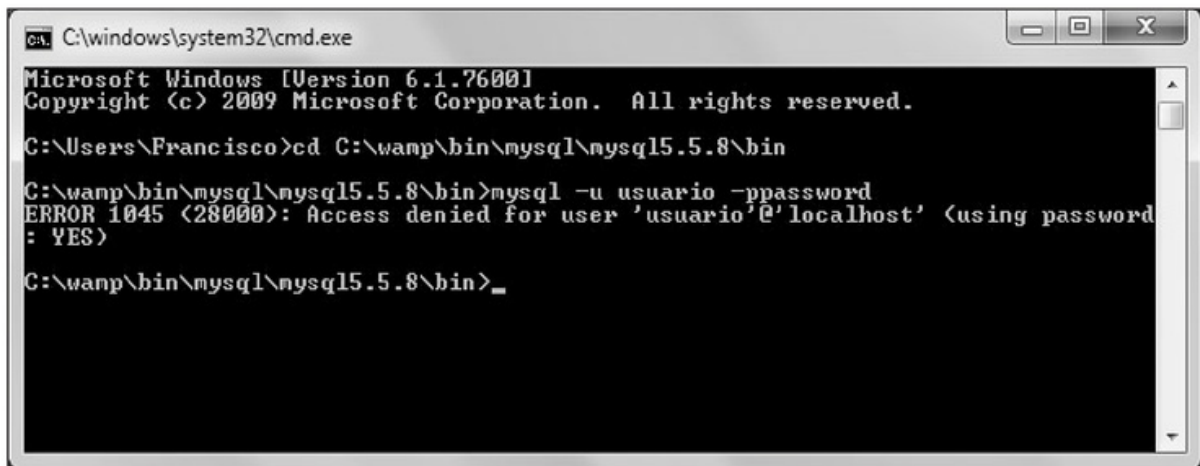


TAREAS COMUNES

La tarea de un Administrador de Bases de Datos puede parecer lejana a los usuarios de sistemas o desarrolladores de aplicaciones. Sin embargo, cuando se tiene la necesidad de administrar un servidor web –cosa no tan extraña por estas épocas– se deben poner en práctica algunas de las tareas y métodos utilizados por aquéllos.

Los campos denominados **host**, **password** y **user** admiten la cantidad de **60**, **16** y **16** caracteres respectivamente, mientras que los demás admiten sólo un carácter (recordemos que **N** para restringir, **Y** para permitir). Es importante que estos campos están definidos como de tipo llamado **ENUM**.

Es importante mencionar que en el caso de que nos encarguemos de dejar en blanco –o que pongamos el símbolo de porcentaje (%), el cual hace las veces de comodín– los campos llamados **user**, **host**, o **password**, se estará refiriendo a cualquier usuario, equipo, o contraseña, respectivamente.



```

C:\windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Francisco>cd C:\wamp\bin\mysql\mysql5.5.8\bin
C:\wamp\bin\mysql\mysql5.5.8\bin>mysql -u usuario -ppassword
ERROR 1045 (28000): Access denied for user 'usuario'@'localhost' (using password
: YES)
C:\wamp\bin\mysql\mysql5.5.8\bin>_

```

Figura 1. Mensaje de error al intentar logearse un usuario no válido.

Tabla host

Es importante mencionar que la tabla denominada **host** contiene información sobre las máquinas que podrán acceder al servidor, así como las bases de datos a que tendrán acceso, y los diferentes permisos correspondientes.

Debemos saber que la tabla denominada **host** nos permite realizar la definición de los permisos globales para máquinas con acceso a las bases de datos correspondientes al servidor. Contiene los mismos campos –a excepción del campo **user**– que la tabla **db**. En la siguiente tabla, podemos apreciar la estructura de la tabla **host** y la definición de cada uno de sus campos.

OFERTA ACADÉMICA

Debemos saber que las distintas tareas que cumple un **Administrador de Bases de Datos** –las cuales han sido desarrolladas a lo largo de este capítulo– se volvieron tan importantes gracias a los distintos avances producidos, que incluso desde hace algún tiempo, se dictan cursos y carreras para especializarse en esta área.

CAMPO	DESCRIPCIÓN
Host	Nombre de la máquina desde la cual el usuario tiene permitido conectarse. Por ejemplo, host1.com.ar .
Db	Nombre de la base de datos a la cual tiene permitido conectarse el usuario.
Select_priv	¿Permitir instrucciones SELECT?
Insert_priv	¿Permitir instrucciones INSERT?
Update_priv	¿Permitir instrucciones UPDATE?
Delete_priv	¿Permitir instrucciones DELETE?
Index_priv	¿Permitir la creación o eliminación de índices?
Alter_priv	¿Permitir la modificación de las estructuras de las tablas?
Create_priv	¿Permitir crear tablas?
Drop_priv	¿Permitir borrar tablas?
Grant_priv	¿Permitir otorgar permisos a otros usuarios?

Tabla 2. Estructura de la tabla **HOST**.

Los campos **host** y **db** admiten **60** y **64** caracteres respectivamente, mientras que los demás admiten un carácter (**N** para restringir, **Y** para permitir). Estos campos están definidos como de tipo conocido como **ENUM**.

En el caso de que se dejen en blanco (o poniendo el símbolo **%**, que hace las veces de comodín) los campos denominados **host** o **db**, se estará refiriendo a cualquier equipo o base de datos, respectivamente.

Tabla db

La tabla **db** permite especificar permisos para bases de datos individuales. Las columnas **host**, **DB**, y **USER** sirven para especificar el host desde el que puede acceder el usuario, el nombre de usuario a quien se reconoce permisos, y la base de datos a la que se van a aplicar los permisos, respectivamente.

Los permisos indicados en esta tabla sólo se aplican a la base de datos identificada en la columna **db**. En la página siguiente, podemos ver la estructura de la tabla **db** y la definición de sus campos (**Tabla 3**):



LA PARTE DE ATRÁS

Las bases de datos se han vuelto tan populares y comunes casi en cualquier ámbito que, finalmente, todos terminamos siendo clientes o usuarios de esta clase de servicios. Por esta y otras razones es de vital importancia entender cuál es el modo de funcionamiento de los sistemas de información, cuáles son sus fines, cómo intentan llegar a ellos, cuáles son las perspectivas, etcétera.

CAMPO	DESCRIPCIÓN
Host	Nombre de la máquina desde la cual el usuario tiene permitido conectarse. Por ejemplo, host1.com.ar .
Db	Nombre de la base de datos a la cual tiene permitido conectarse el usuario.
User	Nombre del usuario.
Select_priv	¿Permitir instrucciones SELECT?
Insert_priv	¿Permitir instrucciones INSERT?
Update_priv	¿Permitir instrucciones UPDATE?
Delete_priv	¿Permitir instrucciones DELETE?
Index_priv	¿Permitir la creación o eliminación de índices?
Alter_priv	¿Permitir la modificación de las estructuras de las tablas?
Create_priv	¿Permitir crear tablas?
Drop_priv	¿Permitir borrar tablas?
Grant_priv	¿Permitir otorgar permisos a otros usuarios?

Tabla 3. Estructura de la tabla *db*.

Los campos **Host**, **Db** y **User** admiten **60**, **64** y **16** caracteres respectivamente, mientras que los demás admiten un carácter (**N** para restringir, **Y** para permitir). Estos campos están definidos como de tipo **ENUM**. En el caso de que se dejen en blanco –o poniendo el símbolo %, que hace las veces de comodín– los campos **user**, **host**, o **db**, se estará refiriendo a cualquier usuario, equipo, o base de datos, respectivamente.

Tablas *tables_priv* y *columns_priv*

Las tablas **db** –tratadas anteriormente–, **tables_priv**, **columns_priv** nos proveen de un control individual de las bases de datos, tablas y columnas. **tables_priv** permite especificar permisos para tablas concretas dentro de unas bases de datos:

CAMPO	DESCRIPCIÓN
Host	Nombre de la máquina desde la cual se intenta conectar el usuario. Por ejemplo, host1.com.ar .
Db	Nombre de la base de datos a la cual se intenta conectar el usuario.
User	Nombre del usuario.
Table_name	Nombre de la tabla sobre la cual se aplicarán las siguientes restricciones.
Table_priv	Aquí se ubican, separadas por coma, las operaciones permitidas sobre la tabla <i>Table_name</i> al usuario <i>User</i> que se conecta desde la máquina <i>Host</i> . Las operaciones disponibles son las siguientes: Select, Insert, Update, Delete, Create, Drop, Grant, References, Index, y Alter.
Column_priv	Aquí se ubican, separadas por coma, las operaciones permitidas sobre la columna <i>Column_priv</i> (que pertenece a la tabla <i>Table_name</i> , de la base <i>Db</i>) al usuario <i>User</i> que se conecta desde la máquina <i>Host</i> . Las operaciones disponibles son las siguientes: Select, Insert, Update y References.
Timestamp	Valor timestamp correspondiente a la fecha de inserción del registro.
Grantor	Nombre de la persona que ha concedido los permisos para modificar la tabla.

Tabla 4. La tabla *tables_priv* contiene los siguientes campos.

La tabla **columns_priv** permite especificar permisos para columnas concretas en el interior de una tabla. Contiene los siguientes campos:

CAMPO	DESCRIPCIÓN
Host	Nombre de la máquina desde la cual se intenta conectar el usuario. Por ejemplo, host1.com.ar .
Db	Nombre de la base de datos a la cual se intenta conectar el usuario.
User	Nombre del usuario.
Table_name	Nombre de la tabla que contiene la columna.
Column_name	Nombre de la columna.
Column_priv	Aquí se ubican, separadas por coma, las operaciones permitidas sobre la columna Column_priv (que pertenece a la tabla Table_name, de la base Db) al usuario User que se conecta desde la máquina Host. Las operaciones disponibles son las siguientes: Select, Insert, Update y References.
Timestamp	Valor timestamp correspondiente a la fecha de inserción del registro.

Tabla 5. Campos de la tabla **columns_priv**.

Jerarquías de acceso

Como se dijo anteriormente, al recibir un intento de conexión **MySQL** compara los datos recibidos con los que figuran en la tabla **USER**. Si hay coincidencia –si en esta tabla hay una fila que contenga el mismo **User**, **Password** y **Host**–se autoriza el acceso.

Ante cada instrucción SQL generada por un usuario conectado, **MySQL** vuelve a consultar la tabla **USER**, esta vez para verificar si tiene permisos para ejecutar esa instrucción. Si los tiene, la instrucción se ejecuta; si no los tiene se pasa a consultar la tabla **db** para comprobar si ese usuario tiene permisos específicos sobre la base de datos en cuestión. Si los tiene, la instrucción se ejecuta; si no los tiene se pasa a consultar las tablas **tables_priv** y **columns_priv**. Si allí encuentra los permisos, la acción se ejecuta; si no **MySQL** devuelve un mensaje error.

Repasando los campos de las tablas vemos que algunos se repiten: ¿qué pasaría si incurriéramos en supuestas contradicciones? Por ejemplo, podría ocurrir que denegáramos un permiso a un usuario en la tabla **USER** y se lo habilitáramos en la tabla **db**. Lo que ocurre es que los permisos de la tabla **USER** tienen alcance global, pero no particular. Entonces, en esta situación el usuario no contaría con ese permiso en ningun-

SEGURIDAD

Cuando desarrollamos en PHP, nos encontramos con una dificultad al distribuir aplicaciones: el código fuente podrá ser visto por las personas que administren el servidor. Esta situación puede tener mucha importancia en algunos casos, y una manera de solucionarla está dada por la existencia de programas –normalmente pagos– que nos permiten encriptar el código.

na base de datos, salvo en las bases en que lo habilitamos, que figuran en la tabla **DB**. Asimismo, un usuario habilitado en la tabla **USER** no podrá acceder a una base de datos del servidor desde una máquina si ésta no está habilitada en la tabla **HOST**.

Comandos grant y revoke

Podemos dar o modificar privilegios a los usuarios de la forma anterior; o sea, a través de la modificación de tablas, o a través de los comandos **grant** y **revoke**.

A continuación, daremos una descripción de estos comandos.

Comando grant

Sirve tanto para crear nuevos usuarios como para modificar sus permisos.

```
mysql> grant select, insert on *.* to lucas@'%' identified by 'blablabla';
```

En el ejemplo anterior se da la siguiente situación:

- **Permisos:** ejecutar instrucciones **select** e **insert** (**grant select, insert**).
- **Sobre:** cualquier tabla de cualquier base de datos (***.***).
- **Para el usuario:** lucas (**to lucas**).
- **Que se conecta desde:** cualquier máquina (**@'%'**).
- **Cuyo password es:** blablabla (**identified by 'blablabla'**).

Para dar ciertos permisos sobre una base de datos en particular, podríamos escribir:

```
mysql> grant select, insert on nombre_base.* to lucas@'%'
identified by 'blablabla';
```

En cambio, si quisiéramos otorgar ciertos permisos sobre una tabla de una base de datos en particular, deberíamos escribir:

HÁGALO USTED MISMO

Establecer un nivel de jerarquías en cuanto al acceso de los usuarios a una base de datos es de vital importancia; pero podría ser un buen ejercicio trasladar esos niveles a nuestra aplicación, y permitir distintas actividades a los usuarios de nuestro sistema, al otorgarles permisos para realizar una u otra tarea. Para implementarlo podemos basarnos en la forma de trabajar de MySQL.


```
mysql> grant select, insert on nombre_base.nombre_tabla to lucas@'%'
identified by 'blablabla';
```

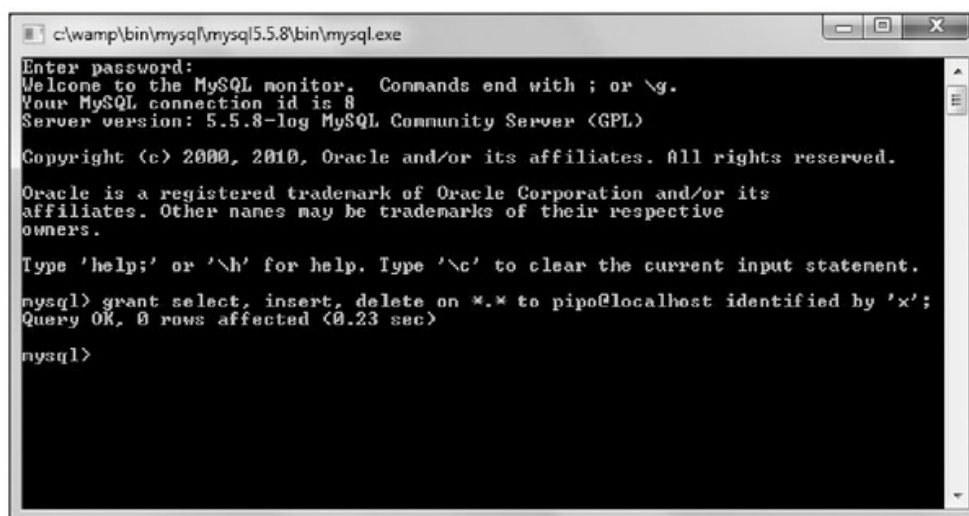


Figura 5. Ejemplo de uso del comando **GRANT**. Alta del usuario *pipo*.

En el caso de que quisiéramos otorgar ciertos permisos sobre una tabla de una base de datos en particular para un usuario que se conecta desde una máquina específica, podríamos escribir lo siguiente:

```
mysql> grant select, insert on nombre_base.nombre_tabla to
lucas@nombre_host identified by 'blablabla';
```

Pueden darse permisos a usuarios de manera tal que se puedan conectar sin informar su password –la parte **identified by** es opcional.

También pueden darse permisos sobre columnas determinadas:

```
mysql> grant select (col1, col2, col3) on nombre_base.nombre_tabla to
lucas@nombre_host identified by 'blablabla';
```

III DBAs

MySQL brinda a los administradores de bases de datos la posibilidad de diplomarse para ampliar sus conocimientos según los requisitos propios de la empresa. Estas certificaciones son de gran valor para muchos clientes que precisan verificar la idoneidad de las personas encargadas de sus sistemas. Podemos obtener información en www.facebook.com/mysqlcertifiedprofessional.

Donde **col1**, **col2**, y **col3** son columnas pertenecientes a la tabla **nombre_tabla**.

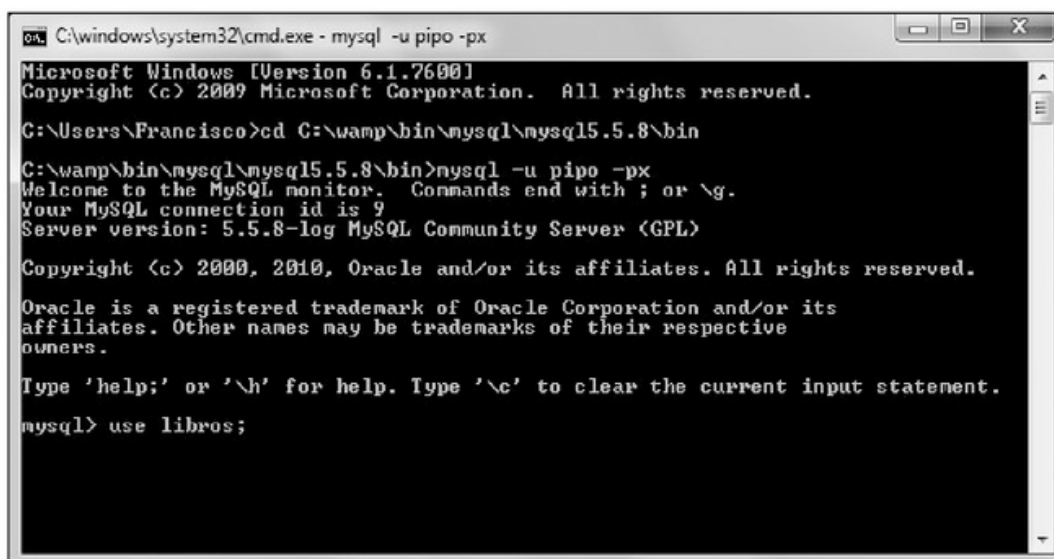


Figura 6. Usuario **pipo** accediendo al sistema.

En los ejemplos anteriores se dan los privilegios **select** e **insert**, pero existen algunos más. Algunos de ellos pueden verse a continuación:

PRIVILEGIO	DESCRIPCIÓN
ALL	Permite al usuario obtener todos los privilegios, excepto grant option. Ver al final de la tabla.
ALTER	Permite el uso de la instrucción ALTER TABLE.
CREATE	Permite el uso de la instrucción CREATE TABLE.
CREATE TEMPORARY TABLES	Permite el uso de la instrucción CREATE TEMPORARY TABLE.
DELETE	Permite el uso de la instrucción DELETE.
DROP	Permite el uso de la instrucción DROP TABLE.
EXECUTE	Permite ejecutar Procedimientos Almacenados (MySQL 5.0).
FILE	Permite el uso de los comandos SELECT ... INTO OUTFILE y LOAD DATA INFILE.
INDEX	Permite el uso de la instrucción CREATE INDEX y DROP INDEX.
INSERT	Permite el uso de la instrucción INSERT.
LOCK TABLES	Permite el uso de la instrucción LOCK TABLES.
PROCESS	Permite el uso de la instrucción SHOW FULL PROCESSLIST.
RELOAD	Permite el uso de la instrucción FLUSH.
SELECT	Permite el uso de la instrucción SELECT.
SHOW DATABASES	Permite el uso de la instrucción SHOW DATABASES.
SHUTDOWN	Permite el uso de la instrucción MYSQLADMIN SHUTDOWN.
UPDATE	Permite el uso de la instrucción UPDATE.
USAGE	Sin privilegios. Sólo permite establecer la conexión.
GRANT OPTION	Permite al usuario dar permisos a otros usuarios.

Tabla 6. Instrucciones para manipular bases de datos.

Tengamos en cuenta que para poder ejecutar el comando **grant**, se deben contar con determinados permisos especiales; por ejemplo, el usuario **ROOT** dispone de ellos. Podemos ingresar al sistema como **ROOT** y agregar usuarios o modificar sus permisos, utilizando para tal fin el comando **grant**.

Comando revoke

La sintaxis y forma de utilización del comando **revoke** es similar a **grant**, pero en lugar de otorgar privilegios, los quita. Por ejemplo:

```
mysql> REVOKE ALL ON *.* FROM jason@'%';
```

Quita todos los privilegios sobre todas las bases de datos disponibles en el servidor al usuario **jason**, desde donde sea que se conecte.

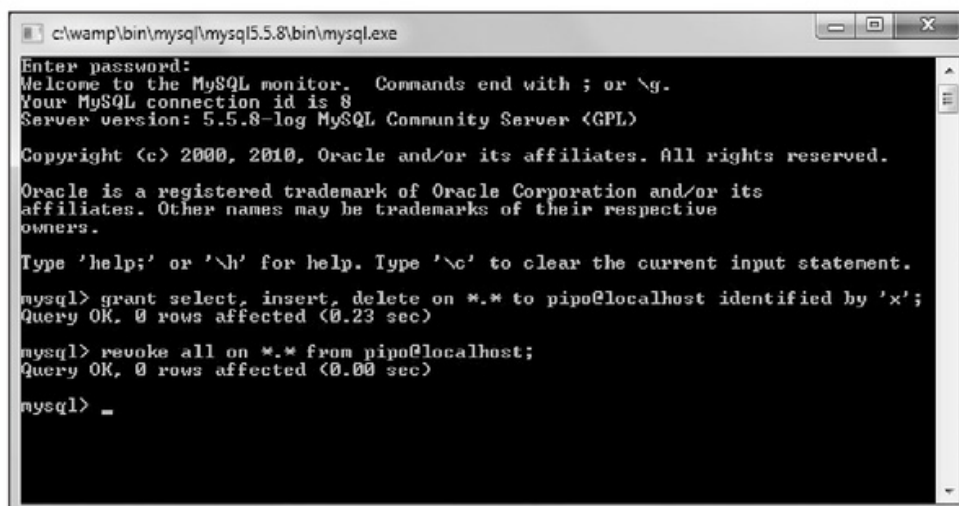


Figura 7. Eliminación del usuario pipo a través de **REVOKE**.

Gestión de privilegios con INSERT

En el siguiente ejemplo, otorgamos al usuario **thomas** –contraseña **blablabla**– el privilegio de insertar datos en la tabla **productos** de la base de datos **alimentos**.



MANTENERSE AL TANTO

Las tablas destinadas por MySQL al control de acceso de usuarios (tanto para el uso de algunas instrucciones SQL como para conectarse desde una máquina u otra) son implementaciones que se respetan a través de las diferentes versiones, pero que, a la vez, se actualizan y reciben más funcionalidades. Por eso, es importante informarse sobre los cambios producidos en esta área.

Si el usuario no existe, será creado:

```
mysql> grant insert on alimentos.productos to thomas@'%'  
identified by 'blablabla';
```

Gestión de privilegios con SELECT

En el ejemplo que se encuentra a continuación, le hemos dado al usuario **rick** con contraseña **blablabla**, que se conecta a través del host llamado **server1.com**, el privilegio de ejecutar instrucciones **select** en la columna **desc_alimentos** de la tabla tipo de la base de datos **alimentos**. Si el usuario no existe, será creado:

```
mysql> grant select (desc_alimentos) on alimentos.tipo to rick@server1.com  
identified by 'blablabla';
```

Gestión de privilegios con UPDATE

Por medio del código que está situado debajo, le hemos otorgado al usuario **jeff** y contraseña **blablabla**, que se conecta desde el host llamado **server1.com**, el privilegio de ejecutar instrucciones **update** en la totalidad de las bases de datos que se encuentran disponibles en el servidor. Si el usuario no existe, será creado:

```
mysql> grant update on *.* to jeff@server1.com identified by 'blablabla';
```

Gestión de privilegios con DELETE

En este ejemplo damos al usuario **peter**, que se podrá conectar desde cualquier host sin especificar su **password**, el privilegio de ejecutar instrucciones **delete** en todas las bases de datos disponibles en el servidor.

* INGRESO RESTRICTO

Implementar en los sistemas un área para el control y el seguimiento de las actividades de los usuarios que ingresan en nuestro sitio puede ser interesante desde el punto de vista estadístico y económico, y también lo será en el aspecto referido a la seguridad: por ejemplo, podríamos evitar el acceso al sitio a usuarios conflictivos o que intenten realizar algún daño.

Si el usuario no existe, será creado:

```
mysql> grant DELETE on *.* to peter@'%';
```

Baja de usuarios

Se puede quitar usuarios borrando las entradas correspondientes en la tabla **user** o bien limitando sus privilegios a través del comando **revoke**, visto anteriormente en este mismo capítulo. Utilizando **revoke** no se eliminará al usuario, pero se logrará que se mantenga totalmente inoperante e inofensivo.

mysql_change_user

Cuando nos conectamos desde PHP a una base de datos MySQL tenemos que utilizar alguna de las funciones provistas para ello (**Capítulo 4**) y darle como argumento, entre otros, el nombre de usuario y su password.

A propósito, PHP incorpora una función que nos permite, una vez establecida la conexión, cambiar de usuario, password e incluso base de datos sin cerrar la conexión.

Esta función es **mysql_change_user** y tiene la siguiente sintaxis:

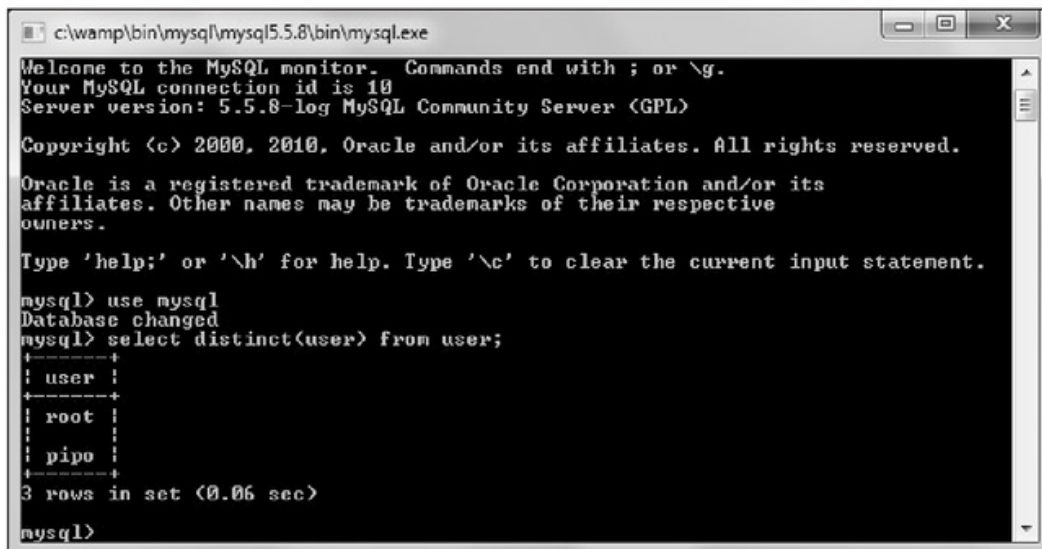
```
mysql_change_user ("nombre de usuario", "password", "base de datos",  
    identificador);
```

Los argumentos **base de datos** e **identificador** son opcionales. Si se especifica **base de datos**, ésta será la base por defecto luego del cambio de usuario. Se especifica **identificador** en el caso de que tenga más de una conexión abierta, para indicar cuál de ellas desea modificar (si especifica **identificador** debe indicar sí o sí la **base de datos**). Si la combinación final no es válida (o sea que el usuario es incorrecto, o que no se corresponde con el password, o que no existe la base de datos, u otro) la función devuelve falso y se mantiene la conexión anterior.

Listar todos los usuarios mediante una consulta

Simplemente tenemos que obtener los usuarios de la tabla denominada **user**, que figuran justamente en la columna llamada **user**:

```
mysql> select distinct(user) from user;
```



```
c:\wamp\bin\mysql\mysql5.5.8\bin\mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
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 mysql
Database changed
mysql> select distinct(user) from user;
+-----+
| user |
+-----+
| root |
| pipe |
+-----+
3 rows in set (0.06 sec)

mysql>
```

Figura 8. Salida de la consulta para listar usuarios.

Con esto, hemos finalizado con el últimos de los temas de este libro referido a la administración de MySQL. A continuación, encontrará tres apéndices que le permitirán profundizar los conocimientos vistos hasta aquí.

... RESUMEN

Acabamos de hacer una introducción al complejo e interesante mundo de los Administradores de Bases de Datos: sus tareas, sus funciones, su relación con el entorno, cómo realizan sus actividades, y demás. Para esto utilizamos las funciones y los comandos provistos tanto por PHP como por MySQL. Una de las tareas fundamentales del administrador es, justamente, administrar o gestionar lo referente a los usuarios, tema predominante de este capítulo.



TEST DE AUTOEVALUACIÓN

- 1** ¿Qué diferencia a los comandos GRANT y REVOKE?

- 2** Nombre cuatro tareas clásicas de un administrador de bases de datos.

- 3** ¿En qué consiste el llamado “sistema de privilegios”?

- 4** ¿Para que sirven básicamente las tablas db, host y user? ¿Qué datos contienen estas tablas en su sistema? Responda esta última pregunta a través de consultas SQL tanto desde PHP como desde el prompt de MySQL.

- 5** ¿Es posible crear dos usuarios con el mismo nombre y distintas contraseñas? Haga la prueba.

- 6** ¿Es posible crear dos usuarios con distintos nombres y contraseñas iguales? Haga la prueba.

- 7** ¿En qué casos cree que se podría aplicar la función `mysql_change_user`?

EJERCICIOS PRÁCTICOS

- 1** Ingrese al servidor como usuario ROOT y dé de alta a un usuario llamado PIPO que tenga permiso para realizar consultas SELECT sobre todas las base de datos. Dele una contraseña y permítale acceder sólo desde el host local (localhost).

- 2** Conéctese a una base de datos como usuario PIPO e intente borrar una tabla.

- 3** Ahora permítale a PIPO realizar, además de consultas SELECT, ejecutar instrucciones DROP.

- 4** Conéctese a una base de datos como usuario PIPO y borre una tabla.

- 5** ¿Hay en su servidor algún usuario con permiso para acceder a algún dato cuyo nombre sea Marty? Responda a esta pregunta mediante una consulta SQL.

Capas de abstracción para acceso a datos

Existen actualmente varias alternativas de software, como PHPLib y Metabase, que nos permiten trabajar con capas de abstracción para acceso a datos.

A través de este capítulo nos preocuparemos de conocer las dos más extendidas en uso: MDB2 y ADOdb.

ARQUITECTURA Y FUNCIONAMIENTO

Si bien PHP provee extensiones para trabajar con un gran número de bases de datos, rescribir el código para que nuestra aplicación trabaje con otra puede convertirse fácilmente en una tarea muy engorrosa (sin hablar del tiempo que nos llevaría) porque deberíamos modificar los nombres de las funciones, verificar si los parámetros son correctos (los argumentos que se le dan a la función para conectarse a una base Oracle pueden ser distintos en tipo y número a los que se utilizan en otras bases, por ejemplo), e incluso verificar la sintaxis de nuestras instrucciones SQL, que pueden no ser soportadas por algunos motores.

Hasta aquí los problemas. La solución consiste en utilizar clases que nos permitan acceder a distintas bases de datos utilizando el mismo código (con una serie de funciones comunes para todas ellas), lo que es igual a contar con una portabilidad de las aplicaciones que puede evitarnos un dolor de cabeza en el futuro.

