

## Teoría: Introducción y uso de MySQL



### 1 - Objetivos

#### Objetivos

- **Conocer la arquitectura** de una aplicación web que utiliza bases de datos.
- **Asimilar el concepto y estructura** de una base de datos.
- **Adquirir conocimientos básicos de la base de datos MySQL** y utilizar correctamente sus sentencias para realizar diferentes operaciones con los contenidos de las bases de datos.
- **Conocer y saber aplicar** correctamente las principales sentencias SQL.

### 2 - Arquitectura de una aplicación web que use bases de datos

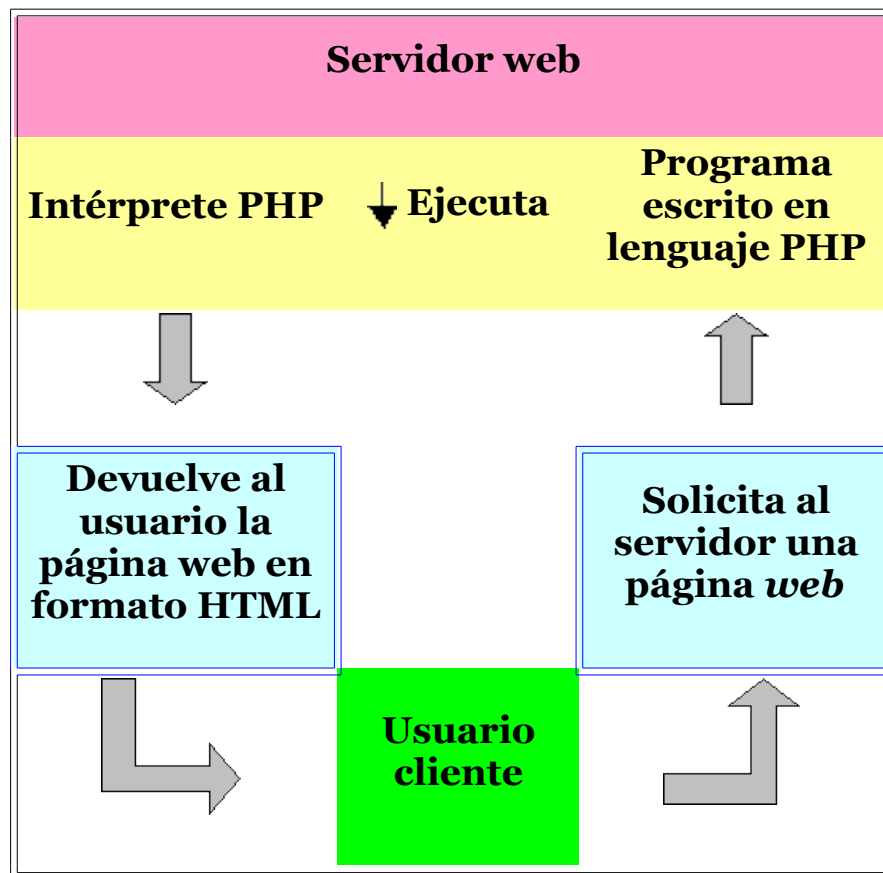
Hemos visto hasta ahora cómo generar páginas dinámicas con PHP. En los ejercicios y ejemplos anteriores hemos podido apreciar la capacidad de este lenguaje para crear este tipo de páginas, si bien la información mostrada hasta el momento es bastante sencilla y elemental. Pero PHP adquiere verdadera potencia y utilidad en la generación de páginas dinámicas cuando utiliza la información archivada en bases de datos.

Éste es precisamente el terreno en el que los lenguajes generadores de páginas *web* alcanzan resultados espectaculares, hasta el punto de que más del 90% de los *websites* se sirven de este procedimiento para enviar al cliente su información. Por ejemplo, cualquier cadena de noticias tiene una base de datos que puede ser actualizada permanentemente, incluso a larga distancia por los corresponsales remotos, cuya información ofrece en tiempo real a los clientes que entran en su *web*. Lo mismo pasa con las empresas de comercio electrónico (*e-commerce*): usan diferentes bases de datos para contener información sobre sus productos, los perfiles de sus clientes reales o potenciales y los datos de las compras que éstos vayan haciendo. También las *web* de las entidades bancarias usan bases de datos para archivar y mostrar información sobre los movimientos en las cuentas corrientes y otras operaciones de sus clientes.

En esta Unidad vamos a abordar conocimientos básicos sobre las bases de datos y cómo se integran en una página *web*, estudiando concretamente la base de datos **MySQL** y el lenguaje de consulta **SQL**. En la siguiente Unidad abordaremos, directamente ya, cómo trata PHP las bases de datos al generar las páginas dinámicas.

## Arquitectura de una aplicación web

En el apartado **Funcionamiento de un programa codificado con PHP** de la **Unidad 1** presentamos el siguiente esquema:



En este gráfico aparecen sólo dos de las tres **capas (tiers)** que integran el proceso completo de interpretación de una página web.

En la **primera capa o nivel** el **usuario**, utilizando su navegador, hace una **solicitud al servidor**, con el que se conecta a través del protocolo HTTP. Le pide que interprete un *script* y le remita el resultado, que se recibe en formato de página HTML.

En la **segunda capa o nivel** el **servidor envía esa solicitud al intérprete PHP** para que ejecute el programa y devuelva el resultado al navegador del cliente.

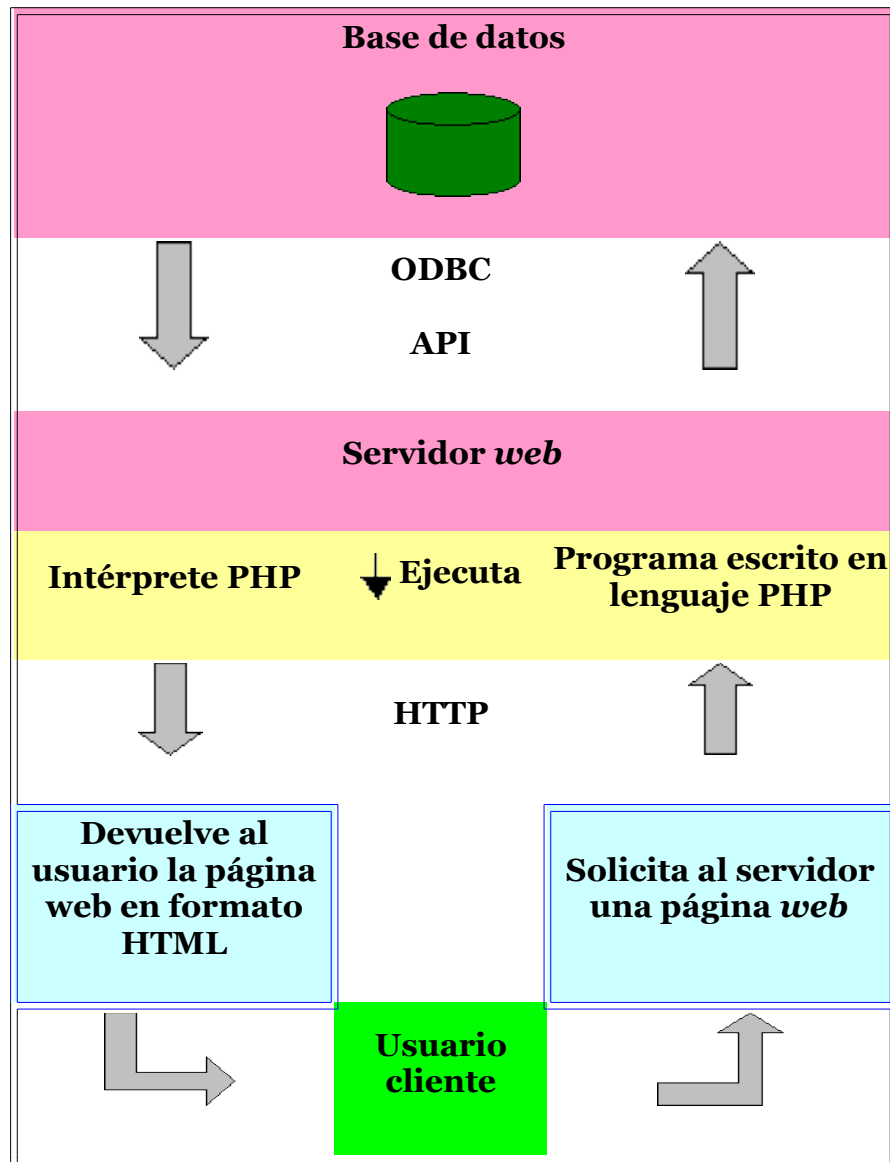
Pero ahora, llegado el momento, debemos añadir una **tercera capa o nivel**, que **está integrada por la base o bases de datos con las que conecta el servidor y de las que extrae la información solicitada por el cliente**. Es decir, el servidor, cuando encuentra en el programa órdenes y funciones de creación, modificación y consulta de una base de datos, acceda a ella y extrae, elimina o actualiza esa información, según las operaciones que deban ejecutarse en el *script*.