Muchos desarrolladores recomiendan usar este tipo de opciones, aun cuando en principio no se sepa con seguridad que se va a tener que migrar de base de datos.

En un principio se tiende a pensar que utilizar una capa de abstracción para acceder a una base de datos resulta muy lento, pero no es así. Evidentemente, la velocidad disminuye, pero para la mayoría de los usuarios este retardo es menos que invisible.

INTRODUCCIÓN A MDB2

Pear es, en realidad, un conjunto de clases y extensiones (también llamadas **paquetes**) diseñadas para propósitos diversos. Uno de esos propósitos son las bases de datos y uno de los paquetes destinados para ello se llama específicamente **MDB2**.

Puede descargar éste y otros paquetes más la documentación, desde el sitio oficial de Pear, el cual se encuentra en la dirección **<http://pear.php.net>**.

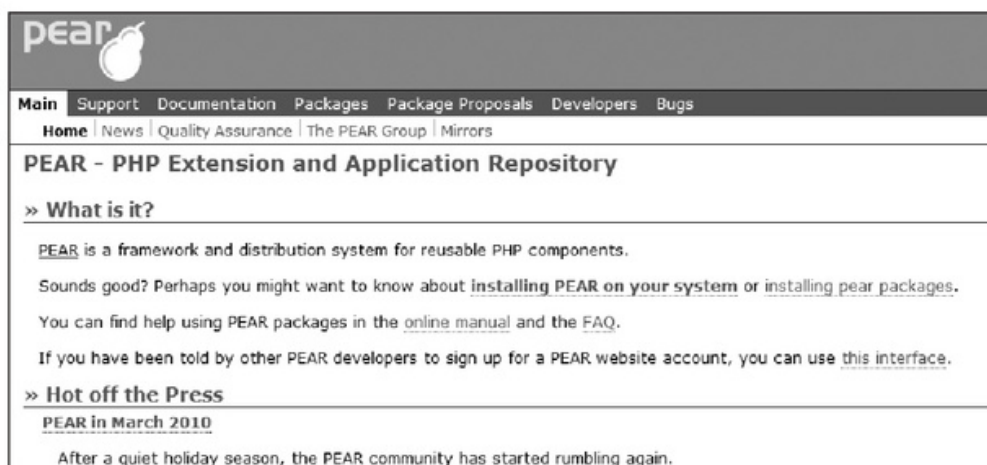


Figura 1. Sitio oficial de Pear.

Pear está supervisada por los desarrolladores de PHP, lo que le da un toque **oficial**.

Se podría dividir el proyecto según en qué lenguaje estén desarrolladas las clases o extensiones que lo conforman: las que están escritas en lenguaje **C/C++** se incluyen en **PEAR** y las que están escritas en **PHP** se incluyen en **PECL**.

Luego de instalar PHP en su sistema podrá ver que se ha creado dentro del directorio de instalación una carpeta llamada justamente **Pear**. Esto sucede porque PHP instala por defecto (a menos que haya configurado PHP con la opción **—without-pear** desde un sistema Linux compatible) los paquetes base de la estructura de Pear.

Ahora, lo que necesitamos saber es cómo agregar más paquetes. Pear nos ofrece un administrador para realizar esta tarea que viene instalado por defecto en las versiones de PHP superiores a la 4.2.x.

Debemos tener en cuenta que si no posee una de estas versiones deberá descargar desde Internet el administrador, de la siguiente manera:

- Para sistemas Unix/Linux/BSD

```
$ lynx -source http://pear.php.net/go-pear | php
```

Importante: algunas distribuciones utilizan el navegador **links**.

- Para sistemas Windows

Dentro del directorio de instalación de PHP, podremos encontrar un archivo llamado **go-pear.bat**. Al ejecutarlo, el mismo programa se encargará de hacernos preguntas referidas al sistema, para luego realizar la instalación.

```
C:\windows\system32\cmd.exe - php go-pear.php
C:\wamp\bin\php\php5.3.5>php go-pear.php
PHP Deprecated: Assigning the return value of new by reference is deprecated in
C:\wamp\bin\php\php5.3.5\go-pear.php on line 725

Deprecated: Assigning the return value of new by reference is deprecated in C:\w
amp\bin\php\php5.3.5\go-pear.php on line 725

PHP Deprecated: Function ereg() is deprecated in C:\wamp\bin\php\php5.3.5\go-pe
ar.php on line 1406
PHP Stack trace:
PHP 1. (main):() C:\wamp\bin\php\php5.3.5\go-pear.php:0
PHP 2. detect_install_dirs() C:\wamp\bin\php\php5.3.5\go-pear.php:162

Deprecated: Function ereg() is deprecated in C:\wamp\bin\php\php5.3.5\go-pear.ph
p on line 1406
Call Stack:
   0.0020   1841112   1. (main):() C:\wamp\bin\php\php5.3.5\go-pear.php:0
   0.1112   1845600   2. detect_install_dirs() C:\wamp\bin\php\php5.3.5\go-pea
r.php:162

Welcome to go-pear!

Go-pear will install the 'pear' command and all the files needed by
it. This command is your tool for PEAR installation and maintenance.

Use 'php go-pear.php local' to install a local copy of PEAR.

Go-pear also lets you download and install the following optional PEAR
packages: PEAR_Frontend_Web_beta, PEAR_Frontend_Gtk2, HDB2.

If you wish to abort, press Control-C now, or press Enter to continue: _
```

Figura 2. Inicio de *go-pear.bat*.

En el caso de que tengamos que actualizar el administrador, deberemos acceder desde el navegador a la dirección **<http://pear.php.net/go-pear>**. Se verá que aparece un código PHP: lo guardamos como **go-pear.php**.

Por ultimo, desde el **prompt** de Windows escribiremos:

```
C:\php> php c:\ruta\al\archivo\go-pear.php
```

Con lo cual se iniciará el proceso de actualización del administrador.

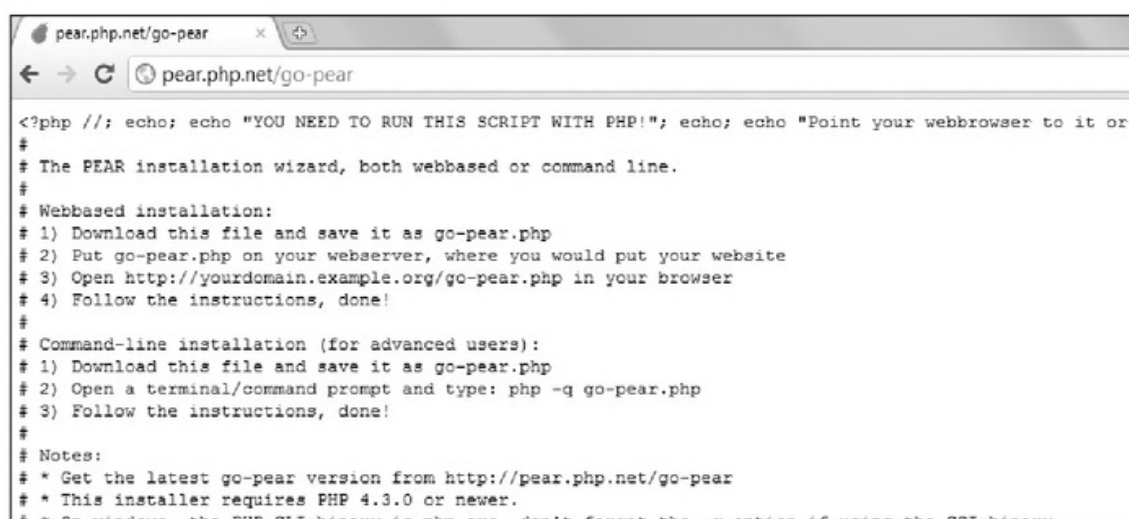


Figura 3. Vista del sitio **go-pear**.

Una vez instalado el administrador, podemos empezar a descargar los paquetes que necesitemos, y una de las formas más fáciles de hacer esto es mediante la línea de comandos del sistema tipeando lo siguiente:

```
pear install nombre_paquete
```

Donde **nombre_paquete** es el nombre del paquete que se quiere instalar. Para saber qué paquetes hay disponibles, se puede ingresar al sitio de Pear o ejecutar el comando:

```
pear remote-list
```

Puede ocurrir también que se haya descargado, a través de un navegador, por ejemplo, el paquete en formato **.TGZ**. Para instalarlo en nuestro sistema deberemos ingresar:

```
pear install nombre_archivo.tgz
```

Es conveniente agregar el PATH de Pear (ruta a la carpeta en donde está instalado) al archivo **php.ini**. Nos debería quedar algo por el estilo:

```
include_path = ".;c:\php\includes;c:\php\pear"
```

MDB2 soporta, hasta el momento, las siguientes bases:

- MySQL
- PostgreSQL
- Oracle
- Frontbase
- Querysim
- Interbase/Firebird (PHP version 5)
- MSSQL
- SQLite

Debemos saber que MDB2 surge como el reemplazo de Pear::Db, incorporando mejoras en funcionalidades y en estabilidad. Para instalar el paquete MDB2, debemos ejecutar la siguiente línea desde una consola:

```
pear install MDB2
```

Figura 4. MDB2 permite acceder a distintas bases de datos emulando las características no disponibles de cada una.

Este es solo el primer paso, ahora deberemos instalar las extensiones necesarias para interactuar con cada gestor de bases de datos en particular (por ejemplo MySQL):


```
pear install MDB2#mysql
```

Otros ejemplos:

```
pear install MDB2#pgsql
```

```
pear install MDB2#sqlite
```

Para mas informacion acerca de PostgreSQL y SQLite podemos visitar sus sitios web oficiales: <http://www.postgresql.org> y <http://www.sqlite.org>.

Para establecer una conexión, necesitaremos crear un **DSN** (*Data Source Name*), que es una cadena de caracteres compuesta por varias partes:

```
gestor://usuario:contraseña@servidor/base
```

En donde, **gestor** puede tomar uno de los siguientes valores:

CONSTANTE	BASE DE DATOS
fbsql	FrontBase
ibase	InterBase / Firebird
mssql	Microsoft SQL Server
mysql	MySQL
mysqli	MySQL (para versiones 4.1 y superiores)
oci8	Oracle 7/8/9/10
pgsql	PostgreSQL
querysim	QuerySim
sqlite	SQLite 2

Tabla 1. Constantes de las bases de datos soportadas.

El DSN creado nos servira para establecer la conexion a traves de uno de los metodos disponibles (**factory**, **connect** o **singleton**):

```
<?php
$driver = "mysql";
```

```

$hostname = "localhost";
$database = "nombre_base";
$username = "root";
$password = "";

$dsn = "$driver://$username:$password@$hostname/$database";

$con = MDB2::factory($dsn);

if(PEAR::isError($con)) {
    die($con->getMessage());
}

$con->disconnect();

?>

```

Es importante saber que en caso de que alguno de los parámetros que han sido ingresados para generar el DSN contenga caracteres especiales, estos deberán ser reemplazados según lo que vemos en la siguiente tabla:

CARÁCTER	REEMPLAZO
:	%3a
+	%2b
?	%3f
/	%2f
(	%28
=	%3d
@	%40
)	%29
&	%26

Tabla 2. Caracteres delimitadores en MDB2.

Existen dos métodos para ejecutar instrucciones SQL: **query** (para aquellas que podrían llegar a devolver filas, por ejemplo, **select**) y **exec** (para las que no, por ejemplo, **insert**, **delete**, **update**, **create**, etcétera).

Es importante destacar que el método denominado **query** recibe como argumento la instrucción entregada y se encarga de devolver un objeto **MDB2_Result**, si todo funcionó bien, o un objeto **MDB2_Error**, si ocurrió un error.

```
$res = $mdb2->query('SELECT * FROM vendedores');

if (PEAR::isError($res)) {
    die($res->getMessage());
}

echo 'Se ejecuto correctamente la instruccion';

?>
```

El método **exec** recibe como argumento la instrucción y devuelve un objeto **MDB2_Result**, si todo funcionó bien, o un entero que representa el número de filas afectadas (actualizadas, eliminadas, agregadas, etcétera).

```
<?php

require_once 'MDB2.php';

$mdb2 = MDB2::connect('mysql://root@localhost/ventas');

if (PEAR::isError($mdb2)) {
    die($mdb2->getMessage());
}

$res = $mdb2->exec("INSERT INTO vendedores (apellidoVendedor,
nombreVendedor) VALUES ('A', 'B')");

if (PEAR::isError($res)) {
    die($res->getMessage());
}

echo "$res fila/s afectada/s";

?>
```

Así como en la extensión **mysql** contamos con la función **mysql_real_escape_string**, **MDB2** provee el método **quote** que recibe cuatro argumentos:

- El valor por escapar (obligatorio).

- El tipo de dato correspondiente (opcional).
- Si encerrar entre comillas los valores o no (opcional).
- Si escapar los caracteres `_` y `%` o no (opcional).

```
$mdb2->quote($valor, 'integer', false, false);
```

```
$mdb2->quote($valor, 'text', true, false);
```

```
$mdb2->quote(date("Y-m-d"), 'date', true, false);
```

Ademas de los vistos anteriormente, contamos con los siguientes tipos de datos para incluir en este metodo: **float**, **decimal**, **timestamp**, **time**, **boolean**, **blob**, y **clob**.

Debemos saber que para manejar los resultados obtenidos a partir de un objeto **MDB2_Result** existen cuatro métodos disponibles:

- **fetchOne**: devuelve en cada llamada el primer campo de la fila de resultados:

```
$res = $mdb2->query("SELECT apellidoVendedor FROM vendedores");

if (PEAR::isError($res)) {
    die($res->getMessage());
}

while (($apellidoVendedor = $res->fetchOne())) {
    echo "<li>".$apellidoVendedor;
}
```

- **fetchRow**: retorna la fila completa de resultados:

```
$res = $mdb2->query("SELECT nombreVendedor, apellidoVendedor FROM
vendedores");

if (PEAR::isError($res)) {
    die($res->getMessage());
}
```

```
}
```

```
while (($row = $res->fetchRow())) {
    echo "<li>".$row[0].", ".$row[1];
}
```

- **fetchCol**: se encarga de regresar un array que contiene valores de la primera columna del resultado, como vemos a continuación:

```
$res = $mdb2->query("SELECT nombreVendedor, apellidoVendedor FROM
vendedores");

if (PEAR::isError($res)) {
    die($res->getMessage());
}

echo '<pre>';
print_r($res->fetchCol());
echo '</pre>';
```

- **fetchAll**: reintegra un array con **todas** las filas del resultado:

```
$res = $mdb2->query("SELECT * FROM vendedores");

if (PEAR::isError($res)) {
    die($res->getMessage());
}

echo '<pre>';
print_r($res->fetchAll());
echo '</pre>';
```

Podemos definir el tipo de acceso a los campos contenidos en una fila de resultados a partir de las constantes que vemos a continuación:

CONSTANTE	DESCRIPCIÓN
MDB2_FETCHMODE_ORDERED	Valor por defecto, índices numéricos.
MDB2_FETCHMODE_ASSOC	Índices asociativos, nombres de columnas.
MDB2_FETCHMODE_OBJECT	Objetos, cada columna es una propiedad de la fila devuelta.

Tabla 3. Constantes para acceder a valores de filas.

```
$res = $mdb2->query("SELECT nombreVendedor, apellidoVendedor FROM
vendedores");

if (PEAR::isError($res)) {
    die($res->getMessage());
}

while (($row = $res->fetchRow(MDB2_FETCHMODE_ASSOC))) {
    echo "<li>".$row[nombrevendedor];
```

Podemos definir un método por defecto a través de **setFetchMode**:

```
$mdb2->setFetchMode(MDB2_FETCHMODE_ASSOC);

$res = $mdb2->query("SELECT nombreVendedor, apellidoVendedor FROM
vendedores");

if (PEAR::isError($res)) {
    die($res->getMessage());
}

while (($row = $res->fetchRow())) {
    echo "<li>".$row[nombrevendedor];
}
```

Los objetos **MDB2_Result** poseen otros métodos útiles:

MÉTODO	DESCRIPCIÓN
numRows	Número de filas devueltas.
numCols	Número de columnas devueltas.
rowCount	Fila actual.
getColumnNames	Nombres de las columnas devueltas.

Tabla 4. Algunos de los métodos disponibles para *MDB2_Result*.

Para simular el **LIMIT** de MySQL contamos con el método **setLimit**:

```
$mdb2->setLimit(10, 5);
$res = $mdb2->query("SELECT * FROM nombreTabla WHERE campo1 > campo2");
```


Para implementar consultas preparadas contamos con **prepare** (se encarga de preparar la instrucción y también obtener un objeto de tipo **MDB2_Statement_Common**) y también con **execute** (para ejecutar la consulta deseada).

```
$tipos = array('text', 'text');

$scp = $mdb2->prepare('INSERT INTO vendedores (apellidoVendedor,
nombreVendedor) VALUES (?, ?)', $tipos, MDB2_PREPARE_MANIP);

$scp->execute(array('x', 'x'));
$scp->execute(array('y', 'y'));
$scp->execute(array('z', 'z'));
```

El método **prepare** recibe como argumentos la instrucción, un array que contiene los tipos de datos de cada campo por reemplazar en el SQL (o **true** para autodetectar), y una constante: para las instrucciones que devuelvan resultados, debemos utilizar **MDB2_PREPARE_RESULT**, mientras que para las demás **MDB2_PREPARE_MANIP**.

El método **execute**, por su parte, recibe un array con los datos por ingresar en la instrucción SQL. Devuelve como resultado el número de filas afectadas.

En el caso de las instrucciones que devuelven resultados, **prepare** puede recibir como tercer argumento, en lugar de **MDB2_PREPARE_MANIP**, un array con los tipos de datos devueltos (uno por columna) o **true** para autodetectar:

```
$scp = $mdb2->prepare('SELECT apellidoVendedor, nombreVendedor FROM
vendedores WHERE idVendedor = ?', array('integer'), array('text', 'text'));

$res = $scp->execute(3);
$res = $scp->execute(2);
$res = $scp->execute(4);
```

Para tratar los resultados obtenidos, contamos con métodos como los siguientes: **getOne**, **getRow**, **getCol**, **getAll** y **getAssoc**, los cuales nos permiten, a través del módulo **Extended**, recuperar filas y columnas:

```
$mdb2->loadModule('Extended');
```

```
$res = $mdb2->extended->getRow('SELECT * FROM vendedores WHERE idVendedor =
?', null, array(1), array('integer'));

echo '<pre>';
print_r($res);
echo '</pre>';
```

Podemos ejecutar más de una instrucción SQL al mismo tiempo a través del método **executeMultiple**, disponible si se habilita el módulo **Extended** de MDB2:

```
$mdb2->loadModule('Extended', null, false);

$datos[] = array('x', 'x');
$datos[] = array('y', 'y');
$datos[] = array('z', 'z');

$cp = $mdb2->prepare('INSERT INTO vendedores VALUES (?, ?)');
$mdb2->extended->executeMultiple($cp, $datos);
```

Debemos saber que estas clases de instrucciones se ejecutan como transacciones, es decir que si alguna falla, las demás se anulan.

Mediante los métodos **autoPrepare** y **autoExecute**, es posible ejecutar instrucciones **INSERT**, **UPDATE**, **DELETE** y **SELECT** sin escribirlas de manera completa, como tradicionalmente hacemos. Nuevamente, deberemos incluir el módulo **Extended**:

INSERT

```
$mdb2->loadModule('Extended');

$campos = array('idVendedor', 'apellidoVendedor', 'nombreVendedor');
$tipos = array('integer', 'text', 'text');
$valores = array(10, 'x', 'x');

$cp = $mdb2->extended->autoPrepare('vendedores', $campos,
MDB2_AUTOQUERY_INSERT, null, $tipos);
$res = $cp->execute($valores);
```

UPDATE

```
$mdb2->loadModule('Extended');

$campos = array('apellidoVendedor', 'nombreVendedor');
$tipos = array('text', 'text');
$valores = array('y', 'y');

$cp = $mdb2->extended->autoPrepare('vendedores', $campos,
MDB2_AUTOQUERY_UPDATE, 'idVendedor = 10', $tipos);
$res = $cp->execute($valores);
```

DELETE

```
$mdb2->loadModule('Extended');

$cp = $mdb2->extended->autoPrepare('vendedores', null,
MDB2_AUTOQUERY_DELETE, 'idVendedor = 10');
$res = $cp->execute();
```

Debemos saber que para probar el funcionamiento tanto de ADOdb (que veremos a continuación) como de MDB2 utilizando una base de datos que no sea MySQL, deberemos tenerla instalada y funcionando, además de verificar que dicha base esté entre las soportadas. También, recordemos que las conexiones varían incluso entre diferentes versiones de una misma base.

INTRODUCCIÓN A ADOdb

Si alguna vez utilizamos **ASP** para acceder a bases de datos veremos que existen muchos puntos de contacto con la sintaxis utilizada por **ADOdb**.

Se puede descargar la última versión desde <http://sourceforge.net/projects/adodb>.

ADOdb (*Active Data Objects DataBase*) funciona solo con PHP versión 4.0.4 o superiores y para realizar la instalación sólo tenemos que extraer todos los archivos (que vienen comprimidos en formato **.ZIP** o **.TGZ**) a una carpeta que pueda ser accesible desde un navegador (o sea dentro del servidor web).

Una vez hecho esto podremos usar todas las funciones provistas por ADOdb incluyendo el archivo **adodb.inc.php** en nuestras páginas:

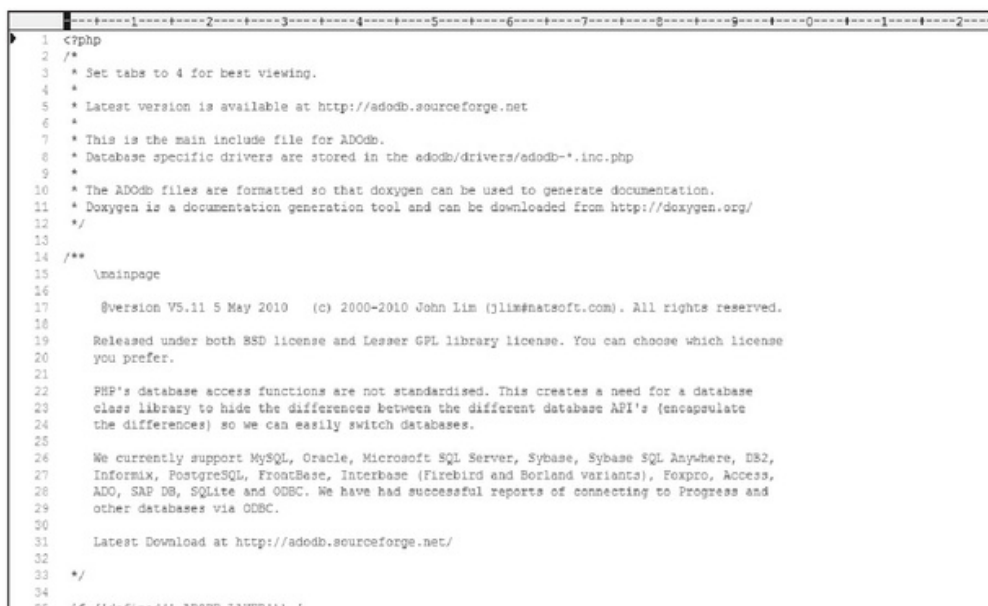

```
include('adodb/adodb.inc.php');
```

En el ejemplo se supone que descomprimos los archivos a una carpeta llamada **adodb** que está en el mismo lugar que nuestro script.

Este archivo contiene las directivas de configuración básicas y la definición de algunas clases que forman ADOdb para conectarse a una base de datos. Las más importantes se encuentran en la sección **CLASS ADOConnection** (en donde justamente se define la clase **ADOConnection**) y son las siguientes:

```
var $database = ''; //nombre de la base
var $host = ''; //nombre del servidor
var $user = ''; //nombre del usuario
var $password = ''; //contraseña del usuario
```

Debemos tener en cuenta que más adelante veremos que estos valores pueden configurarse también desde las llamadas a algunas funciones.



```

1 <?php
2 /*
3  * Set tabs to 4 for best viewing.
4  *
5  * Latest version is available at http://adodb.sourceforge.net
6  *
7  * This is the main include file for ADOdb.
8  * Database specific drivers are stored in the adodb/drivers/adodb-*.inc.php
9  *
10 * The ADOdb files are formatted so that doxygen can be used to generate documentation.
11 * Doxygen is a documentation generation tool and can be downloaded from http://doxygen.org/
12 */
13
14 /**
15  \mainpage
16
17  @version V5.11 5 May 2010 (c) 2000-2010 John Lim (jlim@natsoft.com). All rights reserved.
18
19  Released under both BSD license and Lesser GPL library license. You can choose which license
20  you prefer.
21
22  PHP's database access functions are not standardised. This creates a need for a database
23  class library to hide the differences between the different database API's (encapsulate
24  the differences) so we can easily switch databases.
25
26  We currently support MySQL, Oracle, Microsoft SQL Server, Sybase, Sybase SQL Anywhere, DB2,
27  Informix, PostgreSQL, FrontBase, Interbase (Firebird and Borland variants), Foxpro, Access,
28  ADO, SAP DB, SQLite and ODBC. We have had successful reports of connecting to Progress and
29  other databases via ODBC.
30
31  Latest Download at http://adodb.sourceforge.net/
32
33 */
34
35 #if (defined('ADODB_TRACE'))

```

Figura 7. Archivo de configuración *adodb.inc.php*.

ADOdb soporta trabajar actualmente, entre otras, con las siguientes bases de datos: MySQL, PostgreSQL, Oracle, Interbase, Firebird, DB2, Microsoft SQL Server, Microsoft Access, Microsoft Visual FoxPro, SQLITE (sólo con PHP5), Sybase, Informix, y acceso a través de ODBC.

Podemos obtener más información relevante acerca de **ADOdb** en su sitio en Internet, <http://adodb.sourceforge.net>.

Antes de establecer una conexión (ya veremos cómo) se debe especificar el sistema de bases de datos que vamos a utilizar:

```
$db = ADONewConnection($database);
```

Debemos saber que el argumento denominado **\$database** puede encargarse de contener los valores que detallamos a continuación:

VALOR	PARA ACCEDER A BASES DE DATOS
access	Microsoft Access
ado	ADO genéricas
ado_access	Microsoft Access utilizando ADO
ado_mssql	Microsoft SQL Server utilizando ADO
db2	DB2
vfp	Microsoft Visual FoxPro
fbsql	FrontBase
ibase	Interbase versión 6 o anteriores
firebird	Firebird / Interbase
informix72	Informix versión 7.3 o anteriores
informix	Informix
maxsql	MySQL 4.1 o superiores
mssql	Microsoft SQL Server 7
mssqlpo	mssql con driver portable
mysql	MySQL 4.0 o inferiores
mysqli	MySQL 4.1 o superiores
mysqlt	MySQL 4.1 o superiores (igual a maxmysql)
oracle	Oracle 7
oci8	Oracle 8/9
oci805	Oracle 8.0.5
oci8po	Oracle 8/9 con driver portable
odbc	ODBC genéricas
odbc_mssql	MSSQL vía ODBC
odbc_oracle	Oracle via ODBC
postgres	PostgreSQL
postgres64	PostgreSQL 6.4
postgres7	PostgreSQL 7 (actualmente igual a postgres)
sqlanywhere	Sybase SQL Anywhere
sqlite	SQLite
sybase	Sybase

Tabla 5. Opciones disponibles para el método `ADONewConnection`.

A continuación, nos encargamos de incluir una referencia de las funciones más importantes que nos ofrece ADOdb:

Connect

Sirve para establecer una conexión:

```
connect("servidor", "nombre de usuario", "password", "nombre_base");
```

Donde:

- **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 en donde se encuentra instalado el servidor de bases de datos que utilizaremos. Si estamos trabajando en forma local podemos emplear como nombre de servidor **localhost** ó **127.0.0.1** como dirección IP.
- **Nombre de usuario:** este dato corresponde al nombre de usuario válido para acceder a la base de datos que estamos utilizando.
- **Password:** se trata de la contraseña correspondiente al nombre de usuario que ha sido ingresado en el paso anterior, debemos ingresarla en forma exacta.
- **Base de Datos:** nombre de la base de datos que queremos seleccionar y sobre la cual queremos realizar las consultas.

Esta función devuelve **verdadero** si logramos conectarnos y **falso** si algo falló. Para establecer conexiones persistentes utilizamos la función **pconnect**.

IsConnected

Devuelve verdadero si se logró establecer la conexión.

Execute

Ejecuta una instrucción SQL. Su sintaxis es:

```
execute("instruccion");
```

SelectLimit

Nos sirve para emular la función **LIMIT** provista por MySQL pero no soportada por otras bases de datos. Su sintaxis es la siguiente:

```
selectLimit("instruccion", cantidad, desde);
```

Debemos tener en cuenta que sólo sirve para consultas tipo **SELECT** y devuelve **cantidad** registros a partir del registro **desde**.

Prepare

Es importante saber que en el caso de que la base de datos soporte consultas, esta función permite sacar provecho de esta posibilidad.

GetOne

Se encarga de recuperar el primer campo de la primera fila devuelta por la consulta **instruccion**. Su sintaxis es la siguiente:

GetOne("instruccion");

AffectedRows

Devuelve el número de filas afectadas por la última sentencia **UPDATE** o **DELETE**.

RecordCount

Devuelve el número de filas devueltas por la última sentencia **SELECT**.

Fields("nombre_columna")

Devuelve el valor contenido en la columna **nombre_columna**. Es sensible a mayúsculas y minúsculas. Puede también usar corchetes:

Fields["nombre_columna"]

FetchRow

Se encarga de devuelve un array que contiene la fila actual (o **falso** si no hay más filas). A continuación vemos un ejemplo:

```
<?php

$res = $db->Execute($sql);
if ($res)
    while ($fila = $res->FetchRow()) {
        ....
        ....
    }

?>
```

FetchNextObject

Trata a la fila devuelta como si fuera un objeto y avanza a la próxima automáticamente (devuelve falso si no hay más filas). Por ejemplo:

```

<?php

$rs = $db->Execute('select a, b from tabla1');
if ($rs) {
    while ($obj = $rs->FetchNextObject()) {
        print $obj->A;
        print $obj->B;
    }
}

?

```

Nótese que esta función traduce a mayúsculas los nombres de los campos. Si no quiere que esto suceda utilice la función **FetchNextObj** que es similar.

GetRowAssoc

Devuelve una array que contiene los valores de los campos de la fila devuelta utilizando como índice el nombre de la columna.

Para mover el cursor a la siguiente fila se usa la función **MoveNext**.

MoveNext

Se posiciona en la siguiente fila. Hay veces en las que no es necesario utilizar este tipo de funciones (como en **FetchNextObject**).

MoveFirst

Similar a **MoveNext**, sólo que se posiciona en la primera fila.

MoveLast

Similar a **MoveNext**, sólo que se posiciona en la última fila.

Move(n)

Similar a **MoveNext**, sólo que se encarga de posicionar en la fila que le indiquemos (la fila cuyo número hayamos definido con **n**).

Debemos tener en cuenta que ADOdb pone a disposición de los usuarios una gran cantidad de funciones para, entre otras cosas, manejo de errores, manejo de cadenas de caracteres, BLOBs, fechas, transacciones, y muchas más.

Es importante saber que para obtener un listado completo de todas las funciones provistas por ADOdb, podemos revisar el manual que viene junto con el programa, dentro de la carpeta denominada **docs**.

Para terminar, veamos el siguiente código de ejemplo de página que trabaja con ADOdb, en él se muestran los conceptos que hemos analizado hasta aquí:

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
    <title>Ejemplo ADOdb</title>
</head>
<body>

<?php

if (!include("./adodb/adodb.inc.php"))        {
    echo "No encuentro el archivo adodb.inc.php !";
    exit;
}

$server = '127.0.0.1';
$user = 'jeepers';
$pass = 'creepers';
$database = 'ss';

$driver = 'mysql';

$db = ADONewConnection($driver);

//mostrar mensajes (de error, por ejemplo) por pantalla
$db->debug = true;

$db->Connect($server, $user, $pass, $database);
$rs = $db->Execute('select * from tabla1');

for ($c=0; $c<$rs->RecordCount(); $c++)      {
    print $rs->fields("nom_1").'<br />';
    $rs->MoveNext();
}

?>
```



```
</body>
</html>
```

A continuación, implementaremos una pequeña aplicación, utilizando ADOdb, para que podamos apreciar y poner en práctica lo visto en esta parte del Anexo. Utilizaremos una base de datos MySQL, pero claro está que podríamos haber usado cualquier otra, siempre y cuando fuera soportada por Pear.

La idea de la aplicación es la que sigue: guardaremos en la base de datos información suficiente para que los usuarios que acceden a ella puedan saber qué películas están en cartel cerca de su zona de residencia. Para ello definimos la siguiente estructura:

```
drop table if exists cine;

create table cine (
  idcine int not null,
  nomcine varchar(100),
  dircine varchar(100),
  idzona smallint not null,
  primary key (idcine)
);

drop table if exists zona;

create table zona (
  idzona smallint not null,

  desczona varchar(100),
  primary key (idzona)
);

drop table if exists pelicula;

create table pelicula (
  idpelicula int not null,
  nompelicula varchar(100),
  duracion int,
  primary key (idpelicula)
);
```

```
drop table if exists pelicula_cine;

create table pelicula_cine (
  idcine int not null,
  idpelicula int not null,
  horario datetime not null,
  primary key (idcine, idpelicula, horario)
);

insert into zona values (1, 'zona 1');
insert into zona values (2, 'zona 2');
insert into zona values (3, 'zona 3');

insert into pelicula value (1, 'casablanca', 110);
insert into pelicula value (2, 'lo que el viento se llevo', 120);
insert into pelicula value (3, 'el halcon maltes', 115);

insert into cine values (1, 'america', 'paso 578', 1);
insert into cine values (2, 'arte', 'piedras 78', 2);

insert into pelicula_cine values (1, 1, '2005-04-27 22:35:00');
insert into pelicula_cine values (1, 2, '2005-04-27 19:35:00');
insert into pelicula_cine values (2, 1, '2005-04-28 22:35:00');
insert into pelicula_cine values (2, 3, '2005-04-28
```

El siguiente es el código del archivo denominado **buscador.php**:

```
<?php

include('adodb/adodb.inc.php');

$dbdriver = 'mysql';
$db = ADONewConnection($dbdriver);
$db->Connect('localhost', 'root', '', 'adodb');

?>
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
```

```

<head>
    <title>Buscador con ADOdb!</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>

<?php

if ($_POST[buscar]) {
    $sql = 'select * from pelicula, pelicula_cine, cine, zona where
pelicula.idpelicula = pelicula_cine.idpelicula and pelicula_cine.idcine =
cine.idcine and zona.idzona = cine.idzona';

    if ($_POST[zona])
        $sql .= ' and cine.idzona = '.$_POST[zona];

    if ($_POST[pelicula])
        $sql .= ' and pelicula.idpelicula = '.$_POST[pelicula];

```



```

$rs = $db->Execute($sql);

echo '<table cellpadding="4" cellspacing="0">';
echo '<tr>';
echo '<td>Pelicula</td>';
echo '<td>Cine</td>';
echo '<td>Direccion</td>';
echo '<td>Zona</td>';
echo '<td>Horario</td>';
echo '<td>Duracion</td>';
echo '</tr>';

if ($rc = $rs->RecordCount()) {
    for ($c=0; $c<$rc; $c++) {
        $horario = date("d/m/Y H:i", strtotime($rs->fields("horario"))).' Hs.';

        echo '<tr>';
        echo '<td>'.$rs->fields("nompelicula").'</td>';
        echo '<td>'.$rs->fields("nomcine").'</td>';
        echo '<td>'.$rs->fields("dircine").'</td>';
        echo '<td>'.$rs->fields("desczona").'</td>';
        echo '<td>'.$horario.'</td>';
        echo '<td>'.$rs->fields("duracion").'</td>';
        echo '</tr>';
        $rs->MoveNext();
    }
} else {

    echo '<tr>';
    echo '<td colspan="6">No Se Encontraron resultados !</td>';
    echo '</tr>';
}

echo '</table>';

} else {

    echo '<form method="post">';

```

```

echo '<table cellpadding="4" cellspacing="0">';
echo '<tr>';
echo '<td>Seleccione Zona de Residencia:</td>';
echo '<td>';
echo '<select id="zona" name="zona">';
echo '<option value="0">Seleccione Zona</option>';

$rs = $db->Execute('select * from zona');
if ($rs) {
    while ($fila = $rs->FetchRow())
        echo '<option value="'. $fila[0]. '">'. $fila[1]. '</option>';
}

echo '</select>';
echo '</td>';
echo '</tr>';

echo '<tr>';
echo '<td>Seleccione Pelicula:</td>';
echo '<td>';
echo '<select id="pelicula" name="pelicula">';
echo '<option value="0">Seleccione Pelicula</option>';

$rs = $db->Execute('select * from pelicula');
if ($rs) {
    while ($fila = $rs->FetchRow())
        echo '<option value="'. $fila[0]. '">'. $fila[1]. '</option>';
}

echo '</select>';
echo '</td>';
echo '</tr>';

echo '<tr>';
echo '<td colspan="2"><input type="submit" id="buscar" name="buscar"
value="Buscar"></td>';
echo '</tr>';

```

```

        echo '</table>';
        echo '</form>';
    }

?>

</body>
</html>

```

Buscador con ADOdb!

localhost/10/adodb/buscador.php

Seleccione Zona de Residencia: Seleccione Zona

Seleccione Pelicula: Seleccione Pelicula

Buscar

Figura 9. *Buscador.php: inicio.*

Buscador con ADOdb!

localhost/10/adodb/buscador.php

Pelicula	Cine	Direccion	Zona	Horario	Duracion
casablanca	america	paso 578	zona 1	27/04/2005 22:35 Hs.	110
lo que el viento se llevo	america	paso 578	zona 1	27/04/2005 19:35 Hs.	120
casablanca	arte	piedras 78	zona 2	28/04/2005 22:35 Hs.	110
el halcon maltes	arte	piedras 78	zona 2	28/04/2005 19:05 Hs.	115

Figura 10. *Buscador.php: resultados de la búsqueda.*

Ejemplo práctico

Tengamos en cuenta que el ejemplo de este capítulo intenta mostrar la aplicación práctica de los temas del libro. Esto significa que es una excusa, ya que seguramente se nos ocurrirán variadas ideas acerca de lo que se debería modificar, de lo que está bien, de lo que está mal, y más. Sin dudas, será beneficioso tomar como base este ejemplo para realizarle modificaciones y ampliar su funcionalidad.

Aplicación completa: discos	288
cnx.php	294
index.php	295
buscador.php	296
login.php	298
verdiscos.php	300
carrito.php	305
Menú de opciones	311
Dar de alta discos	312
Dar de baja discos	315
Modificar los datos de los discos	317
Dar de alta categorías	321
Dar de baja categorías	323
Modificar los datos de las categorías	325
Dar de alta intérpretes	327
Dar de baja intérpretes	329
Modificar los datos de los intérpretes	331
Dar de alta sellos	333
Dar de baja sellos	335
Modificar los datos de los sellos	337
Ventas por mes	339
Usuarios más activos	341

APLICACIÓN COMPLETA: DISCOS

A continuación, se encuentra un listado de las tablas que forman parte de la base de datos para utilizar en el ejemplo, con sus respectivas estructuras.

Debemos tener presente que con el signo # podemos acceder a realizar la representación de las claves primarias, y con **FK**, las claves foráneas:

Tabla Disco				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_d	Int	6	Clave única que identifica a cada disco.
	nom_d	Varchar	80	Nombre de disco.
FK	cod_i	Int	11	Código que identifica a cada intérprete.
	cant_d	Int	11	Cantidad de discos en stock.
FK	cod_s	Smallint	3	Código del sello discográfico.
	cod_cat	Smallint	3	Código de categoría.
	precio_d	Float	5.2	Precio del disco.
	fec_d	Date	aaaa-mm-dd	Fecha de ingreso del disco.

Tabla 1. Estructura de la tabla *Disco*.

Tabla Pedidos_m				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_p	Int	11	Clave única que identifica a cada pedido.
FK	cod_u	Int	6	Clave única que identifica a cada usuario.
	fec_p	Datetime	dd-mm-aaaa	Fecha en que se realizó el pedido.
			hh:mm:ss	
	fin_p	Varchar	1	Indica si el pedido se confirmó (s) o no (n).

Tabla 2. Estructura de la tabla *Pedidos_m*.

Tabla Pedidos_d				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_p	Int	11	Clave única que identifica a cada pedido.
#	cod_d	Int	6	Clave única que identifica a cada disco.
	cant_d	Smallint	3	Cantidad de discos en el pedido.

Tabla 3. Estructura de la tabla *Pedidos_d*.

Tabla Usuario				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_u	Int	6	Clave única que identifica a cada usuario.
	nick_u	Varchar	80	Nickname del usuario.
	pass_u	Varchar	80	Password del usuario.

Tabla 4. Estructura de la tabla *Usuario*.

Tabla Sello				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_s	Smallint	3	Código del sello discográfico.
	desc_s	Varchar	80	Nombre del sello discográfico.

Tabla 5. Estructura de la tabla Sello.

Tabla Categoría				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_cat	Smallint	3	Clave única que identifica a cada categoría.
	desc_cat	Varchar	80	Nombre de la categoría.

Tabla 6. Estructura de la tabla Categoría.

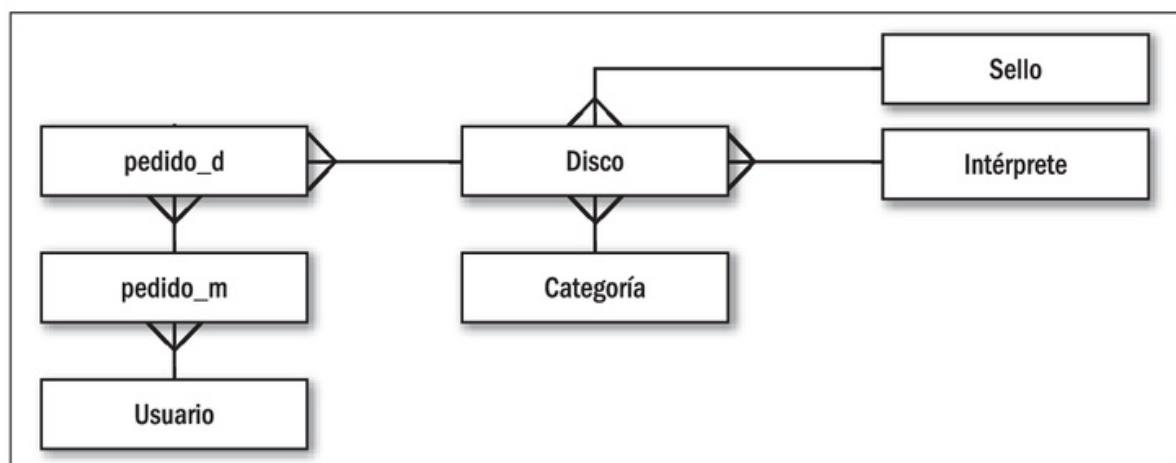
Tabla Intérprete				
	Nombre del Campo	Tipo de dato	Tamaño / Formato	Descripción
#	cod_i	Int	11	Clave única que identifica a cada intérprete.
	desc_i	Varchar	80	Nombre del intérprete.

Tabla 7. Estructura de la tabla Intérprete.

Utilizar nombres mnemotécnicos para las tablas y en cada uno de sus campos es una buena manera de ahorrar tiempo cuando programamos aplicaciones que acceden a bases de datos. Igualmente, existen los llamados diccionarios de datos, que permiten describir el significado de cada campo con nuestras propias palabras.

Recordemos que a la hora de definir el tipo de tabla existen dos opciones: **Engine** y **Type**. **Engine** se utiliza en las últimas versiones de MySQL, de modo que si cuando vamos a realizar una backup de la base la aplicación que se encarga de hacerlo usa **Type**, sería buena idea actualizar dicho programa o bien reemplazar **Type** por **Engine**.

En la imagen se pueden ver con claridad las relaciones entre las tablas.

**Figura 1. Relaciones.**

Las instrucciones SQL para la creación de la base de datos son las siguientes:

```
DROP TABLE IF EXISTS pedido_d;
DROP TABLE IF EXISTS pedido_m;
DROP TABLE IF EXISTS usuario;
DROP TABLE IF EXISTS disco;
DROP TABLE IF EXISTS interprete;
DROP TABLE IF EXISTS sello;
DROP TABLE IF EXISTS categoria;

CREATE TABLE sello (
  cod_s int(3) NOT NULL default '0',
  desc_s varchar(80) default NULL,
  PRIMARY KEY (cod_s)
) ENGINE=innodb;

CREATE TABLE interprete (
  cod_i int(11) NOT NULL default '0',
  desc_i varchar(80) NOT NULL default '',
  PRIMARY KEY (cod_i)
) ENGINE=innodb;

CREATE TABLE categoria (
  cod_cat int(3) NOT NULL auto_increment,
  desc_cat varchar(80) default NULL,
  PRIMARY KEY (cod_cat)
) ENGINE=innodb;

CREATE TABLE disco (
  cod_d int(11) NOT NULL default '0',
  nom_d varchar(80) NOT NULL default '',
  cod_i int(11) NOT NULL default '0',
  cant_d int(11) default NULL,
  cod_s int(3) default NULL,
  cod_cat int(3) default NULL,
  precio_d float(5,2) default 0,
  fec_d date default NULL,
  PRIMARY KEY (cod_d),
  KEY cod_d (cod_d),
  KEY cod_s (cod_s)
```

```

) ENGINE=innodb;

CREATE TABLE usuario (
  cod_u int(3) NOT NULL default '0',
  nick_u varchar(80) default NULL,
  pass_u varchar(80) default NULL,
  PRIMARY KEY (cod_u)
) ENGINE=innodb;

CREATE TABLE pedido_m (
  cod_p int(11) NOT NULL default '0',
  cod_u int(10) NOT NULL default '0',
  fec_p datetime default NULL,
  fin_p varchar(1) default 'n',
  PRIMARY KEY (cod_p),
  KEY cod_p (cod_p)
) ENGINE=innodb;

CREATE TABLE pedido_d (
  cod_p int(11) NOT NULL default '0',
  cod_d int(10) NOT NULL default '0',
  cant_d int(6) default NULL,
  PRIMARY KEY (cod_p,cod_d),
  KEY cod_p (cod_p),
  KEY cod_d (cod_d)
) ENGINE=innodb;

ALTER TABLE disco add FOREIGN KEY(cod_i) REFERENCES interprete(cod_i)
ON DELETE RESTRICT ON UPDATE CASCADE;
ALTER TABLE disco add FOREIGN KEY(cod_s) REFERENCES sello(cod_s)
ON DELETE RESTRICT ON UPDATE CASCADE;
ALTER TABLE disco add FOREIGN KEY(cod_cat) REFERENCES categoria(cod_cat)
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE pedido_m add FOREIGN KEY(cod_u) REFERENCES usuario(cod_u)
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE pedido_d add FOREIGN KEY(cod_p) REFERENCES pedido_m(cod_p)
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE pedido_d add FOREIGN KEY(cod_d) REFERENCES disco(cod_d) ON

```

```
DELETE RESTRICT ON UPDATE CASCADE;
```

```
INSERT INTO sello (cod_s,desc_s) VALUES (1,'B. M. G.');
```

```
INSERT INTO sello (cod_s,desc_s) VALUES (2,'Disney');
```

```
INSERT INTO sello (cod_s,desc_s) VALUES (3,'Emi');
```

```
INSERT INTO interprete (cod_i,desc_i) VALUES (1,'Rod Stewart');
```

```
INSERT INTO interprete (cod_i,desc_i) VALUES (2,'Shakira');
```

```
INSERT INTO interprete (cod_i,desc_i) VALUES (3,'Hannah Montana');
```

```
INSERT INTO interprete (cod_i,desc_i) VALUES (4,'David Guetta');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (1,'blues');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (2,'chamame');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (3,'clasica');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (4,'dance-electronica');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (5,'folklore');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (6,'hip hop');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (7,'infantiles');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (8,'instrumental/new age');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (9,'internacional');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (10,'interprete  
en castellano');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (11,'jazz');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (12,'miscelaneas');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (13,'musica brasilera');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (14,'musica celta');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (15,'musica de peliculas');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (16,'musica del mundo');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (17,'ska');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (18,'rock en castellano');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (19,'salsa y merengue');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (20,'tango');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (21,'tropical/cuartetos');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (22,'variados');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (23,'acid jazz');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (24,'soul');
```

```
INSERT INTO categoria (cod_cat,desc_cat) VALUES (25,'pop');
```

```
INSERT INTO disco (cod_d,nom_d,cod_i,cant_d,cod_s,cod_cat,precio_d,fec_d)  
VALUES (1,'The Great American Songbook 5',1,100,1,9,49.8,'2010-10-15');
```



```
INSERT INTO disco (cod_d,nom_d,cod_i,cant_d,cod_s,cod_cat,precio_d,fec_d)
VALUES (2,'Sale el Sol',2,300,1,25,49.8,'2010-10-15');
```

```
INSERT INTO disco (cod_d,nom_d,cod_i,cant_d,cod_s,cod_cat,precio_d,fec_d)
VALUES (3,'Hannah Montana Forever',3,1200,2,25,41.81,'2010-11-03');
```

```
INSERT INTO disco (cod_d,nom_d,cod_i,cant_d,cod_s,cod_cat,precio_d,fec_d)
VALUES (4,'One More Love (2cd)',4,259,3,4,57.8,'2010-11-27');
```

```
INSERT INTO disco (cod_d,nom_d,cod_i,cant_d,cod_s,cod_cat,precio_d,fec_d)
VALUES (5,'Pop Life',4,123,3,4,42.8,'2007-06-07');
```

```
INSERT INTO `usuario` (`cod_u`, `nick_u`, `pass_u`) VALUES ('1', 'usuario',
'usuario');
```

Es importante asegurarse de que el archivo pueda ejecutarse repetidas veces sin problemas. En el ejemplo de este capítulo, encontraremos una muestra acerca de cómo hacerlo: antes de crear las tablas se las elimina (si es que existían, esto se comprueba con la frase **IF EXISTS**), luego se ingresan los datos.

Si leemos con atención el orden de las instrucciones anteriores, veremos que pueden dividirse en cuatro partes: eliminación de tablas, creación de tablas, creación de claves foráneas e inserción de datos. Lo importante aquí es el orden que se ha mantenido dentro de cada sección:

- **Eliminación de tablas:** para eliminar una tabla, ésta no debe ser referenciada por ninguna otra; se establece un orden lógico para que esto suceda así.
- **Creación de tablas:** aquí no importa el orden de creación pues generamos las relaciones luego de las tablas. Si lo hacemos al crear las tablas, por ejemplo, así:

```
CREATE TABLE disco (
  cod_d int(11) NOT NULL default '0',
  nom_d varchar(80) NOT NULL default '',
  cod_i int(11) NOT NULL default '0',
  cant_d int(11) default NULL,
  cod_s int(3) default NULL,
  cod_cat int(3) default NULL,
  precio_d float default NULL,
  fec_d date default NULL,
  PRIMARY KEY (cod_d),
  KEY cod_d (cod_d),
```

```

    KEY cod_s (cod_s),
    FOREIGN KEY (cod_i) REFERENCES interprete(cod_i),
    FOREIGN KEY (cod_s) REFERENCES sello(cod_s),
    FOREIGN KEY (cod_cat) REFERENCES categoria(cod_cat)
) ENGINE=innodb;

```

En este caso, deberíamos crear las tablas referenciadas antes que las que hacen referencia a ellas. En nuestro ejemplo, aunque no sea necesario, ése es el orden de creación.

- **Creación de claves foráneas:** debemos saber que aquí sí es necesario tener en cuenta el orden mencionado en el párrafo anterior.
- **Inserción de datos:** para continuar, es importante que insertemos los registros en el mismo orden en que creamos las tablas.

Tengamos presente que los archivos que formarán parte de la primera etapa de nuestro proyecto serán los siguientes:

- | | |
|-----------------------|------------------------|
| • cnx.php | • login.php |
| • index.php | • verdiscos.php |
| • buscador.php | • carrito.php |

cnx.php

Este archivo se incluye al principio de cada página y en él se establece la conexión con la base de datos, se abre una sesión, y se incluye una función para escapar los datos a ingresar en las instrucciones **sql**.

```

<?php

$cnx = mysql_pconnect('localhost', 'root', '') or die('Error en la conexion');
$db = mysql_select_db('bddiscos');

session_start();

function clean($s) {
    if (is_array($s)) {
        foreach($s as $c => $v) {
            $s[$c] = mysql_real_escape_string($v);
        }
    }
}

```

```

    } else $s = mysql_real_escape_string($s);

    return $s;
}

?>

```

index.php

Desde aquí podemos acceder a las distintas opciones: Buscador, Ver Carrito, Novedades, Los discos más vendidos y Los artistas más vendidos.

```

<?php include 'cnx.php';    ?>
<html>
<head>
    <title>INICIO</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>

<div class="header">bienvenido a la tienda de discos</div>

<a href='buscador.php'>Buscador</a><br />
<a href='carrito.php'>Ver Carrito</a><br />
<a href='verdiscos.php?modo=n'>Novedades</a><br />
<a href='verdiscos.php?modo=v'>Los discos mas vendidos</a><br />
<a href='verdiscos.php?modo=i'>Los artistas mas vendidos</a>

</body>
</html>

```

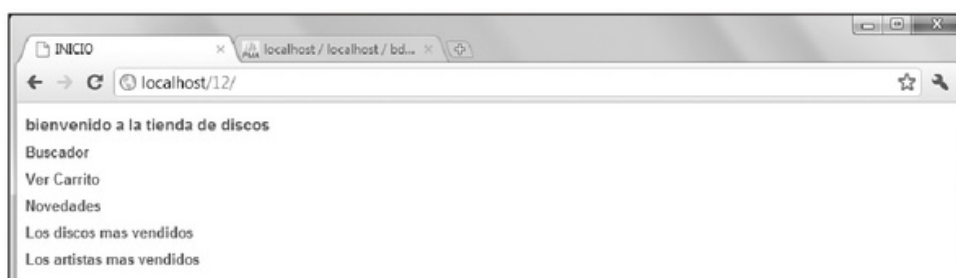


Figura 2. index.php.

buscador.php

Desde aquí podremos realizar búsquedas a través de diversos parámetros.

```
<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Buscador</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>

<div class="header">bienvenido al buscador de discos</div>

<form method="POST" action="verdiscos.php?modo=b">
<table cellpadding="0">
    <tr>
        <td>por palabra clave</td>
        <td><input type="text" id="txtpc" name="txtpc"></td>
    </tr>

    <tr>
        <td>nombre disco</td>
        <td>
            <?php

$select = '-';
$res = mysql_query("select * from disco order by nom_d");
if (mysql_num_rows($res)) {
    $select = '<select id="selND" name="selND">';
    $select .= '<option value=0 selected>Todos los discos</option>';

    while ($row = mysql_fetch_array($res))
        $select .= "<option value='$row[cod_d]'" . $row[nom_d] . "</option>";

    $select .= '</select>';
}

echo $select;

?>
```

```

</td>
</tr>

<tr>
<td>artista</td>
<td>
<?php

$select = '-';
$res = mysql_query("select * from interprete order by desc_i");
if (mysql_num_rows($res)) {
    $select = '<select id="selNI" name="selNI">';
    $select .= '<option value=0 selected>Todos los artistas</option>';

    while ($row = mysql_fetch_array($res))
        $select .= "<option value='$row[cod_i]','$row[desc_i]</option>";

    $select .= '</select>';
}

echo $select;

?>
</td>
</tr>

<tr>
<td>categoria</td>
<td>
<?php

$select = '-';
$res = mysql_query("select * from categoria order by desc_cat");
if (mysql_num_rows($res)) {
    $select = '<select id="selNC" name="selNC">';
    $select .= '<option value=0 selected>Todos las categorias</option>';

    while ($row = mysql_fetch_array($res))
        $select .= "<option
value='$row[cod_cat]','$row[desc_cat]</option>";

```

```

        $select .= '</select>';
    }

    echo $select;

?>
</td>
</tr>

<tr>
<td colspan="2"><input type="submit" id="subbuscar" name="subbuscar"></td>
</tr>
</table>
</form>

</body>
</html>

```

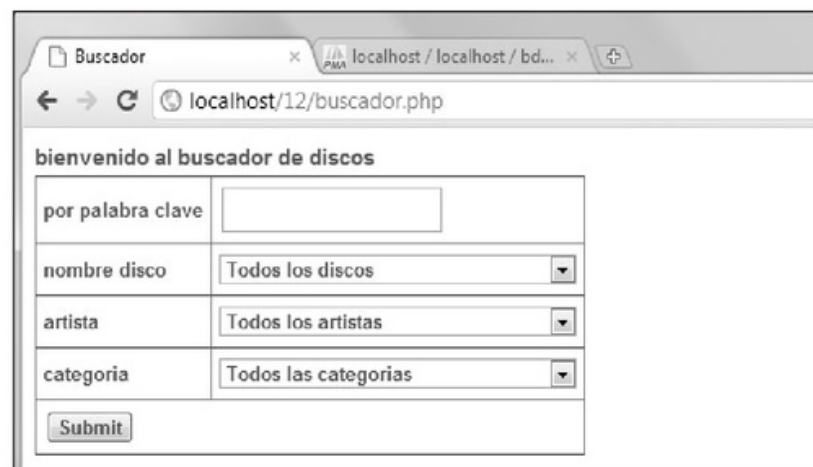


Figura 3. *buscador.php*.

login.php

Sólo los usuarios habilitados podrán realizar pedidos. La validación se hace en este archivo. La tabla **usuario** tal vez no contenga ningún valor; insertamos un registro.

```

<?php

include 'cnx.php';

```



```

if ($_POST[sublogin]) {

    $_POST = clean($_POST);

    $res = mysql_query("select * from usuario where nick_u =
    '$_POST[txtu]' and pass_u = '$_POST[txtp]'");
    if (mysql_num_rows($res)) {
        $row = mysql_fetch_array($res);
        $_SESSION[registrado] = $row[cod_u];
    } else {
        $mensaje = '<br />nombre de usuario / contraseña
incorrectos<br /><br />';
    }
}

if ($_SESSION[registrado]) {
    header("location: carrito.php");
    exit;
} else {
    echo '<html>';
    echo '<head>';
    echo '        <title>Registro de Usuarios</title>';
    echo '        <link rel="stylesheet" type="text/css" href="styles.css">';
    echo '</head>';

    echo '<body>';

    echo $mensaje;
    echo '<form method="POST" action="login.php">';
    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Ingrese su nombre de usuario:</td>';
    echo '<td><input type="text" id="txtu" name="txtu"
value="'. $_POST[txtu].'"></td>';
    echo '</tr>';
    echo '<tr>';
    echo '<td>Ingrese su contraseña:</td>';
    echo '<td><input type="text" id="txtp" name="txtp"
value="'. $_POST[txtp].'"></td>';
    echo '</tr>';

```

```

        echo '<tr>';
        echo '<td colspan="2"><input type="submit" id="sublogin"
name="sublogin" value="login"></td>';
        echo '</tr>';
        echo '<table>';
        echo '</form>';
        echo '<br /><a href="index.php">ir a inicio</a>';
        echo '</body>';
        echo '</html>';
    }

?>

```



Figura 4. login.php.

verdiscos.php

Tanto las búsquedas como las novedades, los discos más vendidos y los artistas más vendidos se resuelven en este archivo.