Para establecer esta conexión entre el servidor y la base de datos, es preciso utilizar enlaces que sirvan de canal de comunicación entre ambos. Estos enlaces pueden ser de dos tipos: **el tipo estándar ODBC** (*Open DataBase Connectivity*), que es más genérico y sirve para conectar cualquier servidor con aquellas bases que lo soporten, que son casi todas; y **el enlace particular o nativo** que tenga cada tipo de base de datos, denominado **API** (*Application*

*Programming Interface*), que está integrado por el conjunto de funciones propias de un tipo de base de datos que puede incorporar el programador cuando está utilizando ese tipo de base.

El primer tipo de enlace es más general y, por tanto, vale para todos los tipos de base que lo soporten, que son la gran mayoría, pero a su vez su ejecución es más lenta que la de las APIs de cada base de datos particular. El segundo tipo de enlace es más rápido y eficaz cuando se trabaja sólo con un tipo de base de datos, pero exige cambiar el código si se utiliza otro tipo de base de datos. Por ejemplo, Oracle y MySQL tienen APIs completamente diferentes, mientras que estos dos tipos de base de datos soportan perfectamente también el enlace ODBC.

### Así queda el esquema completo



Así pues, en las **Unidades 6 y 7** vamos a aplicar el proceso completo:

- Se escribe el código fuente de la página web en lenguaje PHP incluyendo las funciones API nativas de la base de datos, en nuestro caso MySQL.
- Se guarda este código en el servidor web.

- El navegador del usuario solicita al servidor la página codificada y guardada en los dos pasos anteriores.
- El servidor interpreta el código PHP conectando, cuando sea preciso, con la base de datos y realizando con ella las operaciones previstas.
- El servidor envía el resultado del código PHP al navegador del usuario, que ve en su pantalla la página en formato HTML.

### 3 - Las bases de datos

En este apartado vamos a explicar brevemente algunos conceptos fundamentales sobre las bases de datos. Suponemos que el alumno o alumna ya tiene conocimientos suficientes sobre las mismas y, por tanto, abordaremos sólo algunos conocimientos más relacionados con la forma en que PHP accede a las bases de datos y trata su información.

El término **base de datos** es informático, pero puede aplicarse también a la forma como se almacena, ordena y utiliza manualmente la información. Por ejemplo, la agenda en la que apuntamos manualmente los datos con el nombre, dirección y teléfono de nuestros amigos o personas con las que necesitamos relacionarnos, puede ser considerada una base de datos en este sentido más amplio. Igualmente, el archivador en el que guardamos las fichas bibliográficas de una investigación o de una biblioteca también puede ser considerado como tal. En todas ellas se almacena información, se ordena de alguna forma (alfabéticamente en la agenda, por temas, autores o títulos en el fichero de la biblioteca, etcétera) y se consulta cuando es necesario.

Sin embargo, en el sentido informático, la palabra **base de datos** se refiere a una colección, conjunto o depósito de datos, almacenados en un soporte magnético o de otro tipo, accesibles por múltiples usuarios de forma rápida y eficaz mediante el ordenador a través de una o de varias aplicaciones informáticas independientes de los datos. Éstos se relacionan entre sí y están organizados de tal manera que es fácil introducirlos, actualizarlos, recuperarlos o llevar a cabo con ellos otras operaciones de gestión.

En el caso de PHP es el servidor quien conecta con la base de datos, a petición del navegador del usuario, y realiza las operaciones previstas en la aplicación, devolviendo posteriormente al navegador del cliente los resultados de esas operaciones.

Generalmente, en las **bases de datos** relacionales, de las que hablaremos después, la información está almacenada y organizada en ficheros formados por filas y columnas, como puede verse en el ejemplo siguiente, en el que se presentan algunos datos de cinco libros de una biblioteca:

#### Columnas

	Título	Autor	Editorial
	El invierno en Lisboa	Antonio Muñoz Molina	Seix Barral
	¿Tener o ser?	Erich Fromm	Fondo de Cultura Económica
<b>Filas</b>	Crónica de una muerte anunciada	Gabriel García Márquez	Bruguera
	El lobo estepario	Hermann Hesse	Anaya Editores
	La vida está en otra parte	Milan Kundera	Seix Barral

Cada fila contiene el título, el autor y la editorial de un libro y se relaciona con las demás filas gracias a que incluye el mismo tipo de información (datos de los libros) y en todas ellas la información está organizada de la misma forma: la primera columna contiene el título del libro, la segunda, el autor y la tercera, la editorial.

Así pues, una **base de datos** contiene un conjunto de ficheros cuya información está organizada de tal forma que puede ser tratada informáticamente con rapidez y eficacia. La información de una base de datos puede almacenarse en un solo fichero o en varios.

Los ficheros de una **base de datos** están grabados en el servidor. Tienen un nombre (por ejemplo, flores, rios, libros, coches, amigos, artículo, clientes, ventas, facturas, etcétera). Su denominación debe seguir las normas establecidas para que el nombre de un fichero sea correcto. Como puede verse, hemos prescindido de las tildes en los identificadores que las llevan ortográficamente.

El tipo o extensión de estos ficheros de base de datos puede ser muy variado, según el tipo de base de datos utilizado: en dBase es **dbf** (*Data Base File*, Fichero de Base de Datos), en Access **mdb**, en Interbase **db** o **dbf**, en MySQL **myd**, etcétera.

Las **filas** de un archivo de base de datos se denominan **registros** y las **columnas**, **campos** (**fields**, en inglés).

Así pues, un fichero de base de datos **está integrado por registros**, que son cada uno de sus elementos o componentes (flor, río, libro, coche, amigo, artículo, cliente, venta o factura). Todos los registros contienen un **conjunto de campos** en los que se almacena su información; este conjunto define la estructura del fichero que integra una base de datos.

En la representación gráfica siguiente puede observarse, en forma de tabla, la estructura de un

fichero que contiene una base de datos con información sobre personas:

Campos

↓
↓
↓
↓
↓

	Nombre	Sueldo	Fecha_nac	Observacio	Foto
1					
2					
3					
4					
<u>Registros</u> 5					
6					
7					
8					
9					
10					
11					

En las filas aparecen hasta once **registros** cada uno de los cuales, en este caso, contiene los cinco **campos** siguientes: Nombre, Sueldo, Fecha\_nac, Observacio y Foto.

En el ejemplo anterior sólo se han incluido once **registros** y cinco **campos**, pero de hecho en las bases de datos que vamos a usar el número de **registros** es ilimitado (depende de la capacidad del soporte) y el de **campos** es muy amplio, según el tipo de base de datos usada. Todos los **registros** tienen los mismos **campos**.

Si comparamos un fichero de base de datos con los archivadores de una biblioteca, podemos decir que éstos integran la base de datos. Cada archivador es como un fichero de la **base de datos**, las fichas que hay en su interior son los **registros** y los apartados de cada ficha (título, autor, editorial, etcétera) son los **campos**.

Cada **campo** de un **registro** tiene **un nombre**, **un tipo**, **una longitud** o ancho, un **número de decimales** si es de tipo numérico o de coma flotante y **un índice** opcional. Según el tipo de base de datos que se esté utilizando, el identificador del campo, la clase de tipos y la longitud de los mismos pueden ser diferentes. **Nos vamos a referir, como ejemplo, sólo a la estructura de las bases de datos creadas con MySQL.**

El nombre de cada campo puede ser muy largo, si bien recomendamos que en el orden práctico sea lo más breve posible y tenga algún significado. Debe atenderse a las reglas de todos los identificadores ya comentadas anteriormente.

Hay estos tipos de **campos**, que citamos de forma genérica:

1. **Campo de tipo Carácter** . Es el más común (letras, dígitos, signos, etcétera), y contiene información que es tratada como una cadena de caracteres. Se asigna este tipo a un **campo** cuando no se realizan operaciones aritméticas con sus datos, ni contiene una fecha, ni es un texto mayor de 255 caracteres. Por ejemplo, se asigna este tipo al campo cuyo contenido va a ser el nombre de una persona, sus apellidos, domicilio, localidad, provincia, etcétera. Admite índice. En MySQL hay dos tipos de campos para almacenar datos de esta clase: CHAR y VARCHAR. Tiene una longitud desde 1 hasta 255 caracteres. Si la longitud de la cadena que se asigna a este campo no alcanza los 255 caracteres, el tipo CHAR completa con espacios en

blanco los que falten ocupando siempre este campo la misma longitud. En cambio, el tipo de carácter VARCHAR sólo ocupa el espacio que precise la cadena de texto, es decir, su longitud es variable y, por tanto, se ahorra espacio cuando el contenido puede ser muy diferente en longitud.

2. **Campo de tipo Numérico.** Se utiliza para escribir números, incluidos los signos positivo y negativo. Se asigna este tipo a un campo cuando se realizan operaciones aritméticas con números enteros o reales, como sumar, restar, multiplicar, dividir, etcétera. Admite índice. MySQL admite estos valores para determinar los campos de este tipo: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, REAL, DOUBLE, FLOAT, DECIMAL y NUMERIC. Como puede verse, en realidad los valores posibles se refieren a si es un campo de número entero o decimal y al número de dígitos con los que debe representarse el valor en cada caso.

3. **Campo de tipo Fecha.** Puede contener fechas y tiempos (horas, minutos, segundos). Admite índice. MySQL admite aquí los valores siguientes: DATE, TIME, TIMESTAMP y DATETIME.

4. **Campo de tipo Memo.** Es un campo de longitud variable que admite gran cantidad de texto según nuestras necesidades. Para cada registro tendrá una longitud distinta, según la cantidad de texto que se introduzca en este campo. No admite índice. MySQL admite aquí los valores siguientes: TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, TINYTEXT, TEXT, MEDIUMTEXT y LONGTEXT.

5. **Campo de tipo serie.** Es un campo donde se puede poner uno de los valores incluidos en una lista o serie de valores. MySQL admite aquí los valores siguientes: ENUM y SET.

**Cuando utilicemos la base de datos MySQL, veremos concretamente cómo se crean las bases de datos, sus tablas y los campos de cada una.**

## Ventajas de las bases de datos

Hemos dicho que los archivadores de una biblioteca o de una agenda pueden considerarse, en cierta forma, **bases de datos**, pues en ellos se almacena información en un determinado orden y es posible buscar esta información, consultarla, modificarla o eliminarla con facilidad.

Sin embargo, todas estas operaciones suelen llevar mucho tiempo y, en ocasiones, no se efectúan tan fácilmente como desearíamos. Además, ocupan bastante espacio si la información es abundante. Incluso, en ocasiones, algunas operaciones fundamentales son imposibles de realizar manualmente.

Por ejemplo, si tenemos 1.000 fichas bibliográficas ordenadas por autor y necesitamos ordenarlas por título, la operación ha de realizarse manualmente, mirando una a una cada ficha, lo cual puede hacerse muy largo y pesado. Podíamos haber escrito dos ejemplares de cada ficha, uno para el archivo por autores y otro para el de títulos, pero esto hubiera llevado el doble de tiempo, de trabajo y éstas ocuparían el doble de espacio.

Supongamos ahora que necesitamos seleccionar todas las fichas en las que aparece la misma editorial. De nuevo la tarea puede parecernos pesada y larga, y lo es. No digamos si se cambia la situación de los libros en los armarios de la biblioteca. También será necesario modificar la signatura en las fichas.

Hemos puesto este ejemplo para explicar los graves problemas que se derivan de la gestión manual de la información. Las dificultades aumentan a medida que crece el volumen de información que debe manejarse y según sean los criterios de ordenación y selección.

En una **base de datos informática**, en cambio, al gestionarse la información automáticamente, **muchos de los problemas anteriormente mencionados desaparecen**.

En primer lugar, **la rapidez de las operaciones fundamentales** (introducción de datos, ordenación por diferentes campos, consultas, búsquedas, elaboración de informes, actualización y modificación de los datos, etcétera) **aumenta de una forma muy destacada**.

En segundo lugar, **el espacio que ocupa una base de datos es mucho menor** que el de cualquier otra forma de archivo manual. En un disco flexible de 3,5 pulgadas puede almacenarse casi un millón y medio de caracteres. En los discos duros de los actuales servidores el volumen de información puede ser prácticamente ilimitado.

En tercer lugar, **las operaciones fundamentales de gestión de la información son automáticas**, lo cual hace que sean menos pesadas y tediosas si son llevadas a cabo por el ordenador. Así pues, el trabajo se humaniza y el tiempo libre de las personas que manejan la información es mayor.

Finalmente, **la seguridad de los datos informatizados también es mayor** que la contenida en archivos de tipo manual, pues el ordenador nos permite hacer rápidamente cuantas copias queramos de esa información en diferentes soportes.

Desde la aparición de los ordenadores, éstos se han dedicado al almacenamiento y organización de grandes volúmenes de datos. Igualmente, se han aplicado a la evaluación de las diversas soluciones propuestas para resolver los problemas de estructuración y acceso a dicha información.

## Bases de datos relacionales

Se ha descubierto que la mejor forma de resolver estos problemas es organizar la información de forma relacional. De aquí ha surgido el concepto de **bases de datos relacionales** (**RDBMS**, **Relation DataBase Management System**).

El fundamento teórico de las **bases de datos relacionales** es complejo, ya que se basa en el concepto matemático de **relación** entre los elementos de un conjunto. Sus características y propiedades formales requieren ciertos conocimientos de la teoría de conjuntos. Sin embargo, en la práctica, **el concepto de relación es muy sencillo de utilizar porque en ésta la organización de los datos es muy clara e intuitiva**.

En otros tipos de organización de la información, como las **bases de datos jerárquicas** o las **bases de datos en red**, anteriores a las relacionales, aparecían distintas categorías de datos y estructuras muy complejas y poco flexibles que dificultaban la posibilidad de relacionar éstos con eficacia y rapidez. En cambio, en las **bases de datos relacionales** **la información se organiza en ficheros que tienen estructura tabular** o en forma de tabla, en la que todos los datos tienen la misma categoría. Cada tabla también recibe el nombre de **relación**.



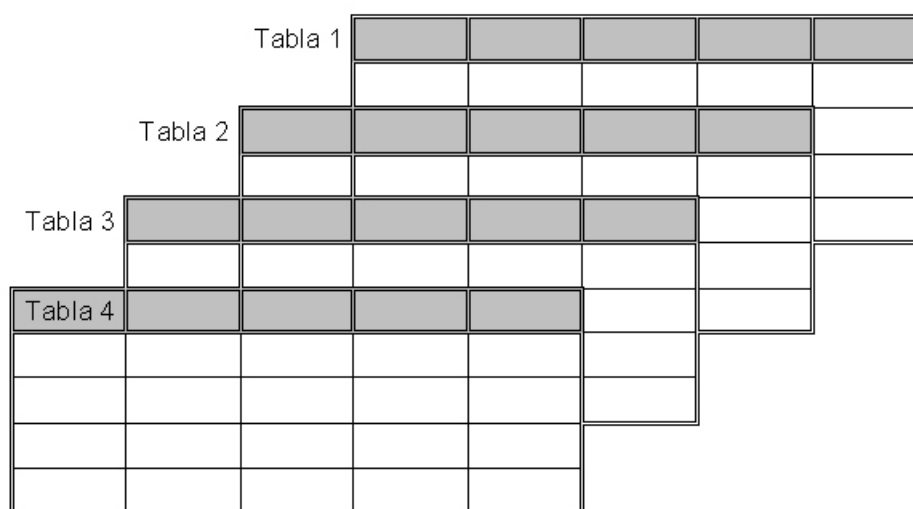
Por ejemplo, en el gráfico siguiente puede observarse una tabla de tipo **dBase** que contiene diversos datos de personas:

<u>Cabecera</u>	Nombre	Dirección	Edad	Sexo	Profesión
1	León García	C/ Zurita, 25	25	V	Admtvo.
2	Maria Pérez	C/ Flores, 15	30	M	Abogada
Filas (Registros) 3	José Rodríguez	C/ Río Sil, 11	50	V	Dependiente
4	Juana de Dios	Avda. Canarias, 50	70	M	Jubilada
5	Begoña López	Pza. Segovia, s/n	15	M	Estudiante
Columnas (Campos)					

Como se ve, una tabla consta de filas y de columnas; en cada columna, denominada **campo** en la **base de datos**, hay un dato: Nombre, Dirección, Edad, etcétera; cada fila es un **registro** que contiene todos los datos de los elementos de la base. Cada tabla tiene un registro especial, denominado **cabecera**, que contiene los nombres de los campos y sus atributos (tipo y longitud). Cada tabla se almacena en un fichero de tipo **dbf**, es decir, un fichero de **base de datos**.