```

<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Ver Discos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>

<?php

```

```

if ($_GET[modo]=='b') {
    $sql = " select * from disco, categoria, interprete, sello";
    $sql .= " where disco.cod_cat = categoria.cod_cat";
    $sql .= " and disco.cod_i = interprete.cod_i";
    $sql .= " and disco.cod_s = sello.cod_s";

    if ($_POST[subbuscar]) {

        $_POST = clean($_POST);

        if ($_POST[txtpc]) {
            $sql .= " and ";
            $sql .= " ( categoria.desc_cat like '%$_POST[txtpc]%' or";
            $sql .= " interprete.desc_i like '%$_POST[txtpc]%' or ";
            $sql .= " sello.desc_s like '%$_POST[txtpc]%' or ";
            $sql .= " disco.precio_d like '%$_POST[txtpc]%' or ";
            $sql .= " disco.nom_d like '%$_POST[txtpc]%' ";
            $sql .= " )";
        }

        if ($_POST[seIND]) {
            $sql .= " and disco.cod_d = '$_POST[seIND]'";
        }

        if ($_POST[seINI]) {
            $sql .= " and interprete.cod_i = '$_POST[seINI]'";
        }

        if ($_POST[seINC]) {
            $sql .= " and categoria.cod_cat = '$_POST[seINC]'";
        }
    }

    $sql .= " order by fec_d desc";

} elseif ($_GET[modo]=='n') {
    $sql = " select * from disco, categoria, interprete, sello";
    $sql .= " where disco.cod_cat = categoria.cod_cat";
    $sql .= " and disco.cod_i = interprete.cod_i";
    $sql .= " and disco.cod_s = sello.cod_s";

```



```

    $sql .= " order by fec_d desc";

} elseif ($_GET[modo]=='v') {
    $sql = " select *, sum(pedido_d.cant_d) as K from disco, categoria,
interprete, sello, pedido_d, pedido_m";
    $sql .= " where disco.cod_cat = categoria.cod_cat";
    $sql .= " and disco.cod_i = interprete.cod_i";
    $sql .= " and disco.cod_s = sello.cod_s";
    $sql .= " and disco.cod_d = pedido_d.cod_d";
    $sql .= " and pedido_m.cod_p = pedido_d.cod_p";
    $sql .= " and pedido_m.fin_p = 's'";
    $sql .= " group by disco.cod_d";
    $sql .= " order by K desc";

} elseif ($_GET[modo]=='i') {
    $sql = " select desc_i, sum(pedido_d.cant_d) as K";
    $sql .= " from disco, categoria, interprete, sello, pedido_d, pedido_m";
    $sql .= " where disco.cod_cat = categoria.cod_cat";
    $sql .= " and disco.cod_i = interprete.cod_i";
    $sql .= " and pedido_m.cod_p = pedido_d.cod_p";
    $sql .= " and disco.cod_s = sello.cod_s";
    $sql .= " and disco.cod_d = pedido_d.cod_d";
    $sql .= " and pedido_m.fin_p = 's'";
    $sql .= " group by disco.cod_i";
    $sql .= " order by K desc";
}

$res = mysql_query($sql);
if (mysql_num_rows($res)) {

    if ($_GET[modo]=='i') {
        echo '<table cellpadding="0">';
        echo '<tr>';
        echo '<td>Artista</td>';
        echo '<td>Cantidad de discos vendidos</td>';
        echo '</tr>';
        while ($row = mysql_fetch_array($res)) {
            echo '<tr>';
            echo '<td>'.$row[desc_i]. '</td>';
            echo '<td>'.$row[K]. '</td>';
        }
    }
}

```

```

        echo '</tr>';
    }
    echo '<table>';
} else {
    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Disco</td>';
    echo '<td>Interprete</td>';
    echo '<td>Categoria</td>';
    echo '<td>Sello</td>';
    echo '<td>Precio</td>';
    echo '<td>comprar</td>';

    if ($_GET[modo]=='v')
        echo '<td>discos vendidos</td>';

    echo '</tr>';

    while ($row = mysql_fetch_array($res)) {
        echo '<tr>';
        echo '<td>'.$row[nom_d]. '</td>';
        echo '<td>'.$row[desc_i]. '</td>';
        echo '<td>'.$row[desc_cat]. '</td>';
        echo '<td>'.$row[desc_s]. '</td>';
        echo '<td>'.$row[precio_d]. '</td>';
        echo '<td><a href="carrito.php?modo=c&cod_d='.$row[cod_d].'">comprar</a></td>';

        if ($_GET[modo]=='v')
            echo '<td>'.$row[K]. '</td>';

        echo '</tr>';
    }

    echo '<table>';
}
} else {
    echo 'No se encontraron resultados';
}

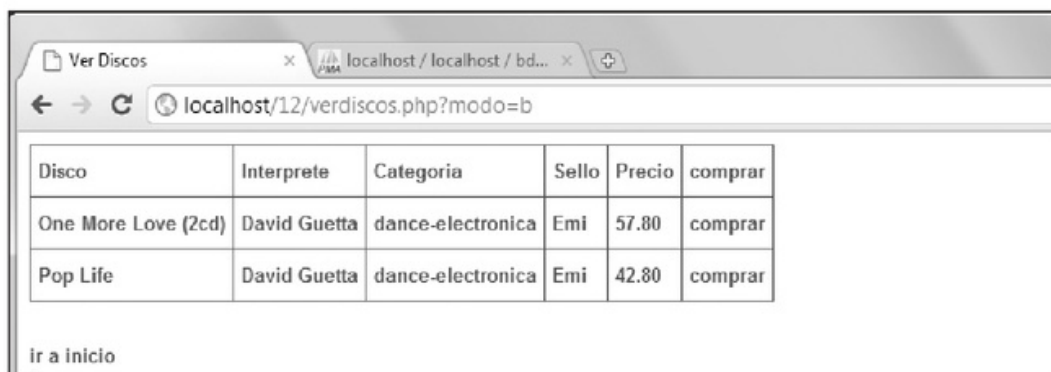
```

```

echo '<br /><a href="index.php">ir a inicio</a>';

?>
</body>
</html>

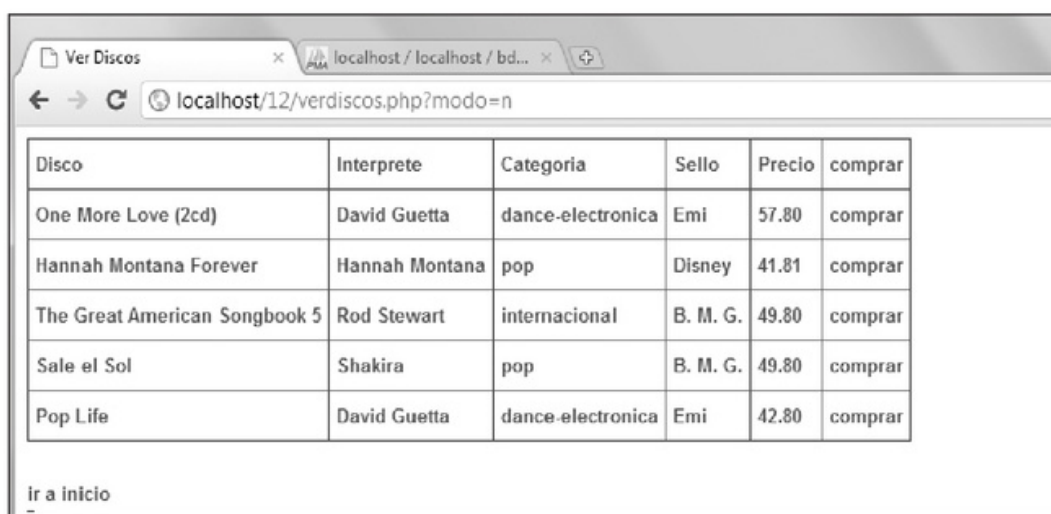
```



Disco	Interprete	Categoria	Sello	Precio	comprar
One More Love (2cd)	David Guetta	dance-electronica	Emi	57.80	comprar
Pop Life	David Guetta	dance-electronica	Emi	42.80	comprar

ir a inicio

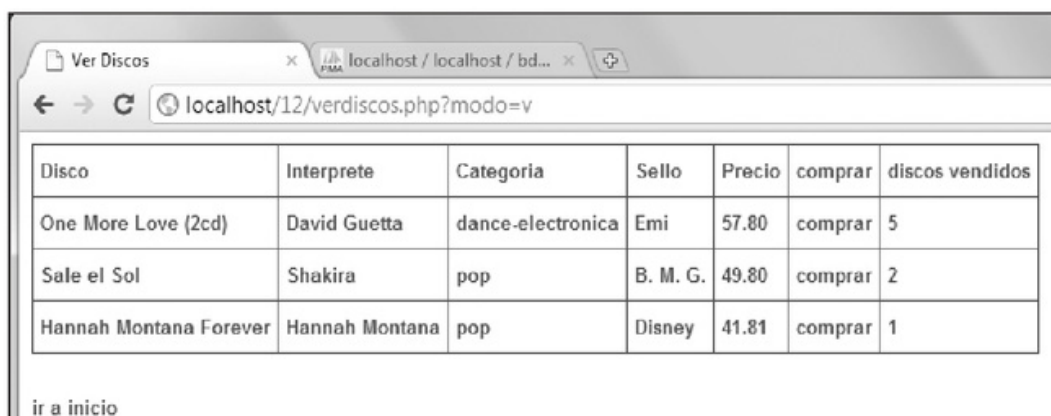
Figura 5. *verdiscos.php: resultado de la búsqueda.*



Disco	Interprete	Categoria	Sello	Precio	comprar
One More Love (2cd)	David Guetta	dance-electronica	Emi	57.80	comprar
Hannah Montana Forever	Hannah Montana	pop	Disney	41.81	comprar
The Great American Songbook 5	Rod Stewart	internacional	B. M. G.	49.80	comprar
Sale el Sol	Shakira	pop	B. M. G.	49.80	comprar
Pop Life	David Guetta	dance-electronica	Emi	42.80	comprar

ir a inicio

Figura 6. *verdiscos.php: novedades.*



Disco	Interprete	Categoria	Sello	Precio	comprar	discos vendidos
One More Love (2cd)	David Guetta	dance-electronica	Emi	57.80	comprar	5
Sale el Sol	Shakira	pop	B. M. G.	49.80	comprar	2
Hannah Montana Forever	Hannah Montana	pop	Disney	41.81	comprar	1

ir a inicio

Figura 7. *verdiscos.php: discos más vendidos.*



Artista	Cantidad de discos vendidos
David Guetta	5
Shakira	2
Hannah Montana	1

[ir a inicio](#)

Figura 8. *verdiscos.php: artistas más vendidos.*

carrito.php

Listado de los discos en el pedido.

```
<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Carrito de compras</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

if ($_SESSION[registrado]) {

    $_POST = clean($_POST);
    $_GET = clean($_GET);

    if ($_POST[subactualizar])    {

        //¿hay stock?
        for ($c=1; $c<$_POST[cant_items]; $c++) {
            $temp1 = 'cod'.'.$c;
            $temp2 = 'can'.'.$c;

            $res = mysql_query("select * from disco where
cod_d='". $_POST["$temp1"]."'" and cant_d >= '". $_POST["$temp2"]."'"");
            if (mysql_num_rows($res)) {
```

```

                                //hay stock
                                mysql_query("update pedido_d set
cant_d='".$$_POST["$temp2"]."' where cod_p='".$$_POST[cod_p]."' and
cod_d='".$$_POST["$temp1"]."'");
                                } else {
                                $mensaje = 'no hay stock disponible para algunos
discos';
                                }
                                }

                                header("location: ?");
                                exit;
                                }

                                if ($_GET[modo]=='e') {
                                mysql_query("delete from pedido_d where cod_p = '$_GET[cod_p]'
and cod_d = '$_GET[cod_d]'");
                                } elseif ($_GET[modo]=='t') {

                                mysql_query("BEGIN WORK");

                                $mensaje = '';
                                $res = mysql_query("select * from pedido_d where
cod_p='".$$_GET[cod_p]."'");
                                while ($row = mysql_fetch_array($res)) {

                                //¿hay stock?
                                $res_c = mysql_query("select * from disco where
cod_d='".$row[cod_d]."' and cant_d >= '".$row[cant_d]."'");
                                if (!mysql_num_rows($res_c)) {
                                $mensaje = 'no hay stock disponible para
algunos discos';
                                }
                                }

                                mysql_data_seek($res, 0);

                                if (!$mensaje) {
                                while ($row= mysql_fetch_array($res)) {
                                mysql_query("update disco set cant_d = cant_d -

```

```

““.$row[cant_d].”” where cod_d = ““.$row[cod_d].””);

        }

        mysql_query(“update pedido_m set fin_p = ‘s’ where
cod_p=’”.$_GET[cod_p].””);
        $mensaje = ‘Pedido confirmado. Gracias por su compra.’;
    }

    mysql_query(“COMMIT”);

} elseif ($_GET[modo]==‘c’) {

    if ($_GET[cod_d]) {

        //¿hay stock?
        $res = mysql_query(“select * from disco where
cod_d=’”.$_GET[cod_d].”” and cant_d > ‘0’”);
        if (mysql_num_rows($res))        {

            //¿pedido nuevo o en curso?
            $res = mysql_query(“select * from pedido_m where
fin_p = ‘n’ and cod_u=’”.$_SESSION[registrado].””);
            if (mysql_num_rows($res)) {
                $row = mysql_fetch_array($res);
                //pedido en curso

                //¿ya habia seleccionado ese disco?
                $res = mysql_query(“select * from
pedido_d where cod_p = ““.$row[cod_p].”” and cod_d = ““.$_GET[cod_d].””);
                if (mysql_num_rows($res)) {
                    $mensaje = ‘el disco ya estaba
seleccionado’;

                } else {
                    //agrego el disco
                    mysql_query(“insert into pedido_d
values ($row[cod_p], $_GET[cod_d], 1)”);
                }
            } else {
                //pedido nuevo

```



```

                                $res = mysql_query("select max(cod_p)
as M from pedido_m");

                                $row = mysql_fetch_array($res);
                                if ($row[M]) {
                                    $max = $row[M] +1;
                                } else {
                                    $max = 1;
                                }

                                $fecha = date("Y-m-d h:i:s");

                                //creo el pedido
                                mysql_query("insert into pedido_m values
('$max', '$_SESSION[registrado]', '$fecha', 'n')");

                                //agrego el disco
                                mysql_query("insert into pedido_d values
('$max', '$_GET[cod_d]', '1')");
                                }

                                } else
                                    $mensaje = 'el disco que pidio no esta en
stock<br />';
                                }
                                }

                                $sql = " select nom_d, pedido_d.cod_p, desc_i, precio_d,
pedido_d.cant_d, disco.cod_d";
                                $sql .= " from pedido_m, pedido_d, disco, interprete, usuario";
                                $sql .= " where pedido_m.cod_p = pedido_d.cod_p";
                                $sql .= " and disco.cod_i = interprete.cod_i";
                                $sql .= " and pedido_m.cod_u = usuario.cod_u";
                                $sql .= " and pedido_m.cod_u = usuario.cod_u";
                                $sql .= " and usuario.cod_u=$_SESSION[registrado];
                                $sql .= " and pedido_d.cod_d = disco.cod_d";
                                $sql .= " and pedido_m.fin_p = 'n'";

                                $res = mysql_query($sql);

```

```

echo $mensaje;

echo '<form method="POST">';

echo '<table cellpadding="0">';
echo '<tr>';
echo '<td>disco</td>';
echo '<td>interprete</td>';
echo '<td>precio</td>';
echo '<td>cantidad</td>';
echo '<td>eliminar</td>';
echo '</tr>';

if (mysql_num_rows($res)) {
    $c=1;
    while ($row = mysql_fetch_array($res)) {
        echo '<tr>';
        echo '<td>'.$row[nom_d]. '</td>';
        echo '<td>'.$row[desc_i]. '</td>';
        echo '<td>'.$row[precio_d]. '</td>';
        echo '<td><input type="text" id="can' . $c . '"
name="can' . $c . '" value="'. $row[cant_d]. '"></td>';
        echo '<td><a href=carrito.php?modo=e&cod_d=' .
$row[cod_d]. '&cod_p=' . $row[cod_p]. '>eliminar</a></td>';
        echo '<tr>';
        echo '<input type="hidden" id="cod' . $c . '"
name="cod' . $c . '" value="'. $row[cod_d]. '" />';

        $cod_p = $row[cod_p];
        $total += $row[precio_d] * $row[cant_d];
        $c++;
    }

    echo '<tr><td colspan="5">Total: '.round($total, 2). '</td></tr>';
    echo '<tr><td colspan="5"><input type="submit"
id="subactualizar" name="subactualizar" value="Actualizar
Cantidades"></td></tr>';
    echo '<tr><td colspan="5"><a href=carrito.php?modo=t&cod_p=' .
$row[cod_p]. '>Terminar Pedido</a></td></tr>';
} else {

```

```

        echo '<tr><td colspan="5">No hay productos cargados</td></tr>';

    }

    echo '</table>';

    echo '<input type="hidden" id="cant_items" name="cant_items"
value="'. $c. '" />';
    echo '<input type="hidden" id="cod_p" name="cod_p" value="'. $cod_p. '" />';
    echo '</form>';

    echo '<br /><a href="index.php">ir a inicio</a>';

} else {
    header("location: login.php");
    exit;
}

?>

</body>
</html>

```

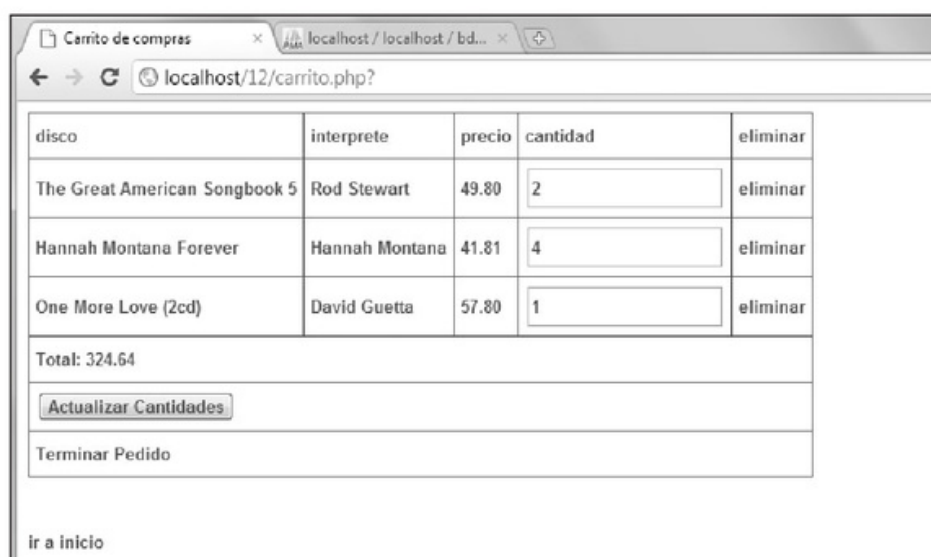


Figura 9. *carrito.php.*

Ahora veremos la parte administrativa de nuestro sitio, un lugar al cual los administradores podrán acceder para realizar tareas de mantenimiento y actualización de datos. Cuenta con las siguientes opciones:

- un menú de opciones.
- dar de alta discos.
- dar de baja discos.
- modificar los datos de los discos.
- dar de alta categorías.
- dar de baja categorías.
- modificar los datos de las categorías.
- dar de alta intérpretes.
- dar de baja intérpretes.
- modificar los datos de los intérpretes.
- dar de alta sellos.
- dar de baja sellos.
- modificar los datos de los sellos.
- ventas por mes.
- ver cuáles fueron los usuarios más activos.

Menú de opciones

Desde aquí podremos acceder de manera directa a las opciones mencionadas anteriormente. Se agrega en las páginas a través de la función **include**.

```
<?php

echo '<table cellpadding="0">';
echo '<tr>';
echo '<td><a href="interprete_alta.php">Alta de Interpretes</a></td>';
echo '<td><a href="interprete_baja.php">Baja de Interpretes</a></td>';
echo '<td><a href="interprete_mod.php">Modificacion de Interpretes</a></td>';
echo '<td><a href="ventasxmes.php">Ventas del mes</a></td>';
echo '</tr>';
echo '<tr>';
echo '<td><a href="disco_alta.php">Alta de Discos</a></td>';
echo '<td><a href="disco_baja.php">Baja de Discos</a></td>';
echo '<td><a href="disco_mod.php">Modificacion de Discos</a></td>';
echo '<td><a href="usuariosactivos.php">Usuarios mas activos</a></td>';
echo '</tr>';
echo '<tr>';
echo '<td><a href="cat_alta.php">Alta de Categorias</a></td>';
echo '<td><a href="cat_baja.php">Baja de Categorias</a></td>';
echo '<td><a href="cat_mod.php">Modificacion de Categorias</a></td>';
```

```
echo '<td><a href="">&nbsp;</a></td>';
echo '</tr>';
echo '<tr>';
echo '<td><a href="sello_alta.php">Alta de Sellos</a></td>';
echo '<td><a href="sello_baja.php">Baja de Sellos</a></td>';
echo '<td><a href="sello_mod.php">Modificacion de Sellos</a></td>';
echo '<td><a href="">&nbsp;</a></td>';
echo '</tr>';
echo '</table><br><br>';

?>
```

Dar de alta discos

```
<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Alta de Discos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

if ($_POST[subgrabar]) {

    $_POST = clean($_POST);

    mysql_query('BEGIN WORK');

    if ($_POST[nombre]) {
        $res = mysql_query("select max(cod_d) as M from disco");
        $row = mysql_fetch_array($res);
        if ($row[M])
            $max = $row[M] +1;
        else
            $max = 1;
    }
}
```

```

        $fecha = date("Y-m-d h:i:s");

        $sql = "insert into disco values (";
        $sql .= $max.", ";
        $sql .= "““._POST[nombre].””, ";
        $sql .= $_POST[interprete].”, ";
        $sql .= $_POST[stock].”, ";
        $sql .= $_POST[sello].”, ";
        $sql .= $_POST[categoria].”, ";
        $sql .= $_POST[precio].”, ";
        $sql .= "““.$fecha.””)";

        $res = mysql_query($sql);

        mysql_query('COMMIT');
    }
}

echo '<form method="POST" action="?">';

$res = mysql_query("select * from interprete order by desc_i asc");
while ($row = mysql_fetch_array($res))
    $select1 .= "<option value=$row[cod_i]>$row[desc_i]</option>";

$res = mysql_query("select * from categoria order by desc_cat asc");
while ($row = mysql_fetch_array($res))
    $select2 .= "<option value=$row[cod_cat]>$row[desc_cat]</option>";

$res = mysql_query("select * from sello order by desc_s asc");
while ($row = mysql_fetch_array($res))
    $select3 .= "<option value=$row[cod_s]>$row[desc_s]</option>";

echo '<table cellpadding="0">';
echo '<tr>';
echo '<td>Nombre</td><td><input type="text" name="nombre"></td>';
echo '</tr>';
echo '<tr>';
echo '<td>Stock inicial</td><td><input type="text" name="stock"></td>';
echo '</tr>';
echo '<tr>';

```



```

echo '<td>Precio</td><td><input type=text name=precio></td>';
echo '</tr>';
echo '<tr>';
echo '<td>Categoria</td><td><select name=categoria>'.$select2.'</select></td>';
echo '</tr>';
echo '<tr>';
echo '<td>Interprete</td><td><select name=interprete>'.$select1.'</select></td>';
echo '</tr>';
echo '<tr>';
echo '<td>Sello</td><td><select name=sello>'.$select3.'</select></td>';
echo '</tr>';
echo '<tr>';
echo '<td colspan="2"><input value="Dar de Alta" type="submit"
name="subgrabar"></td>';
echo '</tr>';
echo '</table>';

echo '</form>';

?>

</body>
</html>

```

Alta de Interpretes	Baja de Interpretes	Modificacion de Interpretes	Ventas del mes
Alta de Discos	Baja de Discos	Modificacion de Discos	Usuarios mas activos
Alta de Categorias	Baja de Categorias	Modificacion de Categorias	
Alta de Sellos	Baja de Sellos	Modificacion de Sellos	

Nombre	
Stock inicial	
Precio	
Categoria	acid jazz
Interprete	David Guetta
Sello	B. M. G.
Dar de Alta	

Figura 10. disco_alta.php: alta de discos.

Dar de baja discos

```

<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Baja de Discos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

if ($_GET[b_cod_d]) {
    $_GET = clean($_GET);

    $sql = "delete from disco where cod_d=".$_GET[b_cod_d];
    $res = mysql_query($sql);
}

$sql = " select * from disco, categoria, interprete, sello";
$sql .= " where disco.cod_cat = categoria.cod_cat";
$sql .= " and disco.cod_i = interprete.cod_i";
$sql .= " and disco.cod_s = sello.cod_s";
$sql .= " order by fec_d desc";

$res = mysql_query($sql);

if (mysql_num_rows($res))    {
    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Disco</td>';
    echo '<td>Interprete</td>';
    echo '<td>Categoria</td>';
    echo '<td>Sello</td>';
    echo '<td>Precio</td>';
    echo '<td>dar de baja</td>';
    echo '</tr>';

    while ($row = mysql_fetch_array($res)) {

```

```

        echo '<tr>';
        echo '<td>'.$row[nom_d].'/>';
        echo '<td>'.$row[desc_i].'/>';
        echo '<td>'.$row[desc_cat].'/>';
        echo '<td>'.$row[desc_s].'/>';
        echo '<td>'.$row[precio_d].'/>';
        echo '<td><a href="disco_baja.php?b_cod_d='.$row[cod_d].
'">dar de baja</a></td>';
        echo '</tr>';
    }

    echo '<table>';
} else {
    echo 'No se encontraron discos';
}

?>

</body>
</html>

```

Alta de Interpretos	Baja de Interpretos	Modificacion de Interpretos	Ventas del mes
Alta de Discos	Baja de Discos	Modificacion de Discos	Usuarios mas activos
Alta de Categorias	Baja de Categorias	Modificacion de Categorias	
Alta de Sellos	Baja de Sellos	Modificacion de Sellos	

Disco	Interprete	Categoria	Sello	Precio	dar de baja
One More Love (2cd)	David Guetta	dance-electronica	Emi	57.80	dar de baja
Hannah Montana Forever	Hannah Montana	pop	Disney	41.81	dar de baja
The Great American Songbook 5	Rod Stewart	internacional	B. M. G.	49.80	dar de baja
Sale el Sol	Shakira	pop	B. M. G.	49.80	dar de baja
Pop Life	David Guetta	dance-electronica	Emi	42.80	dar de baja

Figura 11. disco_baja.php: eliminación de discos.

Modificar los datos de los discos

```

<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Modificacion de Disco</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_GET = clean($_GET);
$_POST = clean($_POST);

if ($_POST[subgrabar]) {
    if ($_POST[nombre]) {
        $sql = "update disco set";
        $sql .= " nom_d = '".$_POST[nombre]."', ";
        $sql .= " cod_i = '".$_POST[interprete]."', ";
        $sql .= " cant_d = '".$_POST[stock]."', ";
        $sql .= " cod_s = '".$_POST[sello]."', ";
        $sql .= " cod_cat = '".$_POST[categoria]."', ";
        $sql .= " precio_d = '".$_POST[precio];
        $sql .= " where cod_d = '".$_POST[cod_d];

        $res = mysql_query($sql);
    }

    $_GET[cod_d]=$_POST[cod_d];
}

if ($_GET[cod_d]) {

    $sql_d = " select * from disco, categoria, interprete, sello";
    $sql_d .= " where disco.cod_cat = categoria.cod_cat";
    $sql_d .= " and disco.cod_i = interprete.cod_i";
    $sql_d .= " and disco.cod_s = sello.cod_s";
    $sql_d .= " and disco.cod_d = '".$_GET[cod_d];

```

```

$sql_d .= " order by fec_d desc";

$res_d = mysql_query($sql_d);
$row_d = mysql_fetch_array($res_d);

echo '<form method=POST action="?">';

$res = mysql_query("select * from interprete order by desc_i asc");
while ($row = mysql_fetch_array($res)) {
    if ($row[cod_i]==$row_d[cod_i])
        $select1 .= "<option selected value=$row[cod_i]>$row
[desc_i]</option>";
    else
        $select1 .= "<option value=$row[cod_i]>$row[desc_i]</option>";
}

$res = mysql_query("select * from categoria order by desc_cat asc");
while ($row = mysql_fetch_array($res)) {
    if ($row[cod_cat]==$row_d[cod_cat])
        $select2 .= "<option selected value=$row[cod_cat]
>$row[desc_cat]</option>";
    else
        $select2 .= "<option value=$row[cod_cat]>$row[desc_cat]</option>";
}

$res = mysql_query("select * from sello order by desc_s asc");
while ($row = mysql_fetch_array($res)) {
    if ($row[cod_cat]==$row_d[cod_cat])
        $select3 .= "<option selected value=$row[cod_s]>$row
[desc_s]</option>";
    else
        $select3 .= "<option value=$row[cod_s]>$row[desc_s]</option>";
}

echo '<table cellpadding="0">';
echo '<tr>';
echo '<td>Nombre</td><td><input type=text name=nombre
value="'. $row_d[nom_d]. '"></td>';
echo '</tr>';
echo '<tr>';

```

```

        echo '<td>Stock inicial</td><td><input type=text name=stock
value=""$.row_d[cant_d]."></td>';
        echo '</tr>';
        echo '<tr>';
        echo '<td>Precio</td><td><input type=text name=precio
value=""$.row_d[precio_d]."></td>';
        echo '</tr>';
        echo '<tr>';
        echo '<td>Categoria</td><td><select name=categoria>'.$select2.'</select></td>';
        echo '</tr>';
        echo '<tr>';
        echo '<td>Interprete</td><td><select name=interprete>'.$select1.
'</select></td>';
        echo '</tr>';
        echo '<tr>';
        echo '<td>Sello</td><td><select name=sello>'.$select3.'</select></td>';
        echo '</tr>';
        echo '<tr>';
        echo '<td colspan="2"><input value=Modificar type="submit"
name="subgrabar"></td>';
        echo '</tr>';
        echo '</table>';

        echo '<input type=hidden name=cod_d value='$.row_d[cod_d].'>';

        echo '</form>';
    } else {
        $sql = " select * from disco, categoria, interprete, sello";
        $sql .= " where disco.cod_cat = categoria.cod_cat";
        $sql .= " and disco.cod_i = interprete.cod_i";
        $sql .= " and disco.cod_s = sello.cod_s";
        $sql .= " order by fec_d desc";

        $res = mysql_query($sql);

        if (mysql_num_rows($res)>0) {
            echo '<table cellpadding="0">';
            echo '<tr>';
            echo '<td>Disco</td>';
            echo '<td>Interprete</td>';

```



```

        echo '<td>Categoria</td>';
        echo '<td>Sello</td>';
        echo '<td>Precio</td>';
        echo '<td>dar de baja</td>';
        echo '</tr>';

        while ($row = mysql_fetch_array($res)) {
            echo '<tr>';
            echo '<td>'.$row[nom_d]. '</td>';
            echo '<td>'.$row[desc_i]. '</td>';
            echo '<td>'.$row[desc_cat]. '</td>';
            echo '<td>'.$row[desc_s]. '</td>';
            echo '<td>'.$row[precio_d]. '</td>';
            echo '<td><a href="disco_mod.php?cod_d='.$row[cod_d].
'>ver detalle</a></td>';
            echo '</tr>';
        }

        echo '<table>';
    } else {
        echo 'No se encontraron discos';
    }
}

?>

</body>
</html>

```

Como vemos en la siguiente imagen, el archivo denominado **disco_mod.php** nos presenta una serie de datos interesantes con respecto a los discos.

BASE

Debemos tener en cuenta que el objetivo principal de este proyecto es tomar el código expuesto para luego poder modificarlo, ampliarlo, y mejorarlo de manera tal que nos sirva para entrar en confianza en relación al lenguaje en sí. De esta forma estaremos mejor preparados para enfrentar otros desafíos.

Alta de Interpretes	Baja de Interpretes	Modificacion de Interpretes	Ventas del mes
Alta de Discos	Baja de Discos	Modificacion de Discos	Usuarios mas activos
Alta de Categorías	Baja de Categorías	Modificacion de Categorías	
Alta de Sellos	Baja de Sellos	Modificacion de Sellos	

Disco	Interprete	Categoria	Sello	Precio	dar de baja
One More Love (2cd)	David Guetta	dance-electronica	Emi	57.80	ver detalle
Hannah Montana Forever	Hannah Montana	pop	Disney	41.81	ver detalle
The Great American Songbook 5	Rod Stewart	internacional	B. M. G.	49.80	ver detalle
Sale el Sol	Shakira	pop	B. M. G.	49.80	ver detalle
Pop Life	David Guetta	dance-electronica	Emi	42.80	ver detalle

Figura 12. disco_mod.php: modificación de datos en discos.

Dar de alta categorías

```
<?php include 'cnx.php';  ?>
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
    <title>Alta de Categorías</title>
</head>
<body>
<?php

include 'menu.php';

if ($_POST[subgrabar]!='')    {
    $res = mysql_query('BEGIN WORK');

    $sql = "select max(cod_cat) as M from categoria";
    $res = mysql_query($sql);
    $row = mysql_fetch_array($res);
    if ($row[M]>0)
        $max = $row[M] +1;
```

```
        else
            $max = 1;

        $sql = "insert into categoria values (";
        $sql .= $max.", ";
        $sql .= "'".$_POST[categoria]."' ) ";

        $res = mysql_query($sql);

        $res = mysql_query('COMMIT');
    }

    echo '<form method=POST action="">';

    echo '<table align=center border=1>';
    echo '<tr>';
    echo '<td>Nombre de categoria</td><td><input type=text name=categoria></td>';
    echo '</tr>';
    echo '<tr>';
    echo '<td colspan="2"><input value="Dar de Alta" type="submit"
        name="subgrabar"></td>';
    echo '</tr>';
    echo '</table>';

    echo '</form>';