Generalmente, una **base de datos** no consta de una sola tabla, sino de varias.

Gráficamente puede representarse así:



Estas tablas no son independientes unas de otras, sino que tienen al menos un campo común con las otras a través del cual se puede acceder a la información que contienen todas en conjunto.

Por ejemplo, la base de datos de una biblioteca puede estar integrada por una tabla de libros, otra de lectores, otra de préstamos y otra de editoriales. El fichero de libros puede contener la información completa de cada volumen: título, autor, editorial, año de edición, precio, número de páginas, código de materia, número de registro, etcétera.

El fichero de editoriales contendrá los datos de cada entidad editora: nombre, dirección, teléfono, plazo de entrega, descuentos, etcétera.

El fichero de lectores estará integrado por los datos personales y profesionales de éstos: nombre, DNI, dirección, teléfono, profesión, centro de trabajo, número de carné, etcétera.

El fichero de préstamos contendrá datos de este tipo: número de registro del libro prestado, número de carné del lector, fecha del préstamo, plazo, etcétera.

Como puede verse, **la información no debe repetirse** en todos los ficheros, **pero sí debe poder relacionarse**. Por ejemplo, los ficheros de libros y editoriales, tienen en común el campo EDITORIAL. Los ficheros de libros y préstamos tienen en común, al menos, el NÚMERO DE REGISTRO del libro prestado, gracias a lo cual desde uno se puede acceder a los datos del otro. Los ficheros de lectores y préstamos tienen en común el campo CARNÉ, etcétera.

**Son bases de datos relacionales** Microsoft Access, Oracle, SQL Server, MySQL y otras. En este curso vamos a utilizar, sobre todo, bases de datos de este tipo.

Hay otro tipo de base de datos que está orientado a los objetos (**ODBMS**, **Object Oriented DBMS**) en las que cada dato es tratado como si fuera un objeto con sus atributos y métodos. Son de este tipo ObjectStore, Versand, GemStore, etcétera.

También hay otro tipo que reúne características propias de los dos tipos anteriores, como PostgreSQL, que son conocidas como **ERDBMS** (**Extended Relacional DBMS**) y como **ORDBMS** (**Object Relational DBMS**). **PHP puede tratar prácticamente todas las bases de datos mencionadas.**

## Diseño de bases de datos

El diseño de bases de datos puede presentar distinto tipo de dificultad dependiendo de la complejidad e interrelación de los datos que se quiera gestionar.

Imaginemos que una compañía aérea quiere gestionar toda la información contenida en una base de datos relativa a los aviones y su mantenimiento, a los vuelos, viajes, destinos, clientes, personal de la empresa, agencias de viajes, billetes, asistencia, etcétera. Es evidente que, en este caso, la complejidad es enorme y que para realizar el diseño de esta base se requiere la colaboración de técnicos especialistas que faciliten la tarea.

Sin embargo, en la mayoría de las ocasiones el diseño de una base de datos se resuelve con uno, dos o tres ficheros como máximo. En este caso no es necesario profundizar en aspectos complejos de técnicas de diseño, sino que basta aplicar el sentido común para organizar los ficheros de la base de datos de forma coherente.

**Deben crearse tantos ficheros como categorías o grupos de elementos distintos haya que organizar.** Por ejemplo, en una tienda que vende al por menor bastaría con crear un fichero de artículos y otro de proveedores, y a lo sumo otros tres: de pedidos, de ventas y de clientes.

**Antes de ponerse a crear una base de datos con el ordenador, es preciso diseñarla previamente sobre el papel.** La planificación es fundamental en este caso para evitar errores graves: falta de datos necesarios, repetición innecesaria de algunos, equivocación del tipo de campo o falta de precisión en su longitud. Aunque es posible modificar la estructura de una base de datos, una vez creada, se puede perder mucho tiempo e incluso datos en esta operación.

Diseñar una base de datos consiste en determinar los datos que van a introducirse en ella, la forma como se van a organizar y el tipo de esos datos. Además, se debe precisar la forma como se van a solicitar y las clases de operaciones que hay que realizar con los mismos: aritméticas, lógicas, de fechas, de carácter, etcétera. También conviene conocer los resultados

concretos que se espera obtener: consultas, informes, actualizaciones, documentos, etcétera.

A continuación, se resumen las operaciones que deben llevarse a cabo al diseñar una base de datos:

### **1. Atendiendo a la información que contiene es preciso:**

- Identificar los diferentes elementos informativos (artículos, clientes, ventas, facturas, etcétera) que forman parte de la base de datos.
- Determinar los datos que debe contener cada uno de esos elementos.
- Precisar el grado de necesidad y de utilización de cada dato.
- Concretar las operaciones que se van a realizar con los datos: aritméticas, lógicas, de salida sólo por la pantalla, de salida también por la impresora, etcétera.
- Seleccionar el dato o datos esenciales que deben ser el campo clave por el que se ordenarán las unidades o elementos mencionados.
- Fijar los datos comunes a los diferentes ficheros de la base de datos que van a permitir relacionar la información distribuida entre ellos.

### **2. Atendiendo a la estructura de la base de datos**

- Distribuir la información en ficheros según los diferentes grupos que se hayan hecho (artículos, clientes, etcétera) y dar un nombre a cada fichero.
- Determinar el nombre de cada campo de los registros de cada fichero. Este nombre ha de ser claro y debe significar algo para que pueda recordarse fácilmente.
- Decidir qué tipo conviene asignar a cada campo según la clase de operaciones que vayamos a realizar con sus datos.
- Asignar a cada campo una longitud apropiada para tener los datos fundamentales sin despilfarro de memoria interna ni de espacio en el disco duro o soporte empleado.
- Establecer un orden lógico y práctico agrupando los campos según un criterio concreto: clase e importancia de los datos, frecuencia de utilización, proximidad, parecido, etcétera.
- Decidir cuál o cuáles van a ser los campos clave permanentes y situarlos al principio de la estructura.
- No incluir campos que puedan ser el resultado de diversas

```
operaciones de tratamiento posterior.
```

```
• Fijar los campos comunes a todos los ficheros para poder relacionarlos con otros de la misma aplicación.
```

## 4 - La base de datos MySQL

La base de datos MySQL ha formado y forma desde el principio una pareja perfecta con PHP. En este apartado vamos a trabajar con ella, ya que es la que utilizamos en el curso.

La mayoría de los desarrolladores, desde que PHP puede utilizar bases de datos, han seleccionado esta base de datos. Por ejemplo, hace ya algunos años que la NASA migró algunas de sus aplicaciones web a PHP utilizando como base de datos precisamente MySQL.

Advertimos que una de las operaciones más pesadas y engorrosas es instalar el servidor MySQL, pero al instalarse el curso se da la instalación ya hecha de forma automática, con los usuarios necesarios para el curso ya creados y los permisos de acceso y privilegios asignados convenientemente, así como la conexión ya hecha. Consideramos que éstas son operaciones propias de los administradores del servidor, para quienes no va dirigido este curso. Así pues, el alumno o alumna podrá trabajar perfectamente sin tener que instalar el servidor MySQL ni poseer conocimientos sobre la configuración adecuada del mismo. Si el alumno o alumna dispone ya de un servidor MySQL instalado en el ordenador recomendamos desinstalarlo para que el servidor MySQL del curso arranque sin problemas.

Con el fin de asimilar bien los contenidos sobre las bases de datos, en esta Unidad sólo se van a explicar de forma sencilla y práctica dos asuntos importantes:

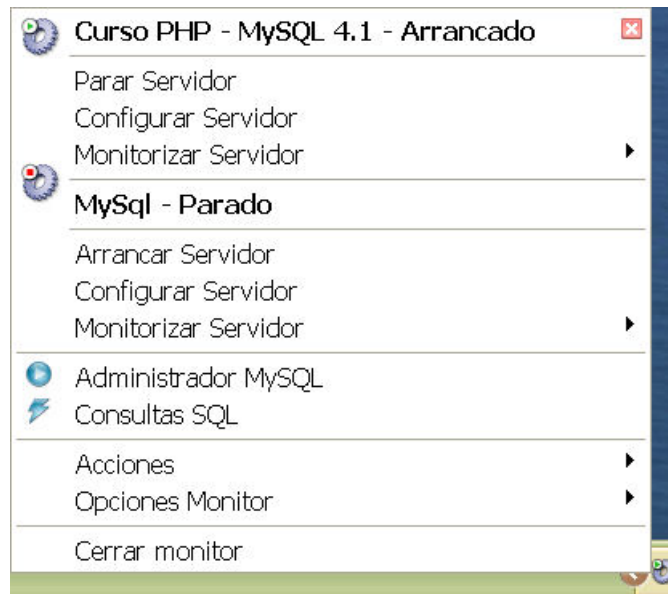
1. La base de datos MySQL, a través del Cliente MySQL, que facilita y hace más intuitiva la comprensión de los conceptos. Así se evita, a la vez, la poco estimulante tarea de escribir una a una las instrucciones en el *prompt* del sistema. El estudio y las prácticas realizadas con esta utilidad prepararán a los alumnos y alumnas para poder aplicar estos conceptos a otros tipos de bases de datos.
2. Las funciones de PHP que permiten tratar las bases de datos y su información en MySQL. Este asunto se abordará en el apartado siguiente.

Con los conocimientos anteriores bien asimilados, será más fácil comprender con detalle y aplicar correctamente las funciones de PHP que permiten conectarse a una base de datos, realizar consultas, acceder a los resultados de las consultas y realizar otras operaciones auxiliares e informativas sobre cualquiera de las bases de datos que usamos en el curso. También será posible utilizar el lenguaje de consulta SQL para obtener informes. Esto es precisamente lo que abordaremos en la Unidad siguiente.

Para tener acceso a las bases de datos, en primer lugar, hay que arrancar el servidor de datos MySQL.

Si el curso se instala en Windows 98/Me, en la carpeta Curso de PHP 5 del escritorio aparece el icono Servidor MySQL 4.1 - Arrancar. Con doble *click* lo arrancamos. Cuando queramos parar el servidor basta con hacer doble clic en el icono Servidor MySQL 4.1 - Parar.

Si el curso se instala en **Windows NT/2000/XP**, en la carpeta **Curso de PHP 5** del escritorio aparece el icono **Servidor MySQL 4.1**. Con doble *click* lo arrancamos. En la parte derecha de la barra de herramientas del escritorio aparece un nuevo icono. Si hacemos clic con el botón derecho sobre el mismo, aparecerá un *Menú emergente* donde deberemos pulsar el botón "Arrancar Servidor" debajo del icono "**Curso PHP - MySQL 4.1 - Parado**". A continuación se cambiará el mensaje en esta ventana indicando que el servidor MySQL está arrancado. Además, en este Menú se ofrecen varias opciones para gestionar el servidor, que nos permiten realizar, entre otras, operaciones como **cerrarlo** (**Parar Servidor**), **configurar el servidor** (**Configurar Servidor**) y **arrancar el servidor** de nuevo (**Arrancar Servidor**), etcétera. A continuación, se muestra este *Menú*:



Ahora ya podemos arrancar el fichero **Mysql.exe**, que aparece en el directorio **C:\cursoPHP5\bin\MySQL4.1\bin**. Este programa se ejecutará en modo MSDOS y se presentará en la pantalla la ventana principal de este gestor de bases de datos. Para arrancarlo debemos escribir **mysql.exe -u root** (con "**-u root**" estamos indicando el usuario con el que queremos conectarnos al servidor) desde una ventana de comandos en el directorio **C:\cursoPHP5\bin\MySQL4.1\bin**. Éste es su texto inicial:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.1.12a
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

Ya conocemos este tipo de programas MSDOS. Podemos traducirlo como

```
Bienvenido al monitor de MySQL. Acabe los comandos con ; o con \g.
Su identificador de conexión MySQL es 4 en el servidor versión: 4.1.12a.
```

```
Escriba 'help;' o '\h' para pedir ayuda. Escriba '\c' para vaciar el buffer.
```

```
mysql>
```

El cursor se coloca detrás del *prompt* **mysql>** esperando que vayamos introduciendo las órdenes o comandos que deban ejecutarse. Por ejemplo, si escribimos

```
mysql> show databases;
```

se muestra a continuación:

```
+-----+
| Database |
+-----+
| cursophp5 |
| mysql     |
+-----+
2 rows in set (0.00 sec)
```

Si escribimos

```
mysql> \h
```

aparece la información de ayuda

MySQL commands:

Note that all text commands must be first on line  
and end with ';'.

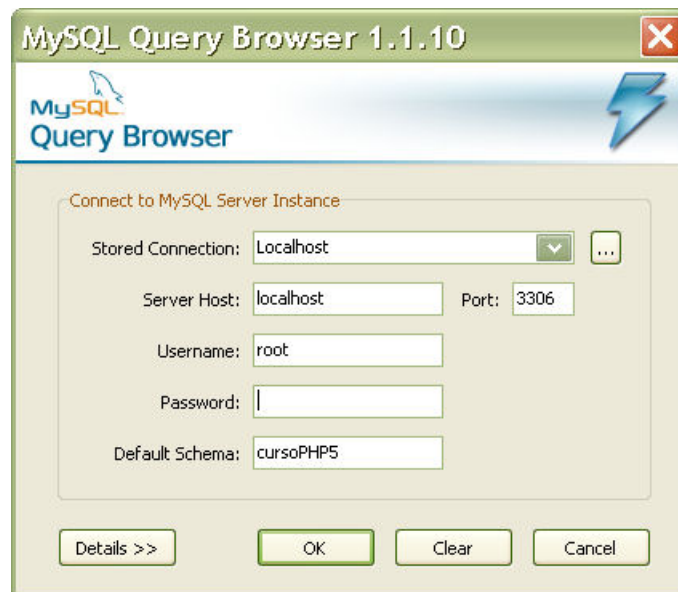
```
help      (\h)      Display this help.
?         (\?)      Synonym for `help'.
clear     (\c)      Clear command.
connect   (\r)      Reconnect to the server. Optional arguments are db and host.
ego       (\G)      Send command to mysql server, display result vertically.
exit      (\q)      Exit mysql. Same as quit.
go        (\g)      Send command to mysql server.
notee     (\t)      Don't write into outfile.
print     (\p)      Print current command.
quit      (\q)      Quit mysql.
rehash    (\#)      Rebuild completion hash.
source    (\.)      Execute a SQL script file. Takes a file name as an argument.
status    (\s)      Get status information from the server.
tee       (\T)      Set outfile [to_outfile]. Append everything into given outfile.
use       (\u)      Use another database. Takes database name as argument.
```

El uso de esta herramienta supone que debemos conocer las órdenes de MySQL y su sintaxis correcta, procedimiento ya algo anticuado y de otra época . Por eso, hemos preferido servirnos de una utilidad más moderna bajo Windows, con la que podemos realizar en MySQL todas las operaciones con las bases de datos, a la vez que aprender de una forma más amigable los comandos de MySQL que permiten realizar todas las operaciones propias de las bases de datos. Se denomina **MySQL Query Browser** (si has instalado el curso en Windows NT/2000/XP) o **MySQL-Front** (si has instalado el curso en Windows 98/Me). Ambas pueden arrancarse desde la carpeta **Curso de PHP 5** pulsado sobre el acceso directo **Ciente MySQL**.