    ?>
</body>
</html>
```

En la siguiente imagen vemos la forma en que se dan de alta categorías.

MAS INFORMACIÓN

Debemos saber que es posible encontrar todos los ejemplos de éste y los demás capítulos que componen este libro en el sitio oficial de la editorial, desde donde, además, podremos descargar capítulos de ejemplo en formato PDF. Para ello debemos visutar la dirección: [\[confirmar\]](#).



Figura 13. *cat_alta.php: alta de categorías.*

Dar de baja categorías

```
<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Baja de Categorías</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_POST = clean($_POST);

if ($_POST[subgrabar]) {
    for ($i=1; $i<=$_POST[cant]; $i++) {
        $temp = 'check'.$i;
        if ($_POST[$temp]) { //atencion a las comillas dobles
            $sql = "delete from categoria where cod_cat =
".$_POST[$temp];
            $res = mysql_query($sql);
        }
    }
}
```

```

$sql = "select * from categoria order by desc_cat asc";
$res = mysql_query($sql);

if (mysql_num_rows($res))    {

    echo '<form action="?" method=POST>';
    echo '<table cellpadding="0">';
    echo '<tr><td>Nombre</td><td>dar de baja</td></tr>';

    $c=1;
    while ($row = mysql_fetch_array($res))  {
        echo '<tr>';
        echo '<td>'.$row[desc_cat]. '</td>';
        echo '<td><input type=checkbox name=check'.$c.'
value='.$row[cod_cat]. '></td>';
        echo '</tr>';
        $c++;
    }

    echo '<input type=hidden name=cant value='.$c.'>';
    echo '<tr>';
    echo '<td colspan="2"><input value="Dar de Baja" type="submit"
name="subgrabar"></td>';

    echo '</tr>';
    echo '</table>';
    echo '</form>';

} else {
    echo 'No se encontraron categorias';

}

?>

</body>
</html>

```

En la siguiente imagen vemos la forma en que se dan de baja categorías.

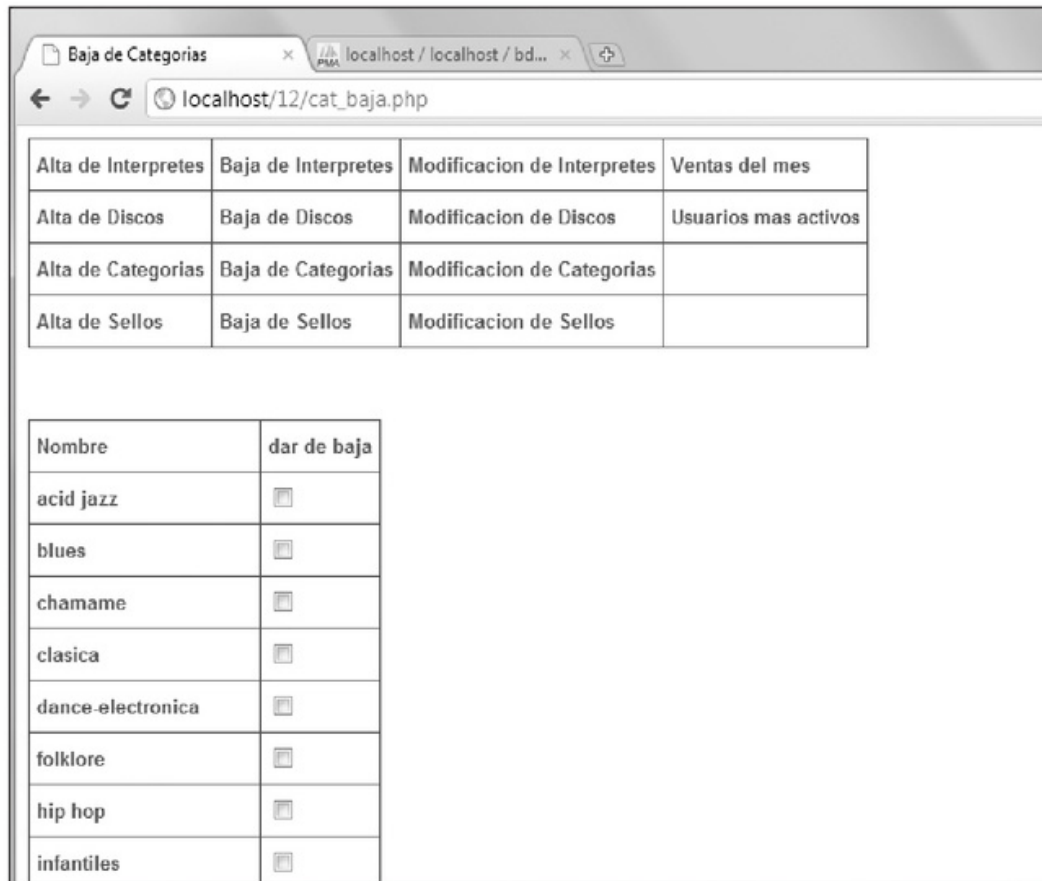


Figura 14. *cat_baja.php: baja de categorías.*

Modificar los datos de las categorías

```
<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Modificacion de Categorías</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_POST = clean($_POST);
$_GET = clean($_GET);

if ($_POST[subgrabar]) {
    $sql = " update categoria set";
```



```

    $sql .= " desc_cat = ".$_POST[categoria]."";
    $sql .= " where cod_cat = ".$_POST[cod_cat];

    $res = mysql_query($sql);
    $_GET[cod_cat]='';
}

if ($_GET[cod_cat]) {

    $sql = "select * from categoria where cod_cat = ".$_GET[cod_cat];
    $res = mysql_query($sql);
    $row = mysql_fetch_array($res);

    echo '<form method=POST action="?">';

    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Nombre de categoria</td><td><input type=text name=categoria
value="'.$row[desc_cat].'"></td>';
    echo '</tr>';
    echo '<tr>';
    echo '<td colspan="2"><input value="Modificar" type="submit"
name="subgrabar"></td>';
    echo '</tr>';
    echo '</table>';

    echo '<input type=hidden name=cod_cat value='.$row[cod_cat].'>';

    echo '</form>';

} else {

    $sql = "select * from categoria order by desc_cat asc";
    $res = mysql_query($sql);

    if (mysql_num_rows($res)>0) {
        echo '<table cellpadding="0">';
        echo '<tr><td>Nombre de categoria</td><td>ver detalle</td></tr>';
        while ($row = mysql_fetch_array($res)) {
            echo '<tr>';

```

```

        echo '<td>'.$row[desc_cat].'/><td><a
href=cat_mod.php?cod_cat='.$row[cod_cat]. '>ver detalle</a></td>';
        echo '</tr>';
    }
    echo '</table>';
} else {
    echo 'no hay categorias disponibles';
}
}
?>
</body>
</html>

```



Figura 15. *cat_mod.php: modificar categorías.*

Dar de alta intérpretes

```

<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Alta de Interpretres</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

```

```
if ($_POST[subgrabar]) {

    $_POST = clean($_POST);

    mysql_query('BEGIN WORK');

    $res = mysql_query("select max(cod_i) as M from interprete");
    $row = mysql_fetch_array($res);
    if ($row[M])
        $max = $row[M] +1;
    else
        $max = 1;

    if ($_POST[interprete])
        mysql_query("insert into interprete values (". $max.",
        "._$_POST[interprete].")");

    mysql_query('COMMIT');
}

echo '<form method="POST" action="?">';

echo '<table cellpadding="0">';
echo '<tr>';
echo '<td>Nombre del Interprete</td><td><input type="text" id="interprete"
name="interprete"></td>';
echo '</tr>';
echo '<tr>';
echo '<td colspan="2"><input value="Dar de Alta" type="submit"
name="subgrabar"></td>';
echo '</tr>';
echo '</table>';

echo '</form>';

?>

</body>
</html>
```

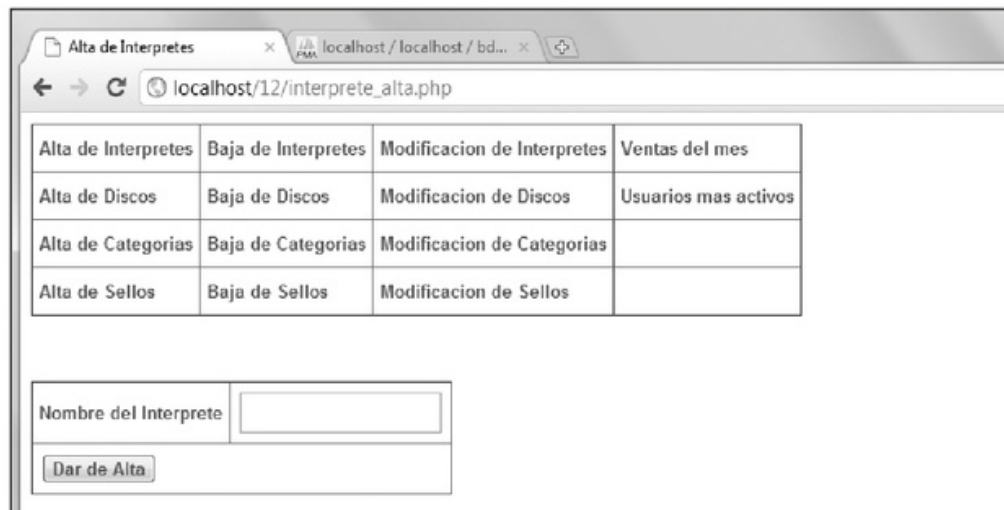



Figura 16. *interprete_alta.php: alta de intérpretes.*

Dar de baja intérpretes

```
<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Baja de Interpretes</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_POST = clean($_POST);

if ($_POST[subgrabar])    {
    for ($i=1; $i<=$_POST[cant]; $i++)    {
        $temp = 'check'.$i;
        if ($_POST["$temp"]) { //atencion a las comillas dobles
            $sql = "delete from interprete where cod_i = ".$_POST["$temp"];
            $res = mysql_query($sql);
        }
    }
}

$sql = "select * from interprete order by desc_i asc";
```

```

$res = mysql_query($sql);

if (mysql_num_rows($res)) {
    echo '<form action="?" method=POST>';
    echo '<table cellpadding="0">';
    echo '<tr><td>Nombre</td><td>dar de baja</td></tr>';

    $c=1;
    while ($row = mysql_fetch_array($res)) {
        echo '<tr>';
        echo '<td>'. $row[desc_i]. '</td>';
        echo '<td><input type=checkbox name=check'. $c. '
value='. $row[cod_i]. '></td>';
        echo '</tr>';
        $c++;
    }
    echo '<input type=hidden name=cant value='. $c. '>';
    echo '<tr>';
    echo '<td colspan="2"><input value="Dar de Baja" type="submit"
name="subgrabar"></td>';
    echo '</tr>';
    echo '</table>';
    echo '</form>';
} else {
    echo 'No se encontraron interpretes';
}

?>

</body>
</html>

```

CSS

Las hojas de estilo nos permiten modificar de manera sencilla y rápida los diseño de nuestros desarrollos. Además de esto, utilizarlas nos da la alternativa de optimizar nuestros proyectos, algo que deriva directamente en la velocidad de carga.

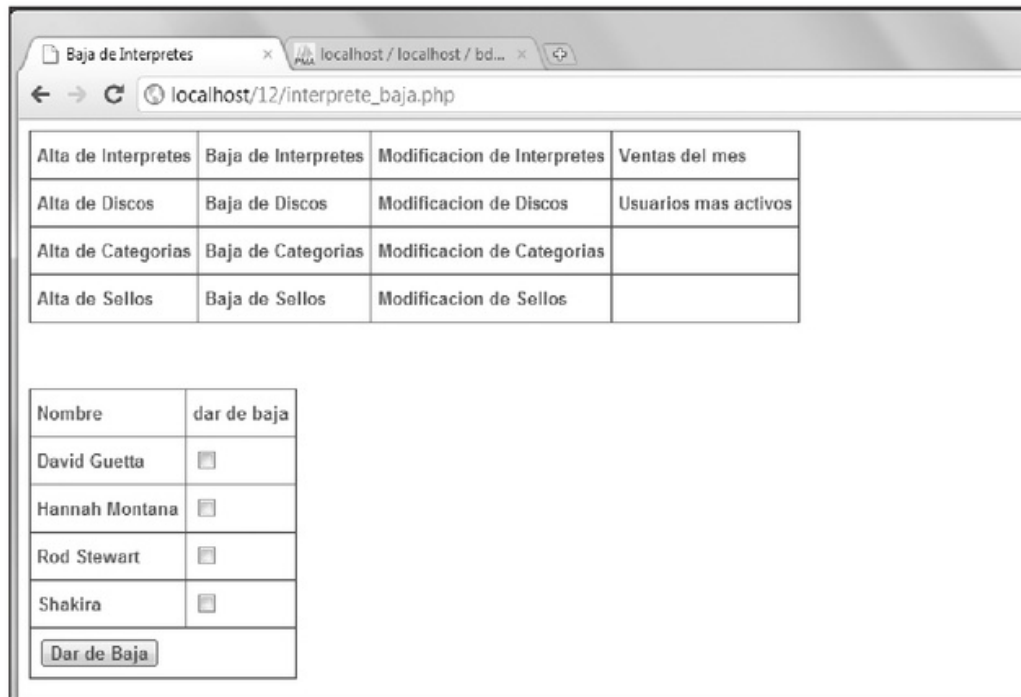


Figura 17. *interprete_baja.php: baja de intérpretes.*

Modificar los datos de los intérpretes

```
<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Modificacion de Interpretes</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_POST = clean($_POST);
$_GET = clean($_GET);

if ($_POST[subgrabar]) {
    $sql = " update interprete set";
    $sql .= " desc_i = '".$_POST[interprete]."'";
    $sql .= " where cod_i = '".$_POST[cod_i];

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



```
    $_GET[cod_i]='';
}

if ($_GET[cod_i]) {

    $sql = "select * from interprete where cod_i = ".$_GET[cod_i];
    $res = mysql_query($sql);
    $row = mysql_fetch_array($res);

    echo '<form method=POST action=?>';

    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Nombre del interprete</td><td><input type=text
name=interprete value="'.$row[desc_i].'"></td>';
    echo '</tr>';
    echo '<tr>';
    echo '<td colspan="2"><input value="Modificar" type="submit"
name="subgrabar"></td>';
    echo '</tr>';
    echo '</table>';

    echo '<input type=hidden name=cod_i value="'.$row[cod_i].'">';

    echo '</form>';

} else {

    $sql = "select * from interprete order by desc_i asc";
    $res = mysql_query($sql);

    if (mysql_num_rows($res)>0) {
        echo '<table cellpadding="0">';
        echo '<tr><td>Nombre del interprete</td><td>ver detalle</td></tr>';
        while ($row = mysql_fetch_array($res)) {
            echo '<tr>';
            echo '<td>'.$row[desc_i]. '</td><td><a
href=interprete_mod.php?cod_i='.$row[cod_i]. '>ver detalle</a></td>';
            echo '</tr>';
        }
    }
}
```

```

    }
    echo '</table>';
  } else {
    echo 'no hay interprete disponibles';
  }
}

?>

</body>
</html>

```

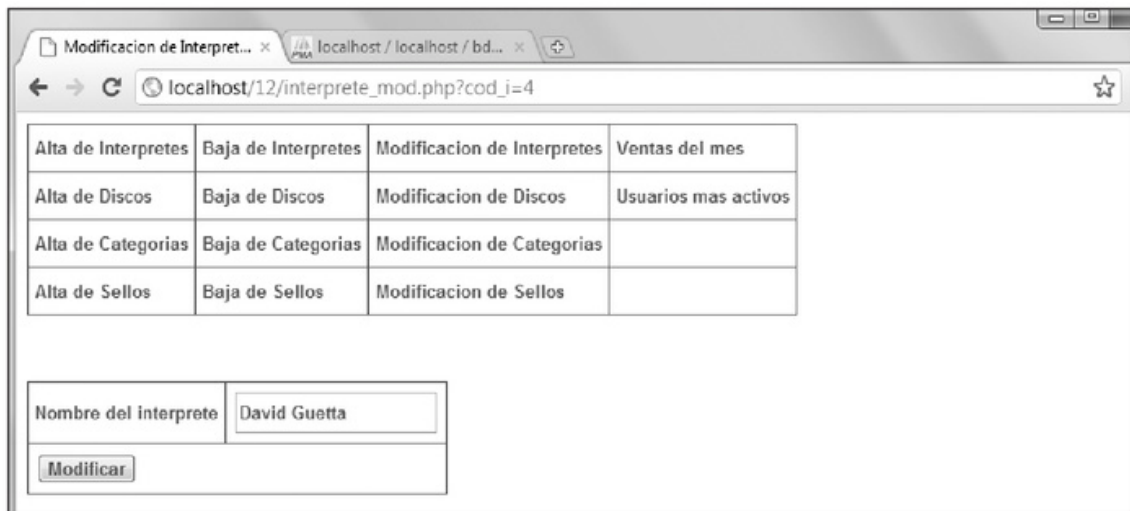


Figura 18. *interprete_mod.php: modificación de datos de intérpretes.*

Dar de alta sellos

```

<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Alta de Sellos Discograficos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

```

```

$_POST = clean($_POST);

if ($_POST[subgrabar]) {
    mysql_query('BEGIN WORK');

    $sql = "select max(cod_s) as M from sello";
    $res = mysql_query($sql);
    $row = mysql_fetch_array($res);
    if ($row[M]>0)
        $max = $row[M] +1;
    else
        $max = 1;

    $sql = "insert into sello values (";
    $sql .= $max.", ";
    $sql .= "“".$_POST[sello].”" ) ";

    $res = mysql_query($sql);

    mysql_query('COMMIT');
}

echo '<form method=POST action=?>';

echo '<table cellpadding="0">';
echo '<tr>';
echo '<td>Nombre del Sello</td><td><input type=text name=sello</td>';
echo '</tr>';
echo '<tr>';
echo '<td colspan="2"><input value="Dar de Alta" type="submit" name="subgrabar"></td>';
echo '</tr>';
echo '</table>';

echo '</form>';

?>

</body>
</html>

```

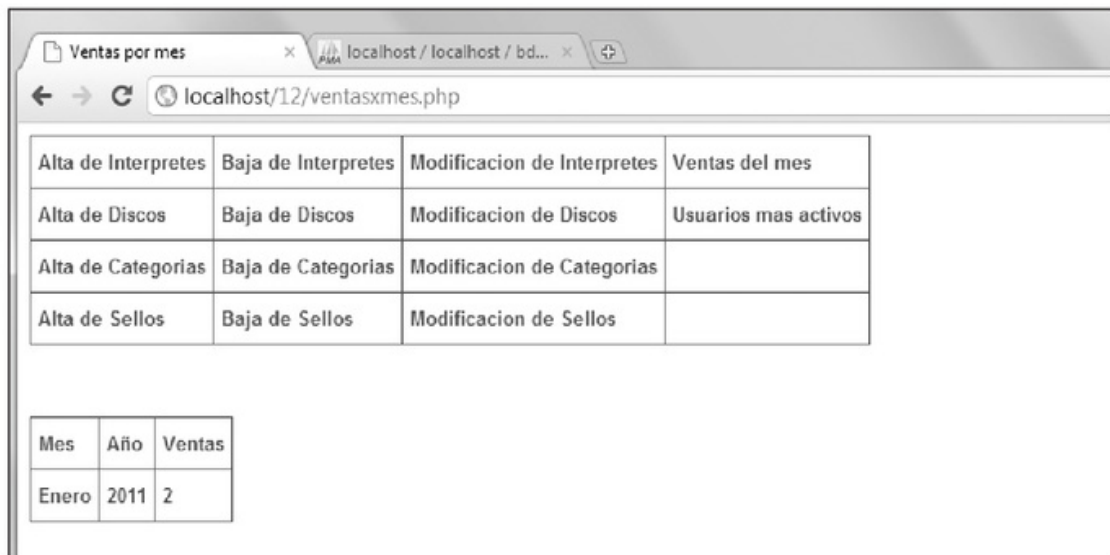



Figura 19. *sello_alta.php: inserción de nuevos sellos discográficos.*

Dar de baja sellos

```
<?php include 'cnx.php';    ?>
<html>
<head>
    <title>Baja de Sellos Discograficos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_POST = clean($_POST);

if ($_POST[subgrabar])    {
    for ($i=1; $i<=$_POST[cant]; $i++)    {
        $temp = 'check'.$i;
        if ($_POST["$temp"]) { //atencion a las comillas dobles
            $sql = "delete from sello where cod_s = ".$_POST["$temp"];
            $res = mysql_query($sql);
        }
    }
}
```

```

$sql = "select * from sello order by desc_s asc";

$res = mysql_query($sql);

if (mysql_num_rows($res)) {
    echo '<form action="" method=POST>';
    echo '<table cellpadding="0">';
    echo '<tr><td>Nombre</td><td>dar de baja</td></tr>';

    $c=1;
    while ($row = mysql_fetch_array($res)) {
        echo '<tr>';
        echo '<td>'.$row[desc_s]. '</td>';
        echo '<td><input type=checkbox name=check'.$c.'
value='.$row[cod_s]. '></td>';
        echo '</tr>';
        $c++;
    }
    echo '<input type=hidden name=cant value='.$c.'>';
    echo '<tr>';
    echo '<td colspan="2"><input value="Dar de Baja" type="submit"
name="subgrabar"></td>';
    echo '</tr>';
    echo '</table>';
    echo '</form>';
} else {
    echo 'No se encontraron sellos';
}
?>
</body>
</html>

```

FORM

Es interesante recordar que los formularios web nacieron hace mucho tiempo y sin embargo siguen siendo utilizados en toda clase de proyectos: su uso nos permite resolver cuestiones cotidianas de manera sencilla y rápida, sin complicaciones.

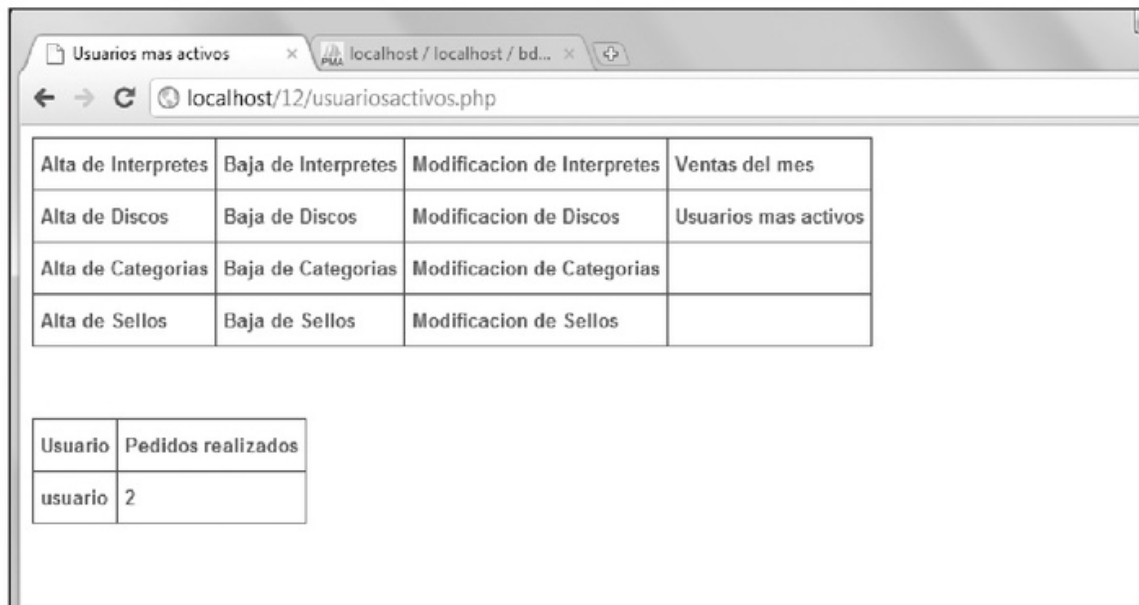


Figura 20. *sello_baja.php: eliminación de sellos discográficos.*

Modificar los datos de los sellos

```
<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Modificacion de Sellos Discograficos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

$_POST = clean($_POST);
$_GET = clean($_GET);

if ($_POST[subgrabar])    {
    $sql = " update sello set";
    $sql .= " desc_s = '".$_POST[sello]."'";
    $sql .= " where cod_s = '".$_POST[cod_s]";

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



```

        $_GET[cod_s]='';
    }

    if ($_GET[cod_s]) {

        $sql = "select * from sello where cod_s = ".$_GET[cod_s];
        $res = mysql_query($sql);
        $row = mysql_fetch_array($res);

        echo '<form method=POST action=?>';

        echo '<table cellpadding="0">';
        echo '<tr>';
        echo '<td>Nombre del sello</td><td><input type=text name=sello
value="'.$row[desc_s].'"></td>';
        echo '</tr>';
        echo '<tr>';
        echo '<td colspan="2"><input value="Modificar" type="submit"
name="subgrabar"></td>';
        echo '</tr>';
        echo '</table>';

        echo '<input type=hidden name=cod_s value='.$row[cod_s].'>';

        echo '</form>';

    } else {

        $sql = "select * from sello order by desc_s asc";
        $res = mysql_query($sql);

        if (mysql_num_rows($res)>0) {
            echo '<table cellpadding="0">';
            echo '<tr><td>Nombre del Sello</td><td>ver detalle</td></tr>';
            while ($row = mysql_fetch_array($res)) {
                echo '<tr>';
                echo '<td>'.$row[desc_s]. '</td><td><a
href=sello_mod.php?cod_s='.$row[cod_s]. '>ver detalle</a></td>';
                echo '</tr>';
            }
        }
    }
}

```

```

    }
    echo '</table>';
  } else {
    echo 'no hay sellos disponibles';
  }
}

?>
</body>
</html>

```

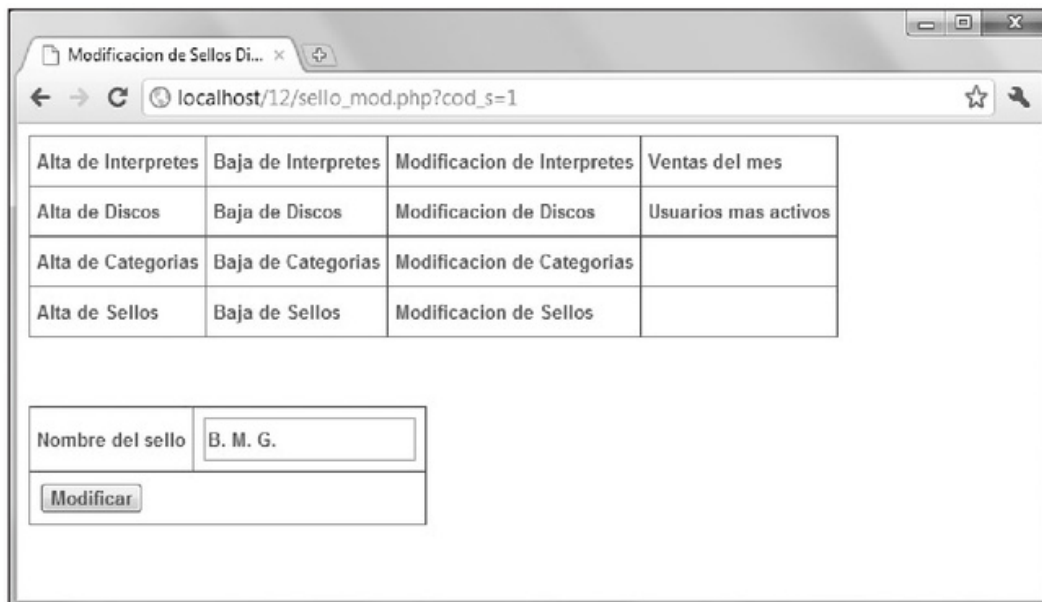


Figura 21. *sello_mod.php: modificación de sellos discográficos.*

Ventas por mes

```

<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Ventas por mes</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

```

```
include 'menu.php';

$res = mysql_query("select month(fec_p) as M, year(fec_p) as A, count(*)
as C from pedido_m group by year(fec_p), month(fec_p) order by fec_p desc");
if (mysql_num_rows($res)) {

    $mes[1]='Enero';
    $mes[2]='Febrero';
    $mes[3]='Marzo';
    $mes[4]='Abril';
    $mes[5]='Mayo';
    $mes[6]='Junio';
    $mes[7]='Julio';
    $mes[8]='Agosto';
    $mes[9]='Septiembre';
    $mes[10]='Octubre';
    $mes[11]='Noviembre';
    $mes[12]='Diciembre';

    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Mes</td>';
    echo '<td>Año</td>';
    echo '<td>Ventas</td>';

    while ($row = mysql_fetch_array($res)) {
        echo '<tr>';
        echo '<td>'.$mes[$row[M]].'</td>';
        echo '<td>'.$row[A].'</td>';
        echo '<td>'.$row[C].'</td>';
    }
}
```

SEGURIDAD

Si bien en este ejemplo no se incluye, se recomienda como ejercicio para con el lector desarrollar un área restringida para que solo los usuarios registrados puedan ingresar a las secciones relativas al agregado, editado, y eliminado de ítems.


```

    }

    echo '</table>';
} else {
    echo 'no se produjeron ventas';
}

?>

</body>
</html>

```

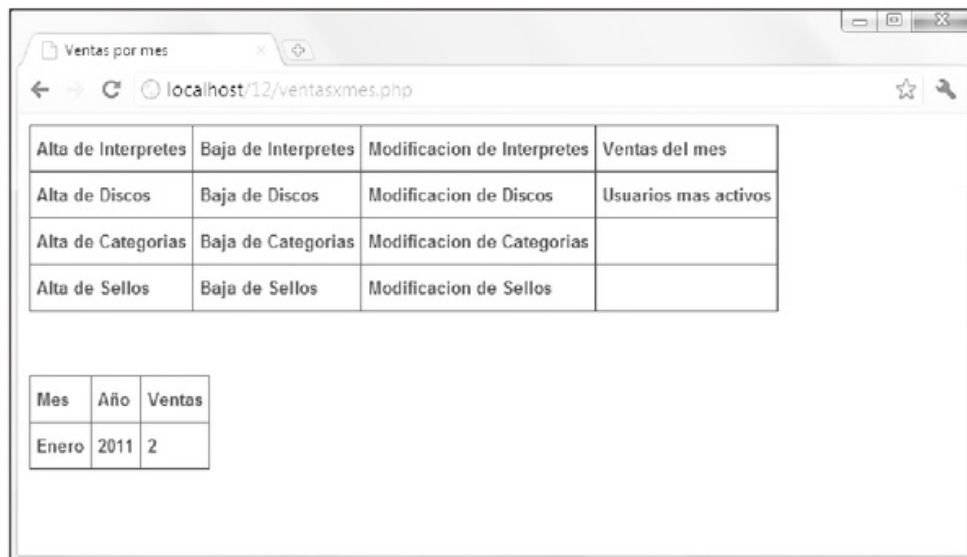


Figura 22. *ventasxmes.php*: número de ventas por mes.

Usuarios más activos

```

<?php include 'cnx.php';  ?>
<html>
<head>
    <title>Usuarios mas activos</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<?php

include 'menu.php';

```

```

$sql = "select count(*) as C, nick_u from pedido_m, usuario where
pedido_m.cod_u=usuario.cod_u group by usuario.cod_u order by C desc";
$res = mysql_query($sql);
if (mysql_num_rows($res)) {

    echo '<table cellpadding="0">';
    echo '<tr>';
    echo '<td>Usuario</td>';
    echo '<td>Pedidos realizados</td>';

    while ($row = mysql_fetch_array($res)) {
        echo '<tr>';
        echo '<td>'.$row[nick_u].'</td>';
        echo '<td>'.$row[C].'</td>';
    }

    echo '</table>';
} else {
    echo 'no se produjeron ventas';
}

?>

</body>
</html>

```

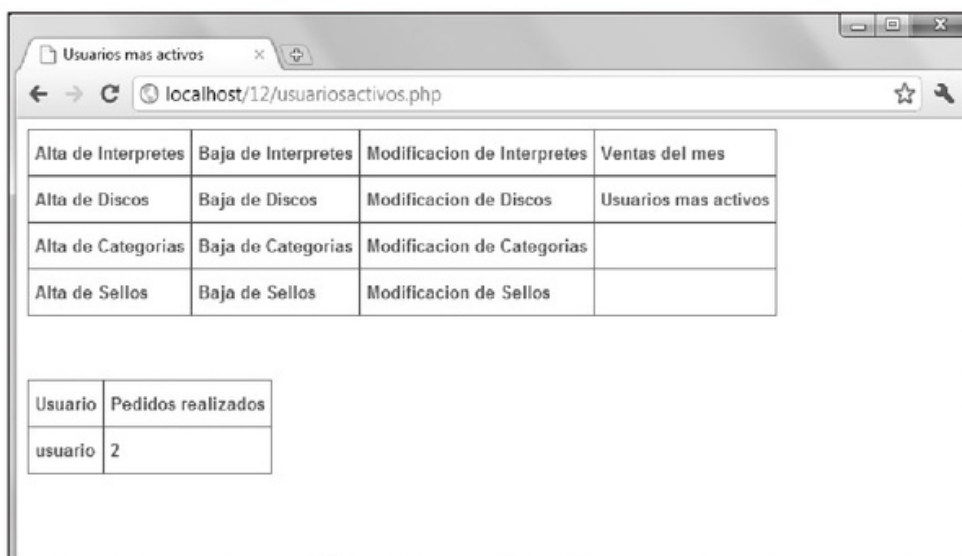


Figura 23. usuariosactivos.php: usuarios que más compraron.

MySQLi

Encontraremos algunas de las funciones provistas por esta extensión parecidas, en cuanto a la sintaxis, a las implementadas por la extensión original de MySQL. Esto sucede porque es necesario brindar compatibilidad a las aplicaciones que quieran migrar a la versión 4.1 o superiores de MySQL. Igualmente, recomendamos leer los capítulos anteriores para tener una visión más amplia en cuanto a los cambios producidos de una extensión a otra.

REFERENCIA DE FUNCIONES

La extensión MySQLi provee grandes mejoras con respecto a la extensión anterior, y a la vez intenta mantener la compatibilidad de las funciones ya existentes. De todas maneras, si vamos a actualizar nuestra aplicación para que trabaje con esta nueva extensión, debemos prestar atención a los nombres de las funciones y a los argumentos que reciben. La *i* de MySQLi significa **improved** –**mejorado** en inglés– y ya por el nombre nos damos cuenta de hacia donde apunta esta nueva característica de PHP que se actualiza a las de MySQL: mantener lo hecho hasta ahora y mejorar las prestaciones, siempre sobre la base de lo ya hecho. Viene con la distribución de PHP, más precisamente desde la versión 5 en adelante. En versiones anteriores no podremos utilizar estas mejoras.

Existen dos formas de trabajar con esta extensión: la procedural, que es la que se utiliza para trabajar con la extensión anterior, y la orientada a objetos (ver **mysqli_connect**). En cada función descrita en este **Apéndice** encontraremos ejemplos y formas de uso según ambas alternativas.

A continuación, incluimos una referencia de las funciones más utilizadas de la extensión **MySQLi**, ordenadas alfabéticamente:

mysqli_affected_rows

Esta función devuelve el número de filas **afectadas** por la última instrucción SQL y difiere de la función **mysql_affected_rows** en el sentido en que no sólo trabaja con instrucciones de tipo **INSERT**, **UPDATE** o **DELETE**, sino que admite cualquier otra.

Su sintaxis es:

```
mysqli_affected_rows(identificador);
```

En donde vemos que el **identificador** corresponde a un identificador de conexión. Este argumento es opcional y si se opta por no incluirlo, la función intenta encontrar una conexión abierta al servidor MySQL.

En MySQLi para establecer una conexión existe la función **mysqli_connect**, descrita en este mismo **Apéndice**. Ejemplos de la función;

Si utilizamos la forma orientada a objetos:

```
<?php  
  
$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");
```

```

$cxn->query("insert into tabla1 values (1, 'a')");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->query("delete from tabla1 where codigo = 3");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->query("update tabla1 set codigo = 4 where codigo = 3");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->query("select * from tabla1 where codigo = 4");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->query("delete from tabla1 where codigo = 3");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->close();

?>

```

Si utilizamos la forma procedural:

```

<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "update tabla1 set codigo = 44 where codigo > 2";
$res = mysqli_query($con);

echo mysqli_affected_rows($con);

mysqli_close();

?>

```

```

<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "delete from tabla1 where codigo > 3";

```

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

echo mysqli_affected_rows($con);

mysqli_close();

?>
```

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

$sql = "insert into tabla1 values (3, 'algo')";
$res = mysqli_query($sql);

echo mysqli_affected_rows($con);

mysqli_close();

?>
```

Si la instrucción SQL sobre la que trabaja, **mysqli_affected_rows** es inválida, la función devuelve **-1**. Si tuvo éxito devuelve el número de filas afectadas.

mysqli_autocommit

Esta función permite configurar si MySQL ejecutará una instrucción **COMMIT** después de cada instrucción, o si el programador deberá definir esta acción de forma expresa. En la implementación orientada a objetos, la función recibe un argumento (**true** o **false**) que determina el comportamiento.

Forma de uso:

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cnx->autocommit(TRUE);

$cxn->query("insert into tabla1 values (1, 'a')");
echo "filas afectadas: ".$cnx->affected_rows;
```



```
$cxn->query("delete from tabla1 where codigo = 3");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->close();

?>
```

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cxn->autocommit(FALSE);

$cxn->query("insert into tabla1 values (1, 'a')");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->query("delete from tabla1 where codigo = 3");
echo "filas afectadas: ".$cxn->affected_rows;

$cxn->close();

?>
```

En la implementación procedural también, y su forma de uso es la siguiente:

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

mysqli_autocommit(TRUE);

$sql = "update tabla1 set codigo = 44 where codigo > 2";

$res = mysqli_query($con,$sql);
echo mysqli_affected_rows($con);

?>
```

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

mysqli_autocommit(FALSE);

$sql = "update tabla1 set codigo = 44 where codigo > 2";
$res = mysqli_query($sql);

echo mysqli_affected_rows($con);

?>
```

mysqli_change_user

Nos permite una vez establecida la conexión cambiar de **usuario**, **password**, e incluso, **base de datos sin cerrar la conexión**. Su sintaxis es la siguiente:

Forma procedural:

mysqli_change_user (identificador , "nombre de usuario", "password",
"base de datos");

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

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

mysqli_change_user("usuario2", "contraseña2", "bd2");

?>
```

Forma orientada a objetos:

objeto_mysqli->change_user("nombre de usuario", "password", "base de datos");

Aquí no se especifica la conexión. Al igual que como sucede en **mysql_change_user** (ver **mysqli_connect** para más información).

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cxn->query("select * from tabla1");

$cxn->change_user("usuario2", "contraseña2", "bd");

$cxn->close();

?>
```

Los argumentos **base de datos** e **identificador** son opcionales. Si se especifica **base de datos**, ésta será la base por defecto luego del cambio de usuario. Se especifica **identificador** en el caso de que tenga más de una conexión abierta, para indicar cuál de ellas desea modificar (en la forma procedural si especifica **identificador** debe indicar sí o sí la **base de datos**).

Si la combinación final no es válida (o sea que el usuario es incorrecto, o que no se corresponde con el password, o que no existe la base de datos, etc.) la función devuelve falso y se mantiene la conexión anterior.

mysqli_close

Podemos cerrar una conexión utilizando la función **mysqli_close**.

Su sintaxis es la siguiente:

Forma procedural:

mysqli_close(identificador);

El **identificador** es 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

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "update tabla1 set codigo = 4";
$res = mysqli_query($sql);
```



```
mysqli_close($con);

?>
```

Forma orientada a objetos:

objeto_mysqli->close();

```
<?php

$link = mysqli_connect("localhost", "u", "p");
mysqli_close($link);

?>
```

Esta función devuelve **true** (verdadero) si tuvo éxito, y **false** (falso) si algo falló.

mysqli_commit

Esta función se encarga de ejecutar la instrucción denominada **COMMIT**. A continuación podemos apreciar algunos ejemplos:

objeto_mysqli->commit;

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cnx->autocommit(FALSE);

$cxn->query("insert into tabla1 values (1, 'a')");

$cxn->query("delete from tabla1 where codigo = 3");

$cxn->commit();

$cnx->close();

?>
```

mysqli_commit(identificador);

Recordemos que el **identificador** consiste en un identificador de conexión. Este argumento es opcional y si elegimos no incluirlo, la función asumirá la última conexión abierta con el servidor MySQL.

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

mysqli_autocommit(FALSE);

$sql = "update tabla1 set codigo = 44 where codigo > 2";
$res = mysqli_query($sql);

mysqli_commit();

mysqli_close();

?>
```

mysqli_connect_errno

Similar a **mysqli_errno**, sólo que se encarga de interceptar errores relacionados al establecimiento de una conexión exclusivamente.

Ejemplo de uso:

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd_que_no_existe");

echo 'error numero: '.mysqli_connect_errno();

?>
```

mysqli_connect_error

Similar a **mysqli_error**, sólo que se encarga de realizar la interceptación errores relacionados al establecimiento de una conexión exclusivamente.

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd_que_no_existe");

echo 'descripción del error: ';
echo mysqli_connect_error();

?>
```

mysqli_connect

Esta función intenta conectar a una base de datos. Para establecer una conexión en la forma procedural, la sintaxis de esta función es la siguiente:

mysqli_connect("servidor", "nombre de usuario", "password", "nombre_bd", puerto);

- **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 en donde se encuentra instalado el servidor de bases de datos MySQL. Si estamos trabajando en forma local podemos utilizar como nombre de servidor **localhost** ó **127.0.0.1** como dirección IP correspondiente.
- **Nombre de usuario:** nombre de usuario válido para acceder a la base de datos.
- **Password:** se trata de la contraseña correspondiente al nombre de usuario que hayamos ingresado con anterioridad.
- **Nombre_bd:** nombre de la base de datos a la que queremos conectar.
- **Puerto:** número de puerto sobre el que queremos trabajar.

Como ya se dijo, los argumentos son opcionales y si no se especifica alguno se asumen los valores por defecto que se encuentran en el archivo **php.ini** (o **php3.ini** según la versión de PHP). Si tenemos valores por defecto en **php.ini** pero igualmente completamos los argumentos de **mysqli_connect**, se tendrán en cuenta estos últimos. Al igual que en la función **mysql_connect**, los argumentos son opcionales pero piramidales. Podemos incluir los siguientes:

- Ningún argumento.
- Sólo el nombre del servidor.
- Sólo el nombre del servidor y el nombre de usuario.
- El nombre del servidor, el nombre de usuario y el password.

Es decir que no se puede, por ejemplo, incluir el password y dejar vacíos los demás argumentos. PHP tiene valores por defecto (no los incluidos en el **php.ini**, otros) que

se usan cuando un determinado argumento fue omitido en la llamada a **mysqli_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.** El del propietario del proceso del servidor.
- **Password.** Vacía.

Esta función devuelve **1 (verdadero)** si logró conectarse y **0 (falso)** si algo falló. Si logró conectarse devuelve, además, un **identificador de conexión**, que podrá ser utilizado como argumento por otras funciones.

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

?>
```

Debemos saber que para establecer una conexión de la forma orientada a objetos, trabajamos de la forma que analizamos a continuación:

```
$variable = new mysqli("servidor", "nombre de usuario", "password", "nombre_bd",  
puerto);
```

Lo que acabamos de hacer es crear una instancia de la clase **mysqli**, que finalmente es una conexión con el servidor de bases de datos.

\$variable es el nombre del objeto y obviamente podría haber sido cualquier otro. Aquí los argumentos vuelven a ser opcionales.

```
<?php

$cxn = new mysqli("localhost", "juan", "666", "clientes");

//la conexión actual se llama $cxn

?>
```

mysqli_data_seek

Función que tiene como propósito el poder posicionarse en algún registro específico de los devueltos por una consulta de selección. Su sintaxis es la siguiente:

Forma procedural,

mysqli_data_seek(id_consulta, numero_fila);

- **id_consulta**: se envía una consulta al servidor (con **mysqli_query**, por ejemplo) de bases de datos. El resultado devuelto por esta función se denomina **id_consulta**.
- **numero_fila**: número de fila a la cual queremos acceder.

A la primera fila corresponde el valor **0** y a la última **mysqli_num_rows(id_consulta) - 1**. Luego de llamar a la función, la primera llamada a cualquier función para recorrer las filas devueltas, devolverá la fila número **numero_fila**. Por ejemplo, si tenemos la tabla:

```
<?php

$sql = "select * from tabla1";

$res = mysqli_query($sql);

//nos posicionamos en la tercera fila
//devuelta por la consulta, si es que existe:

if (mysqli_num_rows($res)>=3)
    mysqli_data_seek($res ,2);

?>
```

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

resultado_mysqli->data_seek(numero_fila);

```
<?php

$res = $cxn->query("select * from tabla1");

if ($res->num_rows >= 3)
    $res->data_seek(2);

?>
```

mysqli_errno

Permite conocer el código del último error cometido, si es que se cometió alguno. Sintaxis de la forma procedural, como sigue:

mysqli_errno(identificador);

El **identificador** es un identificador de conexión. Este argumento es opcional y si se opta por no incluirlo, la función intenta encontrar la última conexión abierta activa al servidor MySQL correspondiente.

Un ejemplo de esta función:

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "tabla1 set codigo = 44 where codigo > ";
$res = mysqli_query($sql);

echo mysqli_errno($con);

?>
```

Debemos saber que ésta función devuelve, como se dijo, el código de error cometido o bien cero si no se cometió ningún error.

Forma orientada a objetos:

objeto_mysqli->errno;

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cxn->query("select * from tabla1 where");

echo 'error numero: '.$cxn->errno;

$cxn->close();

?>
```


mysqli_error

Permite obtener la descripción del último error cometido, si es que se cometió alguno. Sintaxis de la forma procedural:

mysqli_error(identificador);

El **identificador** es un identificador de conexión. Este argumento es opcional y si se opta por no incluirlo la función intentará encontrar la última conexión abierta activa al servidor MySQL que corresponde.

Un ejemplo de esta función:

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "tabla1 set codigo = 44 where codigo > ";
$res = mysqli_query($con);

echo mysqli_error($con);

?>
```

Esta función devuelve, como se dijo, la descripción del error cometido, o bien una cadena vacía si no se cometió ningún error.

Forma orientada a objetos:

objeto_mysqli->error;

```
<?php

$cnx = new mysqli("localhost", "usuario", "contraseña", "bd");

$cnx->query("select * from tabla1 where");

echo 'error numero: '.$cnx->error;

$cnx->close();

?>
```

mysqli_escape_string

Cuando interactuamos con cadenas de caracteres dentro de nuestras instrucciones SQL, debemos tener cuidado al **escapar** explícitamente algunos signos. Veamos un ejemplo de esta situación en el código siguiente:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$buscar = "d'mode";

mysqli_query("select * from tabla where c like $buscar");

?>
```

Lo anterior produce un error porque el carácter ' debe ser **escapado** antes de ejecutar la consulta. Una solución a esto sería la siguiente:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$buscar = "d\'mode";

mysqli_query("select * from tabla where c like $buscar");

?>
```

La función denominada **mysqli_escape_string** (o también su alias, conocido como **mysqli_real_escape_string**, da lo mismo usar una u otra) hace esta clase de tarea automáticamente y trabaja de la siguiente manera:

Forma procedural,

mysqli_escape_string(identificador, cadena);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");
```

```
$buscar = "d'mode";

$buscar = mysqli_escape_string($con, $buscar);

mysqli_query("select * from tabla where c like $buscar");

?>
```

Forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$buscar = "d'mode";

$buscar = $cxn->real_escape_string($cxn, $buscar);

$cxn->query("select * from tabla where c like $buscar");

?>
```

Esta función necesita que le demos como argumento el identificador, porque una conexión puede trabajar con distintos juegos de caracteres.

mysqli_fetch_array

Mediante esta función recuperamos filas y accedemos a los valores de cada campo como si de una matriz (asociativa o numérica) se tratase.

Su sintaxis, en la forma procedural, es la siguiente:

mysqli_fetch_array(id_consulta, tipo_indice);

- **id_consulta:** se envía una consulta al servidor (con **mysqli_query**, por ejemplo) de bases de datos. El resultado devuelto se denomina aquí **id_consulta**: para recuperar los datos devueltos por la consulta (si es que los hay) una de las funciones que se puede utilizar es **mysqli_fetch_array** a la cual se le pasa como argumento el identificador de la consulta (en este caso **id_consulta**).

- **tipo_indice**: se dijo que esta función trata a cada fila como si fuera una matriz. Este argumento permite definir el tipo de índice. Las opciones son **MYSQLI_ASSOC**, **MYSQLI_NUM**, y **MYSQLI_BOTH**. Este argumento es opcional y si se opta por no incluirlo se asume por defecto **MYSQLI_BOTH**. Debe escribirse con letras mayúsculas.

Con **MYSQLI_ASSOC** se utiliza como índice el nombre del campo (columna). Los nombres de campos son sensibles a mayúsculas y minúsculas.

Con **MYSQLI_NUM** se utiliza como índice el número del campo (columna) según el orden dado en la consulta SQL. El índice 0 corresponde al primer campo.

Con **MYSQLI_BOTH** podemos utilizar como índice el número o el nombre del campo (columna). Es como utilizar **MYSQLI_ASSOC** y **MYSQLI_NUM** al mismo tiempo. Los nombres de campos son sensibles a mayúsculas y minúsculas.

Ejemplo de utilización en la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select c1, c2 from tabla";
$res = mysqli_query($con, $sql);

$c=1;

while ($row = mysqli_fetch_array($res, MYSQLI_ASSOC))
{
    echo 'fila '.$c.' - columna 1: '.$row[c1];
    echo '<br>';
    echo 'fila '.$c.' - columna 2: '.$row[c2];
    $c++;
}

mysqli_close($con);

?>
```

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");
```

```
$sql = "select c1, c2 from tabla";
$res = mysqli_query($sql);
$c=1;

while ($row = mysqli_fetch_array($res, MYSQLI_NUM))
{
    echo 'fila '.$c.' - columna 1: '.$row[0];
    echo '<br>';
    echo 'fila '.$c.' - columna 2: '.$row[1];
    $c++;
}

mysqli_close();

?>
```

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select c1, c2 from tabla";
$res = mysqli_query($sql);

$c=1;

while ($row = mysqli_fetch_array($res))
{

    echo 'fila '.$c.' - columna 1: '.$row[c1];
    echo '<br>';
    echo 'fila '.$c.' - columna 2: '.$row[1];
    $c++;
}

mysqli_close();

?>
```

En la forma orientada a objetos, su forma de uso es la siguiente:

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select c1, c2 from tabla");

$c=1;

while ($row = $res->fetch_array(MYSQLI_ASSOC))    {
    echo 'fila '.$c.' - columna 1: '.$row[c1];
    echo '<br>';
    echo 'fila '.$c.' - columna 2: '.$row[c2];
    $c++;
}

$cxn->close();

?>
```

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select c1, c2 from tabla");

$c=1;

while ($row = $res->fetch_array(MYSQLI_NUM))    {
    echo 'fila '.$c.' - columna 1: '.$row[0];
    echo '<br>';
    echo 'fila '.$c.' - columna 2: '.$row[1];
    $c++;
}

$cxn->close();

?>
```



```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select c1, c2 from tabla");

$c=1;

while ($row = $res->fetch_array())    {
    echo 'fila '.$c.' - columna 1: '.$row[c1];
    echo '<br>';
    echo 'fila '.$c.' - columna 2: '.$row[1];
    $c++;
}

$cnx->close();

?>
```

mysqli_fetch_assoc

Mediante esta función recuperamos filas y accedemos a los valores de cada campo como si de una matriz asociativa se tratase. Se toman como índices los nombres de los campos. Los nombres de campos son sensibles a mayúsculas y minúsculas.

Su sintaxis, en la forma procedural, es la siguiente:

mysqli_fetch_assoc(id_consulta);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($sql);

while ($row = mysqli_fetch_assoc($res))
{
    echo $row[X];
    echo '<br>';
    echo $row[Z];
}
```

```

        echo '<br>';
    }

    mysqli_close();

?>

```

En la forma orientada a objetos trabajamos de la siguiente manera:

```

<?php

$cxn = new mysqli("localhost", "u", "p", "bd");
$res = $cxn->query("select X, Y, Z from tabla");

while ($row = $res->fetch_assoc())    {
    echo $row[X];
    echo '<br>';
    echo $row[Z];
    echo '<br>';
}
$cxn->close();

?>

```

mysqli_fetch_field_direct

Sirve para conocer los “**metadatos**” de un campo devuelto por una consulta.

METADATO	DESCRIPCIÓN
name	Nombre de la columna.
orgname	Nombre original de la columna si se dio un alias.
table	Nombre de la tabla a la que pertenece el campo.
orgtable	Nombre original de la tabla si se dio un alias.
def	El valor por defecto para este campo.
max_length	La amplitud máxima del campo.
flags	Un entero que representa los bit bandera para el campo.
type	Tipo de dato utilizado para este campo.
decimals	Número de decimales usados.

Tabla 1. Metadatos de devueltos por *mysqli_fetch_field_direct*.

Forma de uso procedural:

mysqli_fetch_field_direct(id_resultado, indice_campo);

Donde vemos **indice_campo** se trata de la posición del campo (debemos saber que comenzando por 0) dentro de la consulta SQL.

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($con);

$md = mysqli_fetch_field_direct($res, 1);

//metadatos del campo Y:
echo $md->name;
echo $md->table;
echo $md->type;
//...

mysqli_close();

?>
```

Forma de uso orientada a objetos:

resultado_mysqli->fetch_field_direct(numero_fila);

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select X, Y, Z from tabla");

$md = $res->fetch_field_direct(0);

//metadatos del campo X:
```



```

echo $md->name;
echo $md->table;
echo $md->type;
//...

$cnx->close();

?>

```

mysqli_fetch_field

Esta función es bastante parecida a la función denominada **mysqli_fetch_field_direct**, con la singular diferencia de que puede llamarse en forma repetida, y de tal manera obtener los metadatos de todos los campos.

Forma de uso procedural:

mysqli_fetch_field (id_resultado);

```

<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($sql);

//muestra progresivamente los metadatos
//de las columnas X, Y y Z:

while ($md = mysqli_fetch_field($res))    {
    echo $md->name;
    echo $md->table;
    echo $md->type;
}

mysqli_close();

?>

```

Veamos ahora la forma de uso orientada a objetos.

resultado_mysqli->fetch_field ();

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select X, Y, Z from tabla");

//muestra progresivamente los metadatos
//de las columnas X, Y y Z:

while ($md = $res->fetch_field())    {
    echo $md->name;
    echo $md->table;
    echo $md->type;
}

$cxn->close();

?>
```

mysqli_fetch_fields

Similar a **mysqli_fetch_field**, sólo que regresa las columnas como una matriz de objetos, en lugar de regresar un objeto a la vez.

Forma de uso procedural:

mysqli_fetch_fields(id_resultado);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($con);

//muestra progresivamente los metadatos
//de las columnas X, Y y Z:

$md = mysqli_fetch_fields($res);
```

```

for ($c=0; $c<count($md); $c++)    {
    echo $md[$c]->name;
    echo $md[$c]->table;
    echo $md[$c]->type;
}

mysqli_close();

?>

```

Forma de uso orientada a objetos:

resultado_mysqli->fetch_fields();

```

<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select X, Y, Z from tabla");

//muestra progresivamente los metadatos
//de las columnas X, Y y Z:

$md = $res->fetch_fields();

for ($c=0; $c<count($md); $c++)    {
    echo $md[$c]->name;
    echo $md[$c]->table;
    echo $md[$c]->type;
}

$cnx->close();

?>

```

mysqli_fetch_lengths

Esta función regresa una matriz que contiene en su interior la longitud de las columnas de la fila actual en el resultado. A no confundir: no devuelve la capacidad del campo, sino las posiciones utilizadas.

Forma de uso procedural:

mysqli_fetch_lengths(id_resultado);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($sql);

$len = mysqli_fetch_lengths($res);

for ($c=1; $c<=count($len); $c++) {
    echo "Columna $c - longitud: $len[$c]";
}

mysqli_close();

?>
```

Forma de uso orientada a objetos:

resultado_mysqli->fetch_lengths();

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");
$res = $cxn->query("select X, Y, Z from tabla");

$len = $res->fetch_fields();

for ($c=1; $c<=count($len); $c++) {
    echo "Columna $c - longitud: $res->len[$c]";
}

$cxn->close();

?>
```

mysqli_fetch_object

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

Forma de uso procedural:

mysqli_fetch_object(id_consulta);

Se envía una consulta al servidor (con **mysqli_query**, por ejemplo) de bases de datos. El resultado devuelto se denomina **id_consulta**. Para recuperar los datos devueltos por la consulta (si es que los hay) una de las funciones que se puede utilizar es **mysqli_fetch_object** a la cual se le pasa como argumento el identificador de la consulta (en este caso **id_consulta**).

```

<?php

$sql = "select * from tabla";

$res = mysqli_query($sql);

while ($f = mysqli_fetch_object($res))    {
    echo $f->nombre_y_apellido;
    echo "<br>";
    echo $f->nacionalidad;
    echo "<br>";
}

?>

```

Los nombres de campos son sensibles a mayúsculas y minúsculas.

Forma de uso orientada a objetos:

resultado_mysqli->fetch_object();

```

<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select X, Y, Z from tabla");

```

```
while ($fila = $res->fetch_object())    {
    echo $fila->X;
    echo $fila->Y;
    echo $fila->Z;

}

$cnx->close();

?>
```

mysqli_fetch_row

Mediante esta función recuperamos filas y accedemos a los valores de cada campo como si de una matriz numérica se tratase.

Se toman como índices las posiciones de los campos en las consultas de selección. Su sintaxis, en la forma procedural, es la siguiente:

mysqli_fetch_row(id_consulta);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($con,$sql);

while ($row = mysqli_fetch_row($res))
{
    echo $row[0];           //columna X
    echo '<br>';
    echo $row[2];           //columna Z
    echo '<br>';
    echo $row[1];           //columna Y
    echo '<br>';
}

mysqli_close($con);

?>
```

En la forma orientada a objetos trabajamos de la siguiente manera:

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select X, Y, Z from tabla");

while ($row = $res->fetch_row())    {
    echo $row[0];           //columna X
    echo '<br>';
    echo $row[2];           //columna Z
    echo '<br>';
    echo $row[1];           //columna Y
    echo '<br>';
}

$cxn->close();

?>
```

mysqli_field_count

Esta función devuelve el número de columnas de la última consulta SQL. Su sintaxis, en la forma procedural, es la siguiente:

mysqli_field_count(identificador);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select A, B, C, D from tabla";
$res = mysqli_query($con);

$num = mysqli_field_count($con);
// $num = 4

mysqli_close();

?>
```


En la forma orientada a objetos trabajamos de la siguiente manera:

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select A, B, C from tabla");

$num = $cxn->field_count();
// $num = 3

$cxn->close();

?>
```

mysqli_field_seek

Similar a la función denominada **mysqli_data_seek**, sólo que en vez de posicionarse sobre una fila lo hace sobre una columna en particular.

Su sintaxis, en la forma procedural, es la siguiente:

mysqli_field_seek(id_consulta, numero_columna);

Donde **numero_columna** cuenta a partir de cero.

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($con, $sql);

//toma posicion en la segunda columna (Y)
mysqli_field_seek($res, 1);

$md = mysqli_fetch_field($res);

//metadatos de la columna Y
echo $md->name;
echo $md->table;
```

```

echo $md->type;

mysqli_close();

?>

```

En la forma orientada a objetos trabajamos de la siguiente manera:

resultado_mysqli-> field_seek ();

```

<?php

$cxn = new mysqli("localhost", "u", "p", "bd");
$res = $cxn->query("select X, Y, Z from tabla");

//toma posicion en la primer columna (X)
$res->field_seek(0);

$md = $res->fetch_field();

//metadatos de la columna X
echo $md->name;
echo $md->table;
echo $md->type;

$cxn->close();

?>

```

mysqli_free_result

Libera la memoria asociada con el resultado. Ejemplos de uso:

```

<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select X, Y, Z from tabla";
$res = mysqli_query($sql);

```

```
$md = mysqli_fetch_field($res);

echo $md->table;

mysqli_free_result($res);

mysqli_close();

?>
```

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("select X, Y, Z from tabla");

$md = $res->fetch_field();

echo $md->name;

$res->close();

$cxn->close();

?>
```

mysqli_get_client_info

Devuelve la versión del cliente MySQL, en formato de cadena de caracteres.

Forma de uso:

```
<?php

$client = mysqli_get_client_info();

echo "version cliente de MySQL: ".$client;

?>
```

mysqli_get_client_version

La función **mysqli_get_client_version** se encarga de devolver la versión del cliente MySQL, pero a diferencia de lo que ocurre con la función conocida como **mysqli_get_client_info**, devuelve el resultado en forma de entero.

Forma de uso:

```
<?php

$cli = mysqli_get_client_version();

echo "version cliente de MySQL: ".$cli;

?>
```

mysqli_get_host_info

Devuelve el tipo de conexión que está siendo utilizada.

Forma de uso:

```
<?php

$con = mysqli_connect("servidor1", "u", "p", "bd");

$i = mysqli_get_host_info($con);

echo "tipo de conexion: ".$i;

?>
```

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$i = $cxn->host_info;

echo "tipo de conexion: ".$i;

?>
```


mysqli_get_proto_info

Devuelve la versión del protocolo que está siendo utilizado.

Forma de uso:

```
<?php

$con = mysqli_connect("servidor1", "u", "p", "bd");

$i = mysqli_get_proto_info($con);

echo "protocolo: ".$i;

?>
```

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$i = $cxn->protocol_version;

echo "protocolo: ".$i;

?>
```

mysqli_get_server_info

Se encarga de devolver la versión del servidor MySQL que está siendo utilizado actualmente, en formato de cadena de caracteres.

Forma de uso:

```
<?php

$con = mysqli_connect("servidor1", "u", "p", "bd");

$i = mysqli_get_server_info($con);

echo "version del servidor: ".$i;

?>
```

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$i = $cxn->server_info;

echo "version del servidor: ".$i;

?>
```

mysqli_get_server_version

Devuelve la versión del servidor MySQL que está siendo utilizado, en formato de entero. A continuación vemos la forma de uso:

```
<?php

$con = mysqli_connect("servidor1", "u", "p", "bd");

$i = mysqli_get_server_info($con);

echo "version del servidor: ".$i;

?>
```

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$i = $cxn->server_info;

echo "version del servidor: ".$i;

?>
```

mysqli_insert_id

Esta función se encarga de devolver el valor (no el número de fila) del último campo autoincrementable ingresado, y debe llamarse inmediatamente después del ingreso. Su sintaxis es, en la forma procedural:

mysqli_insert_id(identificador);

```
<?php

$sql = "INSERT INTO tabla (c1, c2) VALUES (44, 'rio')";
mysqli_query($sql);

echo "Último valor de c1 : ".mysqli_insert_id();

//devuelve 44

?>
```

Forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "p", "bd");

$res = $cxn->query("INSERT INTO tabla (c1, c2) VALUES (44, 'rio')");

echo "Último valor de c1 : ".$cxn->insert_id;

//devuelve 44

?>
```

mysqli_more_results

Indica si uno o más resultados están disponibles de una llamada previa a la función **mysqli_multi_query**, que vemos a continuación.

mysqli_multi_query

Esta función se encarga de ejecutar una o múltiples consultas. Si hay más de una, deben estar concatenadas por un punto y coma (;).

Devuelve el valor verdadero (**true**) si se llevó a cabo con éxito, o el valor falso (**false**) en caso de que suceda algún error.

Ejemplos de la función, en la forma orientada a objetos:

```

<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$sql = "select * from tabla1";
$sql .= "select * from tabla2";

if ($cxn->multi_query($sql))    {
    do {
        if ($res = $cxn->store_result())    {
            while ($row = $res->fetch_row())
                echo $row[0];
        }
    } while ($cxn->next_result());
}

$cxn->close();

?>

```

En el ejemplo, en primer lugar preguntamos si la/s consulta/s son válida/s. Si lo son, almacenamos el resultado de la primera y lo recorremos, mostrando el valor de la primera columna. Luego pasamos a la siguiente consulta (en caso de que exista otra consulta) y repetimos el procedimiento. Veamos el mismo ejemplo, en la forma procedural:

```

<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "select * from tabla1";
$sql .= "select * from tabla2";

if (mysqli_multi_query($con, $sql))    {
    do {
        if ($res = mysqli_store_result($con))    {
            while ($row = mysqli_fetch_row($res))
                echo $row[0];
        }
    }
}

```



```
        } while ($cxn->mysqli_next_result($con));  
  
    }  
  
    mysqli_close();  
  
?>
```

El argumento **identificador** (identificador de conexión) es opcional: si se omite se toma la última conexión establecida.

mysqli_next_result

Almacena el siguiente resultado de una consulta múltiple. En caso de no haberla devuelve falso. Ver ejemplo de **mysqli_multi_query**, en este mismo **Apéndice**.

mysqli_num_fields

Obtiene el número de campos devueltos por una consulta. Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php  
  
$cxn = new mysqli("localhost", "u", "c", "bd");  
  
$res = $cxn->query("select * from tabla1");  
  
echo "campos devueltos: ".$res->field_count;  
  
$cxn->close();  
  
?>
```

Si utiliza la forma procedural:

```
<?php  
  
$con = mysqli_connect("localhost", "u", "p", "bd");  
  
$sql = "select * from tabla1";
```

```

$res = mysqli_query($sql);

echo "campos devueltos: ".mysqli_field_count($res);

mysqli_close();

?>

```

mysqli_num_rows

Similar a la función denominada **mysqli_affected_rows**, con la diferencia de que sólo trabaja con instrucciones SQL de tipo **select**.

Ejemplos de la función si utiliza la forma orientada a objetos:

```

<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$res = $cxn->query("select * from tabla1");

echo "filas devueltas: ".$res->num_rows;

$cnx->close();

?>

```

Si utiliza la forma procedural:

```

<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "select * from tabla1";

$res = mysqli_query($sql);

echo "filas devueltas: ".mysqli_num_rows($res);

```

```
mysqli_close();
```

```
?>
```

mysqli_ping

Esta función nos permite verificar si la conexión está operativa y de no estarlo intenta reconectar. Ejemplos de la función a continuación:

Si utiliza la forma orientada a objetos,

```
<?php
```

```
$cxn = new mysqli("localhost", "u", "c", "bd");
```

```
if ($cxn->ping())    {  
    echo 'conexión operativa';  
    $cxn->close();  
} else  
    echo 'problemas en la conexion';
```

```
?>
```

Si utiliza la forma procedural:

```
<?php
```

```
$con = mysqli_connect("localhost", "u", "p", "bd");
```

```
if (mysqli_ping($con))    {  
    echo 'conexión operativa';  
    mysqli_close();  
} else  
    echo 'problemas en la conexion';
```

```
?>
```

mysqli_prepare

Nos permite trabajar con consultas preparadas.

Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$buscar = "algo";

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select campo2 from tabla1 where campo1 = ?");

$cp->bind_param("s", $buscar);

$cp->execute();

$cp->bind_result($var);

$cp->fetch();

//cuando campo1 = 'algo', campo2 es igual a:
echo $var;

$cp->close();

$cnx->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$buscar = "algo";

$cp = mysqli_stmt_init();
```



```
$cp = mysqli_prepare($con, "select campo2 from tabla1 where campo1 = ?");

mysqli_stmt_bind_param($cp, "s", $buscar);

mysqli_stmt_execute($cp);

mysqli_stmt_bind_result($cp, $var);

mysqli_stmt_fetch($cp);

//cuando campo1 = 'algo', campo2 es igual a:
echo $var;

mysqli_stmt_close($cp);

mysqli_close();

?>
```

En una consulta preparada únicamente puede haber una y sólo una instrucción SQL. Los argumentos (no es necesario que haya uno sólo: puede haber más de uno) son especificados por medio de los signos ?. Los argumentos no son válidos para especificar nombres de tablas, nombres de columnas u operadores binarios.

mysqli_query

Ejecuta una instrucción SQL.

Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$sql = "insert into tabla1 values (1, 'a')";

$cxn->query($sql);

$cxn->query("insert into tabla1 values (2, 'b')");

$cxn->query("delete from tabla1");
```

```
$cxn->query("select * from tabla1");

$cnx->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$sql = "update tabla1 set codigo = 4";
$res = mysqli_query($sql);

$res = mysqli_query("select * from tabla1");

mysqli_close();

?>
```

mysqli_rollback

Ejecuta la instrucción **ROLLBACK**.

Ejemplos de uso:

objeto_mysqli->rollback;

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cnx->autocommit(FALSE);

$cxn->query("insert into tabla1 values (1, 'a')");

$cxn->commit();

$cxn->query("delete from tabla1 where codigo = 3");
```

```
$cxn->rollback();

$cnx->close();

?>
```

mysqli_rollback(identificador);

Identificador consiste en un identificador de conexión. Este argumento es opcional y en caso de que optemos por no incluirlo la función asumirá la última conexión abierta con el servidor MySQL.

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

mysqli_autocommit(FALSE);

$sql = "update tabla1 set codigo = 44 where codigo > 2";
$res = mysqli_query($sql);

mysqli_commit();

$res = mysqli_query("update tabla1 set codigo = 5");

mysqli_rollback();

mysqli_close();

?>
```

mysqli_select_db

Se trata de una función que permite seleccionar una base de datos para su uso durante un script y puede ser usada cuando, por ejemplo, deseamos mantener los datos de una conexión y sólo cambiar de base de datos.

Su sintaxis procedural es la siguiente:

mysqli_select_db("base de datos", identificador);

- **Base de datos:** nombre de la base de datos que queremos seleccionar y sobre la cual queremos realizar las consultas.
- **Identificador** consiste en un identificador de conexión. Este argumento es opcional y en caso de que optemos por no incluirlo la función asumirá la última conexión abierta con el servidor MySQL.

```
<?php

$con = mysqli_connect("168.22.22.3", "u", "p", "bd");

$sql = "update tabla1 set codigo = 4";
$res = mysqli_query($sql);

//cambiamos de base de datos, pero mantenemos la conexion
mysqli_select_db("bd2");

mysqli_close();

?>
```

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

Su sintaxis orientada a objetos es la siguiente:

objeto_mysqli->select_db("base de datos");

```
<?php

$cxn = new mysqli("localhost", "usuario", "contraseña", "bd");

$cxn->query("select * from tabla1");
echo "filas afectadas: ".$cxn->affected_rows;

//cambiamos de base de datos, pero mantenemos la conexion
$cxn->select_db("bd2");

$cxn->close();

?>
```


mysqli_stat

Devuelve información acerca del estado actual del servidor.

Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

echo $cxn->stat();

$cxn->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

echo mysqli_stat($con);

mysqli_close();

?>
```

mysqli_stmt_affected_rows

Hace lo mismo que **mysqli_affected_rows**, pero trabaja sobre consultas preparadas.

Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$buscar = "algo";

$cp = $cxn->stmt_init();
```

```

$cp = $cxn->prepare("select campo2 from tabla1 where campo1 = ?");

$cp->bind_param("s", $buscar);

$cp->execute();

echo 'filas afectadas: '.$cp->affected_rows;

$cp->close();

$cnx->close();

?>

```

Si utiliza la forma procedural:

```

<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$buscar = "algo";

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select campo2 from tabla1 where campo1 = ?");

mysqli_stmt_bind_param($cp, "s", $buscar);

mysqli_stmt_execute($cp);

echo 'filas afectadas: ';

echo mysqli_stmt_affected_rows($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>

```

mysqli_stmt_bind_param

Esta función se utiliza para enlazar parámetros de una consulta preparada con valores. Dependiendo del tipo de argumento, se especificará uno de los caracteres que podemos apreciar en la siguiente tabla:

CARÁCTER	DESCRIPCIÓN
i	Para variables tipo integer.
d	Para variables tipo double.
s	Para variables tipo string.
b	Para variables tipo blob.

Tabla 2. Caracteres disponibles para *mysqli_stmt_bind_param*.

En el siguiente código, que se encuentra bajo este párrafo, vemos un ejemplo de la función utilizando la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$var1 = 1;
$var2 = 'hola';
$var3 = 3.14;
$var4 = 'adios';

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("insert into tabla values (?, ?, ?, ?)");

$cp->bind_param('isds', $var1, $var2, $var3, $var4);

$cp->execute();

$cp->close();

$cxn->close();

?>
```

En el ejemplo anterior, al haber múltiples argumentos, agrupamos los caracteres correspondientes en una cadena, según el orden en que los pasamos (**Tabla 3**).

ORDEN	TIPO	NOMBRE DE LA VARIABLE	CADENA RESULTANTE
1	Integer	\$var1	idss
2	String	\$var2	
3	Double	\$var3	
4	String	\$var4	

Tabla 3. Tipo de cadenas.

Si utiliza la forma procedural, de la siguiente forma:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$var1 = 1;
$var2 = 'hola';
$var3 = 3.14;
$var4 = 'adios';

$cp = mysqli_stmt_init();

$cp = mysqli_prepare("insert into tabla values (?, ?, ?, ?)");

mysqli_stmt_bind_param($cp, 'isds', $var1, $var2, $var3, $var4);

mysqli_stmt_execute($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_bind_result

Se encarga de enlazar las variables pasadas como argumento, a una consulta que se encuentra preparada para almacenar los resultados obtenidos de dicha instrucción. Es similar en algún sentido a **mysqli_query**.

Veamos primero un ejemplo de la función si utiliza la forma orientada a objetos.


```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select campo1, campo2 from tabla1");

$cp->execute();

$cp->bind_result($var1, $var2);

while($cp->fetch())    {
    echo $var1;        //campo1
    echo $var2;        //campo2
}

$cp->close();

$cxn->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select campo1, campo2 from tabla1");

mysqli_stmt_execute($cp);

mysqli_bind_result($cp, $var1, $var2);

while (mysqli_stmt_fetch($cp))    {
    echo $var1;        //campo1
```

```

        echo $var2;      //campo2
    }

    mysqli_stmt_close($cp);

    mysqli_close();

?>

```

mysqli_stmt_close

Esta función se ocupa de cerrar una consulta preparada. Ejemplos de la función si utiliza la forma orientada a objetos:

```

<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("insert into t1 values (1,2)");

$cp->execute();

//cerramos la consulta preparada:
$cp->close();

$cxn->close();

?>

```

Si utiliza la forma procedural:

```

<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

```

```
$cp = mysqli_prepare($con, "insert into t1 values (1,2)");

mysqli_stmt_execute($cp);

//cerramos la consulta preparada:
mysqli_stmt_close($cp);

mysqli_close();

?>
```

Esta función devuelve **true** (verdadero) si tuvo éxito y **false** (falso) si algo falló.

mysqli_stmt_data_seek

Función similar a **mysqli_data_seek**, tiene como propósito el poder posicionarse en algún registro específico de los devueltos por una consulta preparada -que devuelva registros, claro. Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select c1, c2 from t1");

$cp->execute();

$cp->bind_result($var1, $var2);

$cp->store_result();

$cp->data_seek(2);

$cp->fetch();

echo $var1;      //c1, 3ra fila
echo $var2;      //c2, 3ra fila
```

```
$cp->close();  
  
$cnx->close();  
  
?>
```

Si utiliza la forma procedural:

```
<?php  
  
$con = mysqli_connect("localhost", "u", "p", "bd");  
  
$cp = mysqli_stmt_init();  
  
$cp = mysqli_prepare($con, "select c1, c2 from t1");  
  
mysqli_stmt_execute($cp);  
  
mysqli_bind_result($cp, $var1, $var2);  
  
mysqli_stmt_store_result($cp);  
  
mysqli_data_fetch($cp, 2);  
  
mysqli_stmt_fetch($cp);  
  
echo $var1;      //c1, 3ra fila  
echo $var2;      //c2, 3ra fila  
  
mysqli_stmt_close($cp);  
  
mysqli_close();  
  
?>
```

mysqli_stmt_errno

Devuelve el número de error cometido (si es que se cometió alguno) en una consulta preparada. Veamos el ejemplo de la función si utiliza la forma orientada a objetos.


```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select c1, c2 from t1");

$cp->execute();

echo $cp->errno;

$cp->close();

$cxn->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select c1, c2 from t1");

mysqli_stmt_execute($cp);

echo mysqli_stmt_errno($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_error

Devuelve la descripción del error cometido en una consulta preparada. Ejemplos de la función si utiliza la forma orientada a objetos y si utiliza la forma procedural:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select c1, c2 from t1");

$cp->execute();

echo $cp->error;

$cp->close();

$cxn->close();

?>
```

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select c1, c2 from t1");

mysqli_stmt_execute($cp);

echo mysqli_stmt_error($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_execute

Esta función nos permite ejecutar una consulta preparada.

Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();
$cp = $cxn->prepare("select * from tabla");
$cp->execute();
$cp->close();

$cxn->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();
$cp = mysqli_prepare($con, "select * from tabla");

mysqli_stmt_execute($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_fetch

Esta función nos brinda la posibilidad de obtener los valores devueltos por una consulta preparada y almacenarlos en las variables, pasadas como argumento a la función **mysqli_bind_result**. En la siguiente página, podemos encontrar ejemplos de la función si utiliza la forma orientada a objetos:

```

<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select A, B, C from tabla1");

$cp->execute();

$cp->bind_result($var1, $var2);

while($cp->fetch())    {
    echo $var1;        //A
    echo $var2;        //B
}

$cp->close();

$cnx->close();

?>

```

Si utiliza la forma procedural:

```

<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select A, X from tabla1");

mysqli_stmt_execute($cp);

mysqli_bind_result($cp, $var1, $var2);

while (mysqli_stmt_fetch($cp))    {

```



```
        echo $var1;      //A
        echo $var2;      //X
    }

    mysqli_stmt_close($cp);

    mysqli_close();

?>
```

Esta función arrojará **true** (verdadero) en el caso de que la consulta haya devuelto algún dato, regresará **false** (falso) si algo falló, y devolverá **NULL** si no hay más filas o si la consulta no devolvió ningún dato.

mysqli_stmt_free_result

Libera la memoria asociada con el resultado devuelto por una consulta preparada. Si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$sql = "select * from tabla1";

$cp = $cxn->prepare($sql);

$cp->execute();

$cp->store_result();

$cp->free_result();

$cxn->close();

?>
```

Si utilizamos la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select A from tabla1");

mysqli_stmt_execute($cp);

mysqli_stmt_store_result($cp);

mysqli_stmt_free_result($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_init

Inicializa una consulta preparada y devuelve un objeto que nos servirá para trabajar con la función **mysqli_stmt_prepare**. Ejemplos si utiliza la forma orientada a objetos:

```
<?php

$cnx = new mysqli("localhost", "u", "c", "bd");

$cp = $cnx->stmt_init();

$cp = $cnx->prepare("insert into t value (3, 5)");

$cp->execute();

$cp->close();

$cnx->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");
$stmt = mysqli_stmt_init();
$stmt = mysqli_prepare($con, "insert into t value (3, 5)");

mysqli_stmt_execute($stmt);

mysqli_stmt_close($stmt);

mysqli_close();

?>
```

mysqli_stmt_num_rows

Representa el número de filas devueltas por una consulta preparada. Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cnx = new mysqli("localhost", "u", "c", "bd");

$stmt = $cnx->stmt_init();

$stmt = $cnx->prepare("select A, B, C from tabla1");

$stmt->execute();

$stmt->store_result();

echo "numero de filas: ".$stmt->num_rows;

$stmt->close();

$cnx->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select A from tabla1");

mysqli_stmt_execute($cp);

mysqli_stmt_store_result($cp);

echo "numero de filas: ".mysqli_num_rows($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_param_count

Devuelve el número de argumentos pasados a una consulta preparada. Ejemplos de la función si utiliza la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$var1 = 1;
$var2 = 2;
$var3 = 3;
$var4 = 4;

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("insert into tabla values (?, ?, ?, ?)");
```



```
echo "numero de argumentos: ".$scp->param_count;

$scp->bind_param('iiii', $var1, $var2, $var3, $var4);

$scp->execute();

$scp->close();

$cnx->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$var1 = 1;
$var2 = 2;
$var3 = 3;
$var4 = 4;

$scp = mysqli_stmt_init();

$scp = mysqli_prepare("insert into tabla values (?, ?, ?, ?)");

mysqli_stmt_bind_param($scp, 'iiii', $var1, $var2, $var3, $var4);

echo "numero de argumentos: ";
echo mysqli_stmt_param_count($scp);

mysqli_stmt_execute($scp);

mysqli_stmt_close($scp);

mysqli_close();

?>
```

mysqli_stmt_prepare

Después de crear e inicializar la consulta preparada con **mysqli_stmt_init**, la preparamos con **mysqli_stmt_prepare** para su posterior ejecución.

Ejemplos de la función si utilizamos la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("delete from t1");

$cp->execute();

$cp->close();

$cnx->close();

?>
```

Si utiliza la forma procedural:

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare("delete from t1");

mysqli_stmt_execute($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

mysqli_stmt_result_metadata

Similar a **fetch_field_direct**, esta función nos permite conocer los llamados metadatos de un campo devuelto por una consulta preparada. Algunos de los disponibles son:

METADATO	DESCRIPCIÓN
name	Nombre de la columna.
orgname	Nombre original de la columna si se dio un alias.
table	Nombre de la tabla a la que pertenece el campo.
orgtable	Nombre original de la tabla si se dio un alias
def	El valor por defecto para este campo.
max_length	La amplitud máxima del campo.
flags	Un entero que representa los bit bandera para el campo.
type	Tipo de dato utilizado para este campo.
decimals	Número de decimales usados.

Tabla 4. Metadatos de *mysqli_stmt_result_metadata*.

Forma de uso procedural:

mysqli_stmt_result_metadata(consulta_preparada);

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();
$cp = mysqli_prepare($con, "select A from tabla1");

mysqli_stmt_execute($cp);

$res = mysqli_stmt_result_metadata($cp);

$columna = mysqli_fetch_field($res);

echo "nombre columna: ".$columna->name;

mysqli_stmt_close($cp);

mysqli_close();

?>
```

Forma de uso orientada a objetos:

```
$metadatos = consulta_preparada->result_metadata();
```

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$cp = $cxn->prepare("select B from T1");

$cp->execute();

$res = $cp->result_metadata();

$columna = $res->fetch_field();

echo "nombre columna: ".$columna->name;

$cp->close();

$cnx->close();

?>
```

mysqli_stmt_store_result

Esta función se encarga de transferir a la máquina cliente un set de resultados, obtenido a través de una consulta preparada. Esta consulta, obviamente, debe ser de las que pueden recibir valores (**SELECT**, **SHOW**, **DESCRIBE**, **EXPLAIN**).

Ejemplos de la función si utilizamos la forma orientada a objetos:

```
<?php

$cxn = new mysqli("localhost", "u", "c", "bd");

$cp = $cxn->stmt_init();

$sql = "select * from tabla1";
```



```
$cp = $cxn->prepare($sql);

$cp->execute();

//almacenamos los datos en la maquina cliente
$cp->store_result();

$cxn->close();

?>
```

```
<?php

$con = mysqli_connect("localhost", "u", "p", "bd");

$cp = mysqli_stmt_init();

$cp = mysqli_prepare($con, "select A from tabla1");

mysqli_stmt_execute($cp);

//almacenamos los datos en la maquina cliente
mysqli_stmt_store_result($cp);

mysqli_stmt_close($cp);

mysqli_close();

?>
```

Servicios al lector

A continuación, explicaremos cómo realizar las tareas de instalación, configuración y testeo sobre distintas plataformas de Apache, PHP y MySQL.

Apache	410
PHP	412
MySQL	412
Configuración	413
Instalación sobre Linux	415
Apache	415
PHP	416
MySQL	417
Configuración	419
Información acerca del funcionamiento	420
Configuración de MySQL a través del archivo php.ini	425
Configuración del archivo my.cnf	428
Índice temático	429

APACHE

Veamos primero la instalación sobre Windows. Existen dos opciones: descargar la versión de **Apache** autoinstalable (también llamada **binary version**, generalmente, un archivo **.MSI**) o descargar el código fuente (*source code*) y compilarlo con, por ejemplo, **Microsoft Visual C++**. Aquí optaremos por la primera alternativa por tratarse de la más segura y sencilla.

En definitiva, deberíamos descargar un archivo de nombre **apache_1_3_xx-win32-no_src.msi** (donde xx es el número que completa la versión).

La versión de **Apache 2.x** está orientada para trabajar con sistemas operativos Windows NT, 2000, XP y .Net Server 2003, lo que significa que su funcionamiento correcto no está asegurado si lo instalamos sobre Windows 95, 98 o ME, incluso posiblemente no funcione.

Si utilizamos alguno de estos sistemas debemos considerar la opción de instalar Apache versión 1.3.x.

De acuerdo al sistema operativo en el que se realice la instalación, deberemos descargar el **Instalador de Microsoft** (*Microsoft Installer*), preferentemente la última versión.

Para Windows 9.x (o sea 95 o 98 o ME) podremos descargarlo desde

www.microsoft.com/downloads/release.asp?ReleaseID=32831

y para Windows NT 4.0 y 2000 desde

www.microsoft.com/downloads/release.asp?ReleaseID=32832

Para Windows XP no es necesario.

Tengamos en cuenta que tal vez no haga falta, porque algunos sistemas operativos (como 2000 y ME) lo traen instalado por defecto.

Asumimos que se instalará Apache para uso en desarrollos locales, o sea en nuestra máquina, con el fin de probar nuestros programas.

Luego de descargar Apache, podremos comenzar a instalarlo haciendo clic sobre el archivo. Luego, el mismo programa se encargará de hacernos algunas preguntas referidas a la configuración del servidor.

Tengamos en cuenta que todos los datos que ingresemos en esta etapa podrán ser modificados luego, ya veremos cómo, en el archivo **http.conf** (ubicado dentro de la carpeta **conf** del directorio en donde instalamos Apache; por ejemplo, **c:\apache group\apache\conf**).

Si instalamos Apache 2.x nos preguntará también si deseamos que el servidor responda en el **puerto 80** para todos los usuarios, o si deseamos que sólo se active para el usuario actual en el puerto **8080**, cuando se inicie manualmente. Lo normal es que lo activemos para todos los usuarios.

Apache se instala normalmente —si no se le dice lo contrario— en la carpeta **c:\archivos de programa\apache group\apache**.

- **Para poner en marcha el servidor:** si todo funcionó bien podemos iniciar el servidor desde **Inicio/Programas/Apache HTTP Server/Start Apache in Console** (para **Apache 1.x**) o desde **Inicio/Programas/Apache HTTP Server 2.0.44/Control Apache Server/Start Apache in Console** (para **Apache 2.x**).

Otra forma de hacerlo es desde una consola DOS, posicionarnos en el directorio de Apache, utilizando el siguiente comando:

```
apache -k start
```

En ambos casos, veremos que se muestra un mensaje que nos dice que Apache está activo, que está corriendo.

- **Para detener el servidor:** podemos simplemente cerrar la ventana que se muestra luego de iniciar el servidor (o presionar **CONTROL + C**), o también desde una consola DOS, posicionarnos en el directorio de Apache, utilizando el siguiente comando:

```
apache -k stop
```

Prestemos atención a que para detener el servidor utilizando esta última opción, deberemos abrir otra ventana en modo DOS y tipear el comando descrito anteriormente. También podemos usar **apache -k shutdown**.

- **Para reiniciar el servidor:** Podemos reiniciar Apache desde una consola DOS, para ello debemos posicionarnos en el directorio de Apache, por medio del comando que vemos a continuación:

```
apache -k restart
```

Los comandos **apache -k ...** están disponibles desde Apache versión 1.3.3 en adelante.

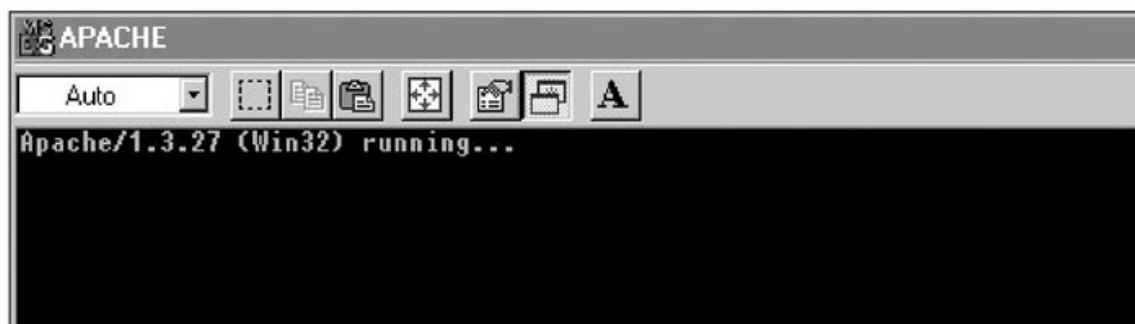


Figura 1. Pantalla indicando que Apache está activo.

Deberá poner sus páginas (.HTML, .PHP, otras) bajo la carpeta **htdocs** (normalmente, dentro del directorio Apache).

Para desinstalar Apache se puede ir al **Panel de control/Agregar o Quitar Programas** y seleccionar **Apache**.

PHP

Para instalar PHP, debemos bajarlo desde **www.php.net**. Luego, descomprimos el archivo y copiamos su contenido (todos los archivos y carpetas) a la carpeta **c:\php** (si no existe, la creamos). Si instalamos Apache versión 2.x, necesitaremos PHP 4.3.0 o superior.

MYSQL

Para la instalación de MySQL deberemos descargar la versión que más se acerque a nuestras necesidades desde **www.mysql.com** y descomprimir el archivo que será del tipo **mysql-x.x.x-win.zip** a una carpeta temporal donde **x.x.x** es la versión. No descarguemos archivos de tipo **mysql-x.x.x-win-src.zip** ya que contienen el código fuente de MySQL. Luego, ejecutamos el programa de instalación (probablemente **setup.exe**) y seguimos las instrucciones.

Entre las preguntas que nos hará el programa de instalación podemos destacar: lugar en donde deseamos que se instale el programa (seleccionamos **c:\mysql**, siempre y cuando no tengamos otra versión instalada en ese directorio) y nombre de usuario y contraseña del administrador.

Si no nos pide un nombre de usuario y contraseña (recordemos que por defecto en la instalación el único usuario se llama **root** y viene sin clave y con todos los permisos de ejecución), se la asignamos. Para esto abrimos una ventana DOS e ingresamos lo siguiente:

Reemplace 'contraseña' por una clave elegida por usted. Recuerdela.

Se supone que 'mysql' es el directorio en donde instalo MySQL

```
C:\> cd mysql\bin
```

```
C:\mysql\bin> mysqld  
C:\mysql\bin> mysqladmin -u root -pcontraseña
```

Recordemos que el usuario **root** tiene todos los permisos de ejecución. Luego, se podrán crear más usuarios accediendo como **root**.

Configuración

Veamos ahora los pasos para realizar la configuración.

Opciones del archivo httpd.conf

Debemos agregar las siguientes líneas:

```
ScriptAlias /php/ "c:/php/"  
AddType application/x-httpd-php .php  
Action application/x-httpd-php "/php/php.exe"
```

Otras opciones importantes de este archivo, también válidas para **Linux**, son:

- **ServerRoot**: ruta al directorio de instalación de Apache. Generalmente, en Windows es C:/Archivos de programa/Apache Group/Apache.
- **KeepAlive**: determina si se utilizarán conexiones persistentes (es decir, no establecer una conexión nueva ante cada petición de un usuario).
- **Listen**: puerto que se utilizará para atender las peticiones (por defecto será el 80). Otra opción que se puede dar aquí es la de establecer qué direcciones **IP** aceptará. Por ejemplo:

```
Listen 8080  
Listen 179.179.255.1:80  
Listen 179.179.255.3:8080
```

- **LoadModule**: para definir qué módulos –extensiones– cargar.
- **ServerAdmin**: dirección de correo electrónico del administrador del servidor.
- **ServerName**: nombre del servidor. Si está trabajando de forma local puede escribir aquí **localhost** ó **127.0.0.1**. Por ejemplo:

```
ServerName localhost
```

También podemos incluir el puerto de conexión:

```
ServerName localhost:80
```

- **DocumentRoot:** directorio raíz desde el cual se podrán acceder los documentos a través de un navegador. Por defecto en Windows es:

```
DocumentRoot "C:/Archivos de programa/Apache Group/Apache/htdocs"
```

- **DirectoryIndex:** archivo por defecto que se buscará en cada directorio.

```
DirectoryIndex index.html index.php index.htm
```

En caso de que en la carpeta **htdocs** tengamos un archivo llamado **index.php**, al acceder a **http://localhost/** nos mostrará automáticamente la página **index.php**.

El orden de los archivos determinará la prioridad de los mismos en aquellos casos en los cuales se encuentre más de uno en la misma carpeta con esos nombres.

Aparte de configurar el servidor desde el archivo **httpd.conf**, podemos hacerlo desde un fichero con el nombre **.htaccess**, que se puede hallar dentro de cualquier directorio dentro del directorio **DocumentRoot**. Dentro de cada uno de estos ficheros, se pueden ubicar las directivas de configuración del archivo **httpd.conf**.

La diferencia radica en que los valores de las directivas que se encuentran en el interior de un fichero **.htaccess** prevalecen frente a los valores de configuración especificados dentro del fichero **httpd.conf**. Por ejemplo, la directiva **DirectoryIndex** se puede encontrar dentro de un fichero **.htaccess**.

Opciones del archivo php.ini

Debemos buscar un archivo dentro del directorio en donde se instaló PHP, cuyo nombre es **php.ini-dist**. Luego, lo copiamos al directorio de Windows –por lo general, es **C:\Windows**–y lo renombramos como **php.ini**.

Más adelante, en esta sección, veremos varias directivas de configuración del archivo **php.ini**, pero las imprescindibles para poner en funcionamiento y coordinar el servidor web Apache con PHP son las siguientes:

```
//indicar la ruta al directorio de extensiones
extension_dir = "c:\php\extensions"
```


INSTALACIÓN SOBRE LINUX

Antes de comenzar a planear la instalación, debemos verificar que no tenemos instalado en nuestro sistema **Linux** algunas o todas estas herramientas, ya que es muy común que en la mayoría de las distribuciones vengán incorporadas.

APACHE

Una instalación típica de Apache sería la siguiente:

```
$ cd /usr/local
$ tar -zxvf apache-1.3.x.tar.gz
$ ln -s /usr/local/apache-1.3.x /usr/local/apache
$ cd /usr/local/apache
$ ./configure --prefix=/usr/local/apache
$ make
```

```
$ make install
$ cp /usr/local/apache/bin/apachectl /etc/rc.d/init.d/apache
$ /etc/rc.d/init.d/apache start
```

Si todo funcionó bien, ingresamos a la dirección **http://localhost/** y veremos un mensaje que confirma el funcionamiento de Apache.

PHP

Instalamos PHP luego de haber instalado Apache y MySQL. Descargamos PHP desde **www.php.net**. Descomprimos y desempaquetamos el archivo a una carpeta, por ejemplo **/usr/local/php** (si no existe, la creamos).

En el siguiente código, suponemos que la carpeta **/usr/local/php** existe y que allí está el archivo **php-xxx.tar.gz**:

```
$ cd /usr/local/php
```



```
$ gunzip php-xxx.tar.gz
$ tar -xvf php-xxx.tar
```

Luego, instalamos y configuramos PHP:

```
$ ./configure --with-mysql=/usr/local/mysql \
--with-apache=/usr/local/apache \
--enable-track-vars
$ make
$ make install
$ cp php.ini-dist /usr/local/lib/php.ini
```

MYSQL

Para la instalación de MySQL, deberemos descargar la versión que más se acerque a nuestras necesidades desde www.mysql.com, y descomprimir el archivo a una carpeta temporal, que será del tipo **mysql-x.x.x-win.zip**, donde **x.x.x** es la versión. Copiamos el archivo en un directorio –por ejemplo, **/usr/local**. Se debe crear un directorio –por ejemplo, **/usr/local/mysql**. Allí será instalado MySQL. Recordemos que debemos tener permiso para crear archivos en **/usr/local**.

Creamos un grupo y un usuario llamados **mysql**, de la siguiente forma:

```
$ groupadd mysql
$ useradd -g mysql mysql
```

Los nombres de los comandos pueden variar de distribución en distribución. Podrían llamarse, por ejemplo, **addgroup** y **adduser** respectivamente.

Ingresemos al directorio donde está el archivo de instalación:

```
$ cd /usr/local
```

A continuación, lo próximo que tendremos que hacer será descomprimir el archivo de instalación en un directorio llamado **mysql-x.x.x-OS** (con el nombre exacto del archivo que habíamos descargado anteriormente), y luego creamos un enlace sim-

bólico. Por medio de este enlace, lograremos acceder más fácilmente al directorio de instalación (accederemos como `/usr/local/mysql`).

```
$ gunzip < /usr/local/mysql-x.x.x-OS.tar.gz | tar xvf -  
$ ln -s /usr/local/mysql-x.x.x-OS mysql
```

Ingresamos al directorio de instalación:

```
$ cd mysql
```

Luego, compilamos e instalamos MySQL:

```
$ ./configure --without-debug --prefix=/usr/local/mysql  
$ make  
$ make install  
$ cp /usr/local/mysql/support-files/mysql.server /etc/rc.d/init.d/mysql  
$ chmod 755 /etc/rc.d/init.d/mysql
```

Debemos crear las tablas necesarias para que MySQL funcione correctamente:

```
$ scripts/mysql_install_db
```

Ahora, modificamos los permisos de los distintos archivos y directorios creados:

```
$ chown -R root /usr/local/mysql/.  
$ chown -R mysql /usr/local/mysql/data  
$ chgrp -R mysql /usr/local/mysql/.
```

Para iniciar el servidor, tipeamos:

```
$ bin/safe_mysqld --user=mysql &  
//O también:  
$ /etc/rc.d/init.d/mysql start/etc/rc.d/init.d/mysql start
```

Asignamos el **password del administrador (root)** de MySQL:

```
$ /usr/local/mysql/bin/mysqladmin -u root password "clave"
```

Configuración

Y ahora, los pasos para realizar la configuración

Opciones del archivo **httpd.conf**

Debemos agregar las siguientes líneas:

```
ScriptAlias /php/ "usr/local/php/"  
AddType application/x-httpd-php .php  
Action application/x-httpd-php "/php/php"
```

COMPROBAR QUE TODO FUNCIONA CORRECTAMENTE

Guardamos el siguiente código bajo el nombre de **ejemplo1.php** en la carpeta **htdocs** de nuestro servidor web. Iniciamos Apache y MySQL –como se dijo anteriormente. En segundo término, debemos abrir el navegador e ingresamos a la dirección donde se encuentra nuestro ejemplo (**http://localhost/ejemplo1.php**):

```
<?php  
  
//configure los datos correctos:  
$servidor = "localhost";  
$usuario = "root";  
$password = "";  
  
//se establece una conexion con MySQL  
$link = mysql_connect($servidor, $usuario, $password)  
    or die("Error en la conexion");  
echo 'Mysql – PHP – Apache : funcionando !';  
  
?>
```

Si todo funcionó bien, deberemos obtener el siguiente mensaje en el navegador:

```
Mysql - PHP - Apache : funcionando !
```

INFORMACIÓN ACERCA DEL FUNCIONAMIENTO

Un recorrido por algunas de las funciones que nos provee PHP para conocer diversas informaciones y controlar el estado de estas dos herramientas.

mysql_stat

Muestra estadísticas referidas al servidor. Algunas de ellas son:

- Tiempo de funcionamiento del servidor desde el último arranque.
- Consultas. Consultas por segundo.
- Número de tablas abiertas.

Esta función sólo muestra algunas estadísticas acerca del funcionamiento del servidor de bases de datos. Para un listado completo se puede utilizar el comando **show status** desde el **prompt** de MySQL.

```
mysql> show status;
```

mysql_get_client_info

Esta función nos brinda la posibilidad de conocer la versión del cliente de MySQL que estamos utilizando.

No debemos confundir esta función con la que habíamos visto anteriormente, llamada **version**, la cual se podía implementar directamente desde el prompt de MySQL y que se utiliza para conocer la versión del servidor MySQL al que nos estamos conectando. En el código que se encuentra a continuación, podemos apreciar un ejemplo de la función **mysql_get_client_info**:

```
<?php
echo mysql_get_client_info();
//la salida dependera de la version instalada en su sistema.
?>
```


mysql_get_host_info

Devuelve el tipo de conexión (nombre del servidor con el cual se estableció la conexión y, además, el protocolo utilizado).

Esta función recibe como argumento opcional un identificador de conexión. Si se omite, se tomará por defecto la última conexión abierta. Su sintaxis es:

```
mysql_get_host_info(identificador);
```

Por ejemplo:

```
<?php
mysql_connect("localhost", "u", "p");
echo "Nombre del servidor: ".mysql_get_host_info();
//la salida dependera de su situacion particular. Podria ser "localhost
  via TCP/IP"
?>
```

mysql_get_proto_info

Recibe como argumento un identificador de conexión, y devuelve la versión del protocolo que se está usando en dicha conexión. Su forma de uso es:

```
<?php
mysql_connect("localhost", "u", "p");
echo "Version del protocolo: ".mysql_get_proto_info();
?>
```

mysql_get_server_info

Esta función nos permite conocer la versión del servidor de MySQL al que estamos conectando. En capítulos anteriores vimos una función idéntica a ésta, que se podía utilizar desde el **prompt** de MySQL llamada **version**.

```
SELECT VERSION();
```

Ejemplo de **mysql_get_server_info**:

```
<?php
echo mysql_get_server_info();
```

```
//la salida dependera del servidor al que este conectando.  
?>
```

phpinfo

Esta función nos brinda gran cantidad de información acerca de aspectos variados referidos a PHP y sus extensiones. Entre otros:

- Versión de PHP.
- Equipo en donde se encuentra instalado PHP (nombre, sistema operativo).
- Lugar en donde se encuentra instalado el archivo **php.ini**.
- Personas que participaron en el desarrollo de PHP.
- Ubicación del directorio de extensiones.
- Extensiones disponibles y habilitadas.
- Ubicación del directorio raíz del servidor web.
- Información acerca de la licencia.
- Y muchísima más información,

Si llamamos a esta función sin argumentos, como en la página siguiente:

```
<?php  
  
echo phpinfo();  
  
?>
```

Nos devolverá todos los datos posibles. También, existe la posibilidad de pasarle argumentos, para reducir la información por mostrar y obtener sólo lo que nos interesa:

ARGUMENTO	EQUIVALENTE	DESCRIPCIÓN
INFO_GENERAL	1	Información general acerca de PHP.
INFO_CREDITS	2	Participantes del desarrollo de PHP y sus extensiones.
INFO_CONFIGURATION	4	Valores de opciones de configuración.
INFO_MODULES	8	Módulos cargados y sus respectivos parámetros.
INFO_ENVIRONMENT	16	Variables de entorno (las que se guardan en el array \$_ENV).
INFO_VARIABLES	32	Variables predefinidas.
INFO_LICENSE	64	Información de la licencia.
INFO_ALL	-1	Muestra toda la información disponible. Es como si no se le pasaran argumentos.

Tabla 1. Argumentos de la función *phpinfo*.

No olvidemos escribir los argumentos en mayúsculas.
Por ejemplo:

```
<?php
echo phpinfo(8); //identica a phpinfo(INFO_MODULES)
//devuelve informacion general acerca de los modulos
//extensiones- habilitados
?>
```

CONFIGURACIÓN DE PHP A TRAVÉS DEL ARCHIVO PHP.INI

Ya hemos visto anteriormente, algunas directivas de configuración accesibles desde el archivo **php.ini**. A continuación, tendremos un panorama mucho más amplio de las distintas opciones existentes.

short_open_tag

Admite los valores **On** y **Off**. Si está en **On** interpreta todo lo que esté encerrado dentro de los tags **<?** y **?>** como código PHP. Si está en **Off**, toma sólo los tags **<?php** y **?>**. Al momento de distribuir nuestro código, tengamos en cuenta que probablemente esta opción no esté en **On**, por lo que podrían ocurrir errores.

disable_functions

Permite inhabilitar ciertas funciones, principalmente, por cuestiones de seguridad. Las funciones se ponen una detrás de la otra separadas por coma.

max_execution_time

Otra opción que podemos definir es la del tiempo máximo de ejecución de un script. Este tiempo se mide en segundos.

```
<?php

while ($var = 10) {
    echo 'este bucle nunca termina ';
}
?>
```

El código anterior, nos daría como resultado, luego de pasado el tiempo máximo, un mensaje de error como el que sigue:

```
Fatal error: Maximum execution time of xx seconds exceeded in c:\archivos
de programa\apache group\apache\htdocs\nombre_pagina.php on line 10
```

Poner **max_execution_time = 0** equivale a que no haya tiempo máximo de ejecución.

Importante: en consultas a bases con grandes cantidades de información, puede suceder que se sobrepase el tiempo máximo sin que haya errores en el código.

include_path

Cuando usamos las funciones **include()** o **require()** desde PHP y no definimos la ruta completa al archivo que queremos incluir, PHP lo buscará en los directorios que pongamos en esta opción, separados por punto y coma. Por ejemplo:

```
include_path = ".;c:\php\includes;c:\php\pear"
```

Nombre archivo: **ejemplo1.php**.

```
<?php
Include('mis_funciones.txt');
?>
```

Aquí PHP buscará el archivo **mis_funciones.txt** en:

- El directorio en donde se encuentra guardado el archivo **ejemplo1.php**.
- El directorio **c:\php\includes**.
- El directorio **c:\php\pear**.

Configuración de MySQL a través del archivo php.ini

Luego de ver las opciones de configuración para PHP, trataremos las opciones disponibles para MySQL. Tengamos en cuenta que al configurar MySQL o PHP a través del archivo **php.ini**, estos cambios probablemente –aunque se pueden evitar, ya veremos cómo– tendrán repercusión sobre todos los usuarios de nuestro sistema. Por esto, es conveniente no pensar las características de cada sistema en detalle al momento de configurar este archivo, sino tener una visión general.

mysql.allow_persistent

Aquí se configura la opción de permitir o no que MySQL acepte conexiones persistentes, como vimos al tratar la función **mysql_pconnect**.

Se le puede asignar dos valores: **On** (permitir las conexiones persistentes) y **Off** (no permitir las conexiones persistentes).

mysql.max_persistent

Permite configurar el número máximo de conexiones persistentes simultáneas a un servidor MySQL.

Por defecto es **-1** (**mysql.max_persistent = -1**), lo que significa que no hay límite en el número de conexiones persistentes.

mysql.max_links

Permite configurar el número máximo de conexiones simultáneas (persistentes y no persistentes) a un servidor MySQL.

Por defecto es **-1** (**mysql.max_links = -1**), lo que significa que no hay límite en el número de conexiones.

mysql.default_port

Cuando se utiliza alguna de las funciones provistas por PHP para establecer una conexión a un servidor de bases de datos, uno de los argumentos opcionales es el puerto de conexión.

En algunos casos, este valor se cargará desde esta línea del **php.ini**.

```
//ejemplo de configuracion del puerto  
mysql.default_port = 80
```

mysql.default_socket

Si utilizamos sockets para establecer conexiones locales, debemos ingresar aquí su nombre por defecto.

mysql.default_host

Si al intentar establecer una conexión desde PHP, omitimos el argumento **servidor**, se tomará por defecto el que figure en esta línea.

```
//ejemplo de configuracion del servidor  
mysql.default_host = "168.23.2.1"
```

mysql.default_user

Si al intentar establecer una conexión desde PHP omitimos el argumento **nombre de usuario**, se tomará por defecto el que figure en esta línea. Veamos el código:

```
//ejemplo de configuracion del usuario  
mysql.default_user = "nombre"
```

mysql.default_password

Si al intentar establecer una conexión desde PHP omitimos el argumento **password**, se tomará por defecto el que figure en esta línea.

```
//ejemplo de configuracion de la clave de usuario  
mysql.default_password = "clave"
```

Por cuestiones de seguridad no se recomienda completar esta línea.

mysql.connect_timeout

Cuando se intenta establecer una conexión, PHP esperará un tiempo prudencial definido, justamente, en esta línea (el tiempo está en segundos). Si se pone **-1** no habrá tiempo máximo de conexión.

```
//ejemplo: se esperaran como maximo 60 segundos  
mysql.connect_timeout = 60
```

No confundamos esta línea de configuración con la **max_execution_time**, explicada anteriormente. Debemos analizar las diferencias existentes entre cada una y tener en cuenta que un script puede demorar su finalización por, entre otras razones, el intento de establecer una conexión.

Función ini_set

Esta función establece momentáneamente algunas opciones de configuración. Al terminar el script desde el cual se llamó a la función, la opción volverá a tener su valor predeterminado. Su sintaxis es:

```
ini_set (opcion, nuevo_valor);
```

Esta función devuelve a su vez el valor original de la opción.

Podremos encontrar una explicación detallada de las opciones disponibles en el manual de PHP, accesible desde **www.php.net**.

Forma de uso:

```
<?php
$var = ini_set("error_reporting", "E_ALL");
?>
```

CONFIGURACIÓN DEL ARCHIVO MY.CNF

Desde la versión 3.22 se pueden configurar ciertos parámetros de MySQL desde el archivo **my.cnf**. El archivo **my.cnf** puede estar ubicado en diferentes lugares.

Si usamos un sistema Linux, MySQL buscará este archivo en los siguientes lugares:

- **/etc/my.cnf**
- **DATA/my.cnf** (DATA es el directorio de datos).
- **~/my.cnf** (directorio en donde se está ejecutando actualmente el script).

Si utilizamos Windows, MySQL buscará este archivo en los siguientes lugares:

- **WINDOWS/my.cnf** (directorio del sistema. Normalmente, c:\windows).
- **C:\my.cnf**

El orden de prioridades es de abajo hacia arriba, es decir que el archivo **my.cnf** del último directorio prevalecerá sobre el del primero.

Esta directiva es útil cuando queremos configurar de forma particular las opciones para un usuario, sin modificarlas para todos.

MySQL trae distintos archivos **.cnf**. Generalmente, se recomienda tomarlos como base para crear los nuestros.

ÍNDICE TEMÁTICO

A

Administración	246
ADODB	273
Apache	400, 405
Arquitectura	32, 262
autocommit	220
Autoincrementables	170

B

Backup	230
Bases de datos	25
BDB	222
BerkeleyDB	86
Bibliotecas incorporadas	26
Bloqueos	223

C

Cadenas de caracteres	73, 87
Campos numéricos	98
Capas de abstracción	262
Claves candidatas	45
Claves primarias	44
Conectar PHP con MySQL	130
Consultas de selección	152
Conversión	113
Creación de base de datos	135
Creación de tablas	138

D

Datos redundantes	57
Delete	172
Diseño de la Base de Datos	42
display_errors	193
display_startup_errors	194

E

Ejecución de sentencias	133
error_append_string	204

error_log	196
error_prepend_string	203
error_reporting	190
Esquemas	46
Estadísticas	114
Excepciones	213
Extensiones	23

G

Gestión de privilegios	257
grant	253

H

heap	85
html_errors	201

I

ignore_repeated_errors	197
ignore_repeated_source	199
INNodb	85, 212, 222
Insert	170
Integridad referencial	62
ISAM	81

L

Licencia de uso	17, 71
Linux	405
log_errors	194
log_errors_max_len	196

M

merge	83
Modelo de datos	32
Modelo de red	34
Modelo jerárquico	33
Modelo orientado a objetos	36
Modelo relacional	35
Monitor de MySQL	123

Multitabla	152
MyISAM	82
MySQL	70, 402
mysql_affected_rows	167
mysql_change_user	258
mysql_close	146
mysql_connect	130
mysql_create_db	135
mysql_data_seek	164
mysql_db_query	133
mysql_errno	207
mysql_error	206
mysql_fetch_array	153
mysql_fetch_assoc	160
mysql_fetch_field	163
mysql_fetch_object	158
mysql_fetch_row	158
mysql_insert_id	171
mysql_list_dbs	141
mysql_list_tables	143
mysql_num_fields	169
mysql_num_rows	166
mysql_pconnect	132
mysql_query	134
mysql_select_db	137
MySQLDump	231
MySQLi	334

N

Normalización	56
Numéricos	76

O

Operaciones relacionales	37
Operadores de comparación	114
Operadores lógicos	115
Optimización de tablas	241

P

Pear DB	262
perror	212
PHP	14, 402
Procesos almacenados	117

R

Recuperación ante fallos	235
Register Globals	208
Registros	164
Relación	43
Reparación de tablas	237
revoke	253

S

Seguridad	230
Seguridad física	243
Selección de una base de datos	137
Selección simple	152
Sistema de privilegios	247
Subconsultas	67, 152

T

Tablas	42, 80
Tablas, tipos de relaciones	52
Tablas transaccionales	221
Tipos de datos	73
track_errors	200
Transacciones	

U

Update	173
user, tabla	248

V

Versiones	18, 73
Vistas	46

CLAVES PARA COMPRAR UN LIBRO DE COMPUTACIÓN

1 SOBRE EL AUTOR Y LA EDITORIAL

Revise que haya un cuadro "sobre el autor", en el que se informe sobre su experiencia en el tema. En cuanto a la editorial, es conveniente que sea especializada en computación.

2 PRESTE ATENCIÓN AL DISEÑO

Compruebe que el libro tenga guías visuales, explicaciones paso a paso, recuadros con información adicional y gran cantidad de pantallas. Su lectura será más ágil y atractiva que la de un libro de puro texto.

3 COMPARE PRECIOS

Suele haber grandes diferencias de precio entre libros del mismo tema; si no tiene el valor en tapa, pregunte y compare.

4 ¿TIENE VALORES AGREGADOS?

Desde un sitio exclusivo en la Red hasta un CD-ROM, desde un Servicio de Atención al Lector hasta la posibilidad de leer el sumario en la Web para evaluar con tranquilidad la compra, o la presencia de adecuados índices temáticos, todo suma al valor de un buen libro.

5 VERIFIQUE EL IDIOMA

No sólo el del texto; también revise que las pantallas incluidas en el libro estén en el mismo idioma del programa que usted utiliza.

6 REVISE LA FECHA DE PUBLICACIÓN

Está en letra pequeña en las primeras páginas; si es un libro traducido, la que vale es la fecha de la edición original.



usershop.redusers.com

VISITE NUESTRO SITIO WEB

» Vea información más detallada sobre cada libro de este catálogo.

» Obtenga un capítulo gratuito para evaluar la posible compra de un ejemplar.

» Conozca qué opinaron otros lectores.

» Compre los libros sin moverse de su casa y con importantes descuentos.

» Publique su comentario sobre el libro que leyó.

» Manténgase informado acerca de las últimas novedades y los próximos lanzamientos.

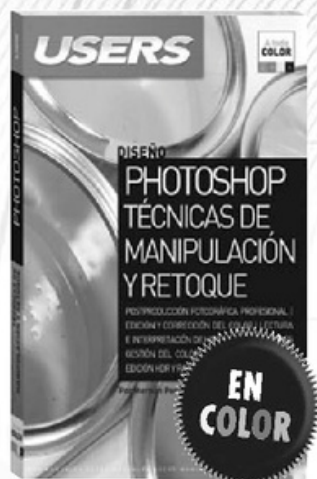
TAMBIÉN PUEDE CONSEGUIR NUESTROS LIBROS EN KIOSCOS O PUESTOS DE PERIÓDICOS, LIBRERÍAS, CADENAS COMERCIALES, SUPERMERCADOS Y CASAS DE COMPUTACIÓN.



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**



Photoshop

En este libro aprenderemos sobre las más novedosas técnicas de edición de imágenes en Photoshop. El autor nos presenta de manera clara y práctica todos los conceptos necesarios, desde la captura digital hasta las más avanzadas técnicas de retoque.

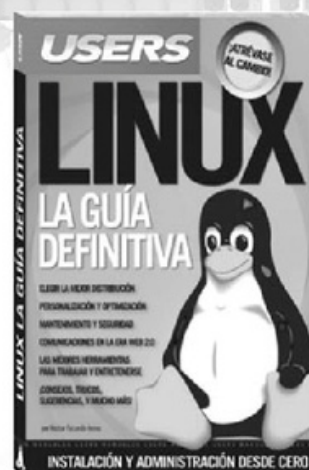
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-05-3



Grabación y producción de música

En este libro repasaremos todos los aspectos del complejo mundo de la producción musical. Desde las cuestiones para tener en cuenta al momento de la composición, hasta la mezcla y el masterizado, así como la distribución final del producto.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-04-6



Linux

Este libro es una completa guía para migrar e iniciarse en el fascinante mundo del software libre. En su interior, el lector conocerá las características de Linux, desde su instalación hasta las opciones de entretenimiento, con todas las ventajas de seguridad que ofrece el sistema.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-8-6



Premiere + After Effects

Esta obra nos presenta un recorrido detallado por las aplicaciones audiovisuales de Adobe: Premiere Pro, After Effects y Soundbooth. Todas las técnicas de los profesionales, desde la captura de video hasta la creación de efectos, explicadas de forma teórica y práctica.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-9-3



Office 2010

En este libro aprenderemos a utilizar todas las aplicaciones de la suite, en su versión 2010. Además, su autora nos mostrará las novedades más importantes, desde los minigráficos de Excel hasta Office Web Apps, todo presentado en un libro único.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-26013-6-2



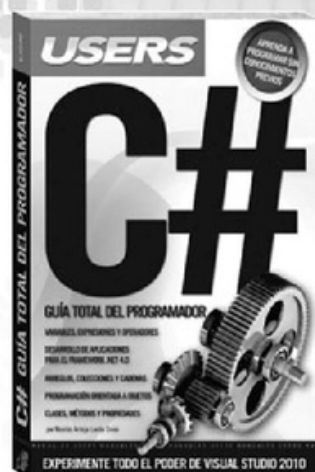
Excel Paso a Paso

En esta obra encontraremos una increíble selección de proyectos pensada para aprender, mediante la práctica, la forma de agilizar todas las tareas diarias. Todas las actividades son desarrolladas en procedimientos paso a paso de una manera didáctica y fácil de comprender.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-4-8

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



C#

Este libro es un completo curso de programación con C# actualizado a la versión 4.0. Ideal tanto para quienes desean migrar a este potente lenguaje, como para quienes quieran aprender a programar desde cero en Visual Studio 2010.

→ COLECCIÓN: MANUALES USERS
→ 400 páginas / ISBN 978-987-26013-5-5



200 Respuestas Seguridad

Esta obra es una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos contestar para conseguir un equipo seguro. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-1-7



Funciones en Excel

Este libro es una guía práctica de uso y aplicación de todas las funciones de la planilla de cálculo de Microsoft. Desde las funciones de siempre hasta las más complejas, todas presentadas a través de ejemplos prácticos y reales.

→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-26013-0-0



Proyectos con Windows 7

En esta obra aprenderemos cómo aprovechar al máximo todas las ventajas que ofrece la PC. Desde cómo participar en las redes sociales hasta las formas de montar una oficina virtual, todo presentado en 120 proyectos únicos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-036-8



PHP 6

Este libro es un completo curso de programación en PHP en su versión 6.0. Un lenguaje que se destaca tanto por su versatilidad como por el respaldo de una amplia comunidad de desarrolladores, que lo convierten en un punto de partida ideal para quienes comienzan a programar.

→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-663-039-9



200 Respuestas: Blogs

Esta obra es una completa guía que responde a las preguntas más frecuentes de la gente sobre la forma de publicación más poderosa de la Web 2.0. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-663-037-5

USERS**PRESENTA...**

¡EL PRIMER EBOOK USERS!

Sí, ya podés leer Hackers al descubierto en tu PC, notebook, Amazon Kindle, iPad, en el celular...

**CONSEGUILO
DESDE CUALQUIER
PARTE DEL MUNDO**

**A UN PRECIO
INCREÍBLE**

**¿QUÉ ESTÁS
ESPERANDO?**



**¡LEELO
DONDE
QUIERAS!**

INGRESA YA A [USERSHOP.REDUSERS.COM](http://usershop.redusers.com) Y ENTERATE MÁS

CLAVES Y TÉCNICAS PARA OPTIMIZAR SITIOS DE FORMA PROFESIONAL



Esta obra es una guía de referencia útil y versátil, tanto para programadores como para diseñadores, que deseen obtener resultados profesionales en sus desarrollos. En sus páginas, repararemos conceptos y técnicas que permitirán mejorar la experiencia final de los usuarios.

» DESARROLLO / INTERNET
» 400 PÁGINAS
» ISBN 978-987-1773-09-1



LLEGAMOS A TODO EL MUNDO VÍA  Y 

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

 usershop.redusers.com //  usershop@redusers.com

CONTENIDO

1 | PHP

Obtención / Licencia / Versiones / Extensiones en PHP /
Facilidad de aprendizaje / Portabilidad

2 | BASES DE DATOS RELACIONALES

¿Qué es y para qué sirve una base de datos? / Modelo
de datos / Operaciones relacionales básicas / Diseño /
Tipos de relaciones entre tablas / Normalización / MySQL /
Integridad referencial / Subconsultas

3 | MYSQL

Obtención / Diferencias entre versiones / Tipos de datos /
Tipos de tablas / Referencia de funciones / Procesos
almacenados / El uso del monitor

4 | PHP Y MYSQL

Integrar PHP con MySQL / Creación de bases de datos /
Selección de una base de datos / Creación de tablas /
Obtener listados / Múltiples bases de datos / Crear
una conexión

5 | SQL EN PHP

Consultas de selección / Selección simple / Consultas
multitabla / Subconsultas / Moverse entre registros /
Número de registros / Campos devueltos / Insert / Delete /
Update

6 | ERRORES

Reportes de error en PHP / Register globals / Códigos
en MySQL / Manejo de excepciones

7 | TRANSACCIONES

¿Qué es una transacción? / Casos de aplicación /
Implementación de transacciones con MySQL / La variable
autocommit / Tablas transaccionales

8 | SEGURIDAD EN MYSQL

Backup de bases de datos / Recuperación ante fallos /
Seguridad a nivel tabla / Introducción a la replicación /
Seguridad física

9 | ADMINISTRACIÓN DE MYSQL

El administrador de bases de datos / El rol del usuario /
Sistema de privilegios / Jerarquías de acceso / Baja
de usuarios

APÉNDICE A | CAPAS DE ABSTRACCIÓN

APÉNDICE B | EJEMPLO PRÁCTICO

APÉNDICE C | MYSQLI

NIVEL DE USUARIO

PRINCIPIANTE

INTERMEDIO

AVANZADO

EXPERTO

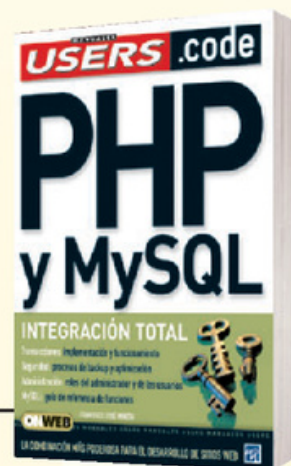
DESARROLLO PHP+MYSQL

Este libro presenta la fusión de dos de las herramientas más populares para el desarrollo de aplicaciones web de la actualidad: la base de datos relacional MySQL, robusta y confiable, integrada con PHP, un lenguaje dinámico que se encuentra en constante evolución. En sus páginas, el autor nos enseñará la sintaxis básica de las funciones, de modo que el programador adquiera rápidamente los conocimientos y, luego, los aplique en sus propios desarrollos. A partir de esa base, tendremos un acercamiento progresivo al lenguaje y a su capacidad particularmente sencilla de resolver problemas.

Todos los procedimientos son expuestos de manera práctica, con diagramas conceptuales, código fuente de ejemplo y la teoría necesaria para comprender en profundidad cada tema presentado. Al finalizar la obra, gracias a la integración aprendida de lenguaje y base de datos, podremos lograr sitios complejos y optimizados en nuestra práctica profesional.

Francisco Minera, reconocido desarrollador y autor de varios libros sobre el tema para esta editorial, nos muestra en estas páginas el resultado de años de experiencia en el trabajo con pequeñas y medianas empresas.

Versión completamente mejorada, actualizada
y ampliada de la obra "PHP y MySQL"



RedUSERS.com

En este sitio encontrará una gran variedad de recursos y software relacionado, que le servirán como complemento al contenido del libro. Además, tendrá la posibilidad de estar en contacto con los editores, y de participar del foro de lectores, en donde podrá intercambiar opiniones y experiencias.

Si desea más información sobre el libro puede comunicarse con nuestro Servicio de Atención al Lector: usershop@redusers.com

PHP + MYSQL DEVELOPMENT

This manual will help the reader strengthen his skills when handling the practical uses that can be achieved by the combination of two of the most powerful development tools: PHP and MySQL.