Abandonamos, pues, con el comando `quit;` la utilidad **MySQL monitor** y arrancamos esta moderna aplicación con la que vamos a aprender a trabajar con las bases de datos de tipo MySQL.

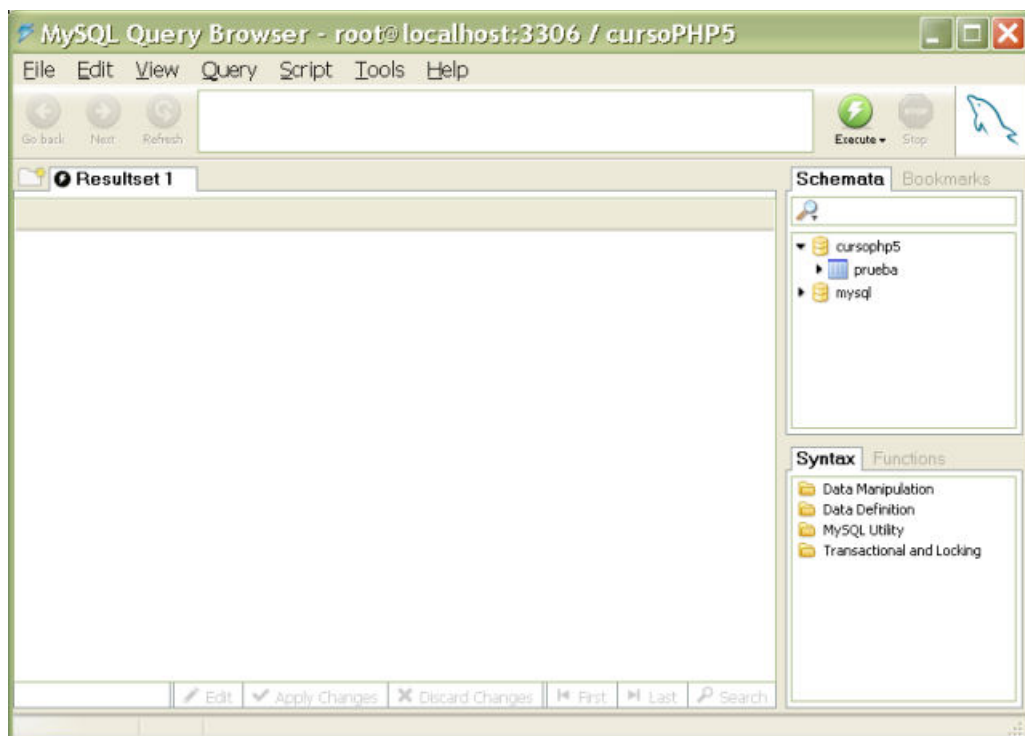
## MySQL Query Browser

Si has instalado el curso en Windows NT/2000/XP se instalará este cliente MySQL. Al arrancar el programa **Cliente MySQL**, se muestra en la pantalla la ventana siguiente de conexión:



Aceptamos los datos que están puestos por defecto: descripción de la conexión ([localhost](#)), nombre del servidor ([localhost](#)), usuario ([root](#)) y la contraseña (*password*) en blanco. El usuario [root](#) (*raíz*) es el que tiene todos los permisos y privilegios de acceso, sin restricciones, a las bases de datos, como si fuera el administrador. Así pues, no hay que cambiar nada, sino sólo pulsar sobre el botón **OK** y seguir.

A continuación, entramos ya en la pantalla de esta aplicación, que es la siguiente:



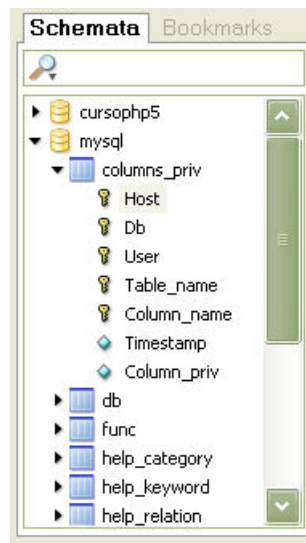


- **En la mitad superior** pueden observarse tres elementos:

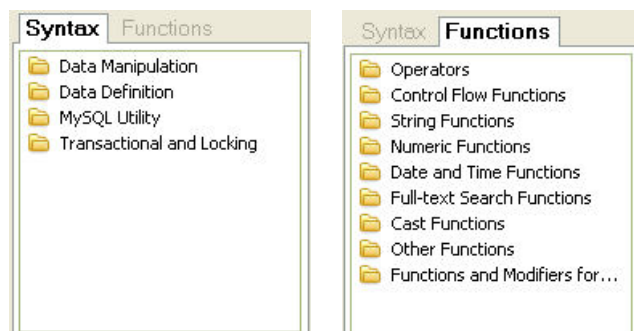
1. **El menú de opciones y los iconos de acceso rápido:**

**File   Edit   View   Query   Scripts   Tools   Help**

2. **La ventana del Schemata (Bases de datos) y Bookmarks (marcadores)** muestra las bases de datos a los que tiene acceso y las tablas y los campos que integran esta base de datos. Si hacemos doble clic sobre el nombre de una base de datos en particular veremos que su texto se cambia a negrita; esto indica que hemos activado esta base de datos y todas las SQL's se aplicarán sobre la misma. La ventana es la siguiente:



3. **La ventana de ayuda donde se describen la sintáxis y las funciones de MySQL es ésta :**

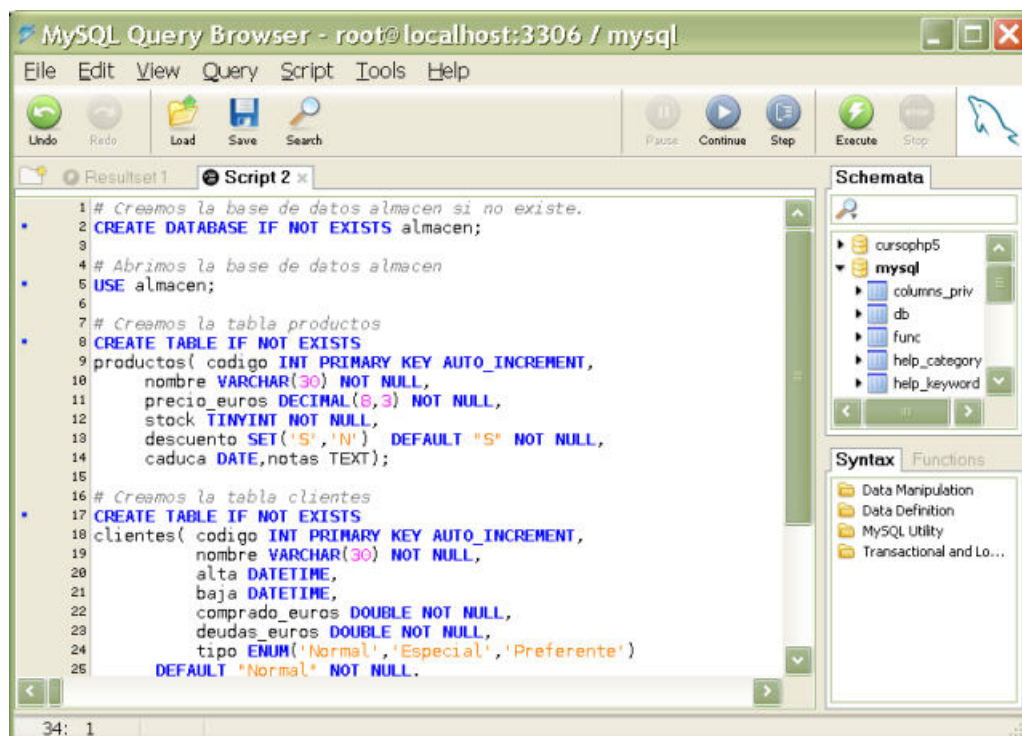


En la ventana siguiente puede verse el cuadro de texto para introducir una **Consulta SQL**. Es importante saber que en este cuadro de texto sólo podremos introducir una única consulta. Notamos que aparece el botón **Go back** y **Next** para recuperar del buffer los comandos anteriores y siguientes. El botón **Execute** sirve para ejecutar la consulta. La ventana es ésta:





Si deseamos ejecutar un script con muchas consultas SQL's entonces debemos hacer clic en la opción **File/New Script Tab** del *Menú* principal. La ventana que se muestra es la siguiente:

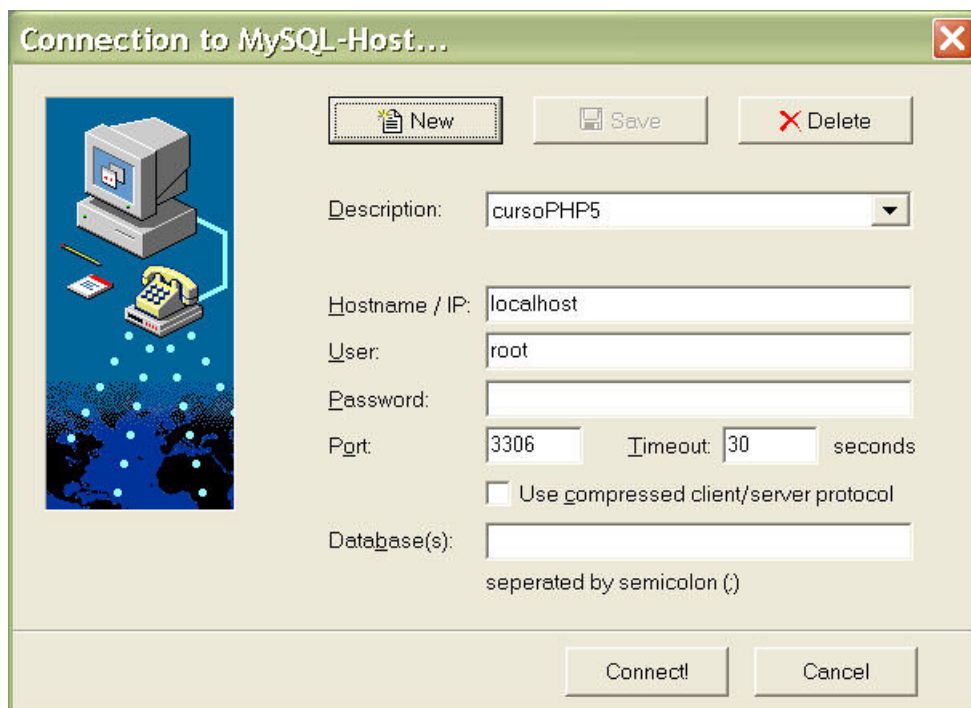


La base de datos con el nombre **mysql** es propia de MySQL y es imprescindible para el funcionamiento del servidor de datos. Por eso, no deben borrarse ni alterarse su estructura o modificarse la información que contienen. En ellas la base de datos MySQL guarda el sistema completo de permisos, privilegios, vistas y zonas horarias. Además, la base de datos **cursophp5** tampoco debe borrarse por ser necesaria para que el curso funcione correctamente.

- **En la parte inferior** de la ventana principal van apareciendo de forma automática mensajes que indican si la ejecución de la SQL ha ido bien mostrando un mensaje de error en caso contrario.

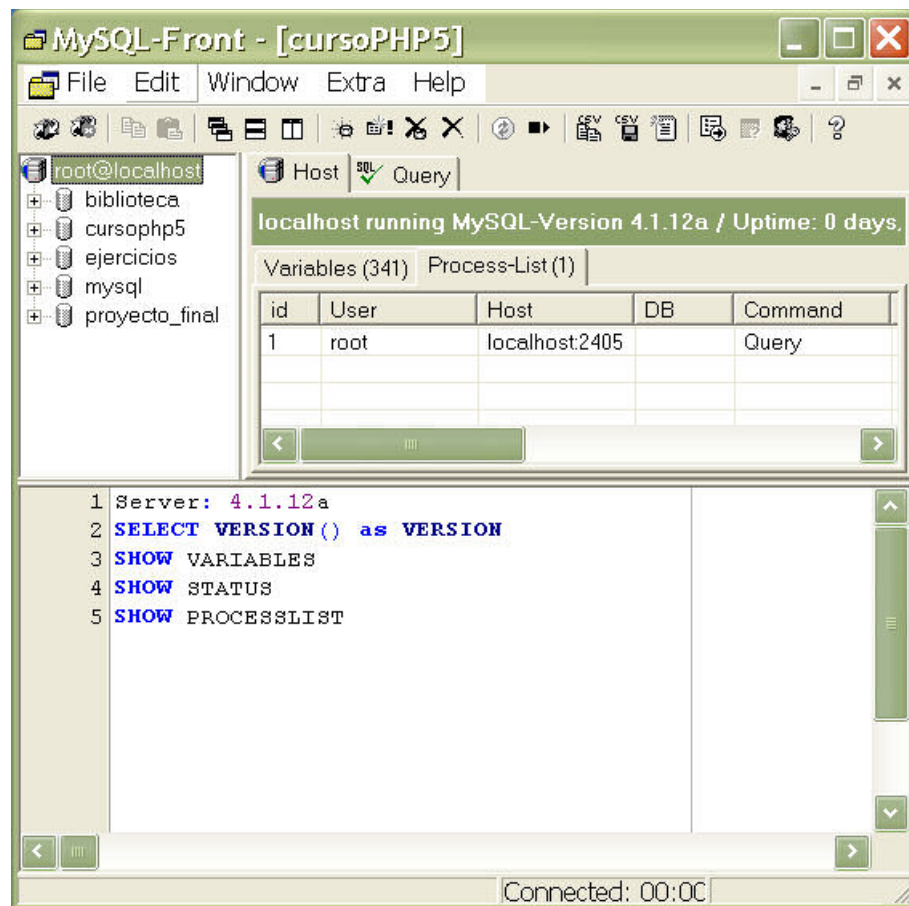
## MySQL -Front

Si has instalado el curso en Windows 98/Me se instalará este cliente MySQL. Al arrancar el programa **Cliente MySQL**, se muestra en la pantalla la ventana siguiente de conexión:



Aceptamos los datos que están puestos por defecto: descripción de la conexión (**cursoPHP5**), nombre del servidor (**localhost**), usuario (**root**) y la contraseña (*password*) en blanco. El usuario **root** (*raíz*) es el que tiene todos los permisos y privilegios de acceso, sin restricciones, a las bases de datos, como si fuera el administrador. Así pues, no hay que cambiar nada, sino sólo pulsar sobre el botón **Connect!** y seguir.

A continuación, entramos ya en la pantalla de esta aplicación, que es la siguiente:



- **En la mitad superior** pueden observarse tres elementos:

1. **El menú de opciones y los iconos de acceso rápido:**

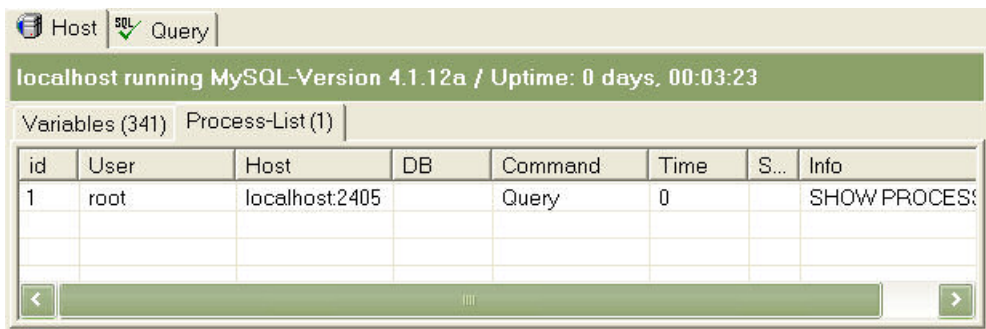
**File    Edit    Window    Extra   Help**



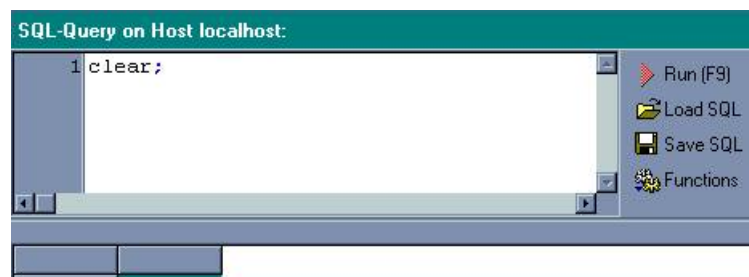
2. **La ventana del usuario y del servidor** al que está conectado con las bases de datos a los que tiene acceso:



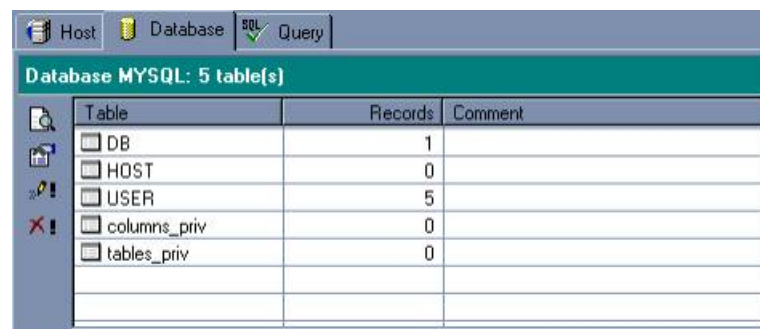
3. La ventana donde se describen y relacionan los elementos que integran el elemento activo de la ventana anterior:



Aquí vemos que el servidor (pestaña **Host**) tiene dos usuario (**admin** y **root**). Si activamos la pestaña **SQL Query** (**Consulta SQL**) vemos que no hay ninguna, pero podemos **escribir una y guardarla** (**Save**), **Abrir una ya existente** (**Load SQL**), **Ejecutar la actual** (**Run F9**) o **Acceder a las funciones** que necesitemos (**Functions**), como puede verse en la ventana siguiente:



Si activamos **en la ventana izquierda** la **base de datos MYSQL**, vemos esta ventana:



Como puede verse, se muestran ahora las tablas que integran esta base de datos. Seleccionado una, se puede **Ver su datos** (**View Data**), **Mostrar las propiedades** de la tabla (**Show Table-Properties**), **Insertar un registro** (**Insert Record**) o **Vaciar la tabla** (**Empty Table**).

- **En la mitad inferior** de la ventana principal van apareciendo de forma automática los comandos que se ejecutan a medida que nosotros vamos realizando operaciones cómodamente desde las opciones de los menús o pulsando sobre los iconos. Conviene fijarse en cada línea, pues así sabremos cómo debemos escribir después cada comando al codificar los *scripts* en lenguaje PHP, que es realmente el objetivo.

**Conviene que tomemos nota de lo siguiente:**

1. En el directorio `C:\cursoPHP5\bin\MySQL4.1\share` están los mensajes de error del servidor para los distintos idiomas. En `C:\cursoPHP5\bin\MySQL4.1\bin` están los ficheros ejecutables y

en `C:\cursoPHP5\bin\MySQL4.1\data` se hallan las bases de datos que hayamos creado, una en cada carpeta.

2. Como hemos dicho, para cada base de datos que nosotros creamos, **MySQL crea un directorio con el nombre que le hemos asignado a la base de datos**. Dentro de este directorio, por cada tabla que definamos MySQL crea tres archivos: `<nombre de la tabla>.MYD`, que contiene los datos, `<nombre de la tabla>.MYI`, que contiene las claves e índices, y `<nombre de la tabla>.frm`, que contiene la estructura de la tabla.

3. Como las bases de datos de MySQL son ficheros de un directorio, **podemos realizar copias de seguridad utilizando las herramientas de compresión que habitualmente usamos en nuestro sistema**.

Creemos que cualquier persona acostumbrada a programar y a utilizar gestores de bases de datos, como debe ser el alumno que se haya matriculado en este curso, no tendrá problemas en aprender rápidamente el manejo del **Cliente MySQL**. Por eso, no vamos a dar más explicaciones sobre el mismo, pero recomendamos practicar sus diferentes opciones y operaciones creando alguna base de datos que contenga varias tablas, como puede ver que hemos hecho nosotros con la base de datos **kursophp5** y sus tablas. Ésta será la mejor forma de familiarizarse con este gestor de datos y de asimilar los comandos de manejo de tablas y su sintaxis.

Cuando el alumno analice la base de datos **kursophp5**, no debe cambiar nada, para que cualquier otro alumno o alumna que haga el curso en el mismo ordenador a otra hora no se vea afectado por sus operaciones. Como trabajo de prácticas, puede crear otra base de datos similar que denomine como su nombre de correo (por ejemplo, `jfer001`) y trabajar con ésta añadiendo tablas, modificándolas, completándolas, etcétera. Al menos cuando acabe el curso o cuando se dé de baja temporal o total, procure copiar esa base a un disco propio, si quiere conservarla, y eliminarla después del disco duro, para no saturar innecesariamente el ordenador.

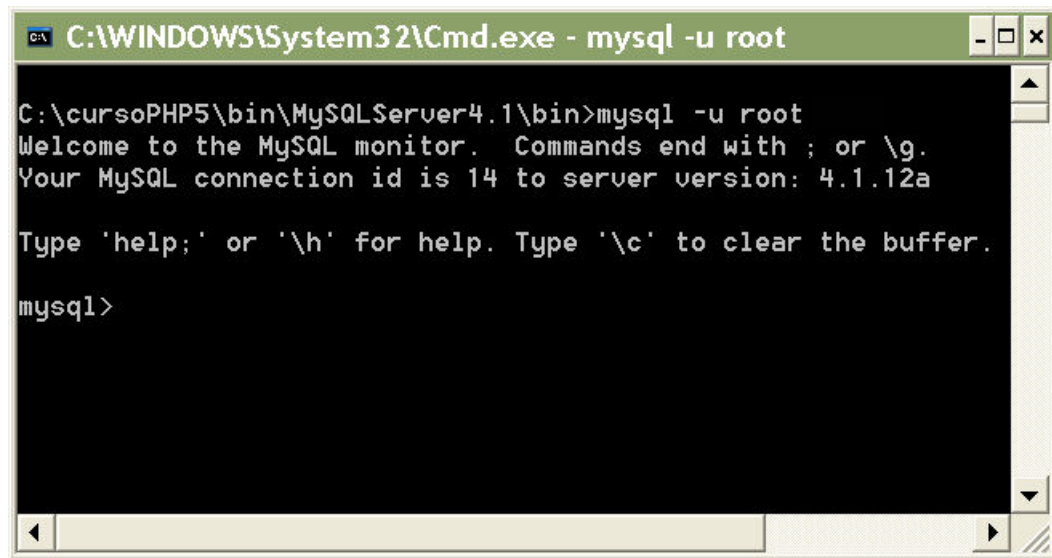
## 5 - Principales sentencias de MySQL

En este apartado vamos a estudiar las sentencias fundamentales que permiten tratar bases de datos de tipo MySQL.

Vamos a utilizar las principales sentencias de este gestor de datos escribiendo detrás del *prompt* cada orden y observando con la aplicación **Cliente MySQL** los efectos que produce la interpretación de cada comando.

Así pues, para estudiar este tema, hay que activar el **Servidor MySQL 4.1** y el **Cliente MySQL** para observar intuitivamente lo que pasa y el programa **mysql.exe**, que se ejecuta en una ventana del DOS. Para arrancarlo debemos escribir `mysql.exe -u root` desde una ventana de comandos en el directorio `C:\cursoPHP5\bin\MySQL4.1\bin`.

Al arrancar este último, se presenta la ventana siguiente:



## Creación de una bases de datos

Para trabajar con una base de datos, en primer lugar hay que crearla. Se hace con la orden `create database` seguida del **nombre que queremos darle**. Para crear la base de datos “**almacen**”, escribimos

```
create database almacen;
```

Podemos comprobar que ha sido creada de cuatro formas diferentes:

1. **Leyendo el mensaje que devuelve MySQL indicando que se ha hecho :**

```
Query OK, 1 row affected (0.00 sec)
```

2. **Mirando con el Explorador de Windows** que se ha creado una carpeta nueva con el nombre “**almacen**” en el directorio `C:\cursoPHP5\bin\MySQL4.1\data`.

3. Arrancando la aplicación **Cliente MySQL** (con la opción **Cliente MySQL** de la carpeta del curso) si no lo está o seleccionando con el botón derecho sobre **Schemata** la opción **Refresh** si está ya arrancada. Se verá que hay una nueva base de datos denominada “**almacen**” que aún no tiene ninguna tabla en su interior.

4. Ejecutando en la propia ventana del **MySQL monitor** la orden `show databases`, que estudiamos a continuación.

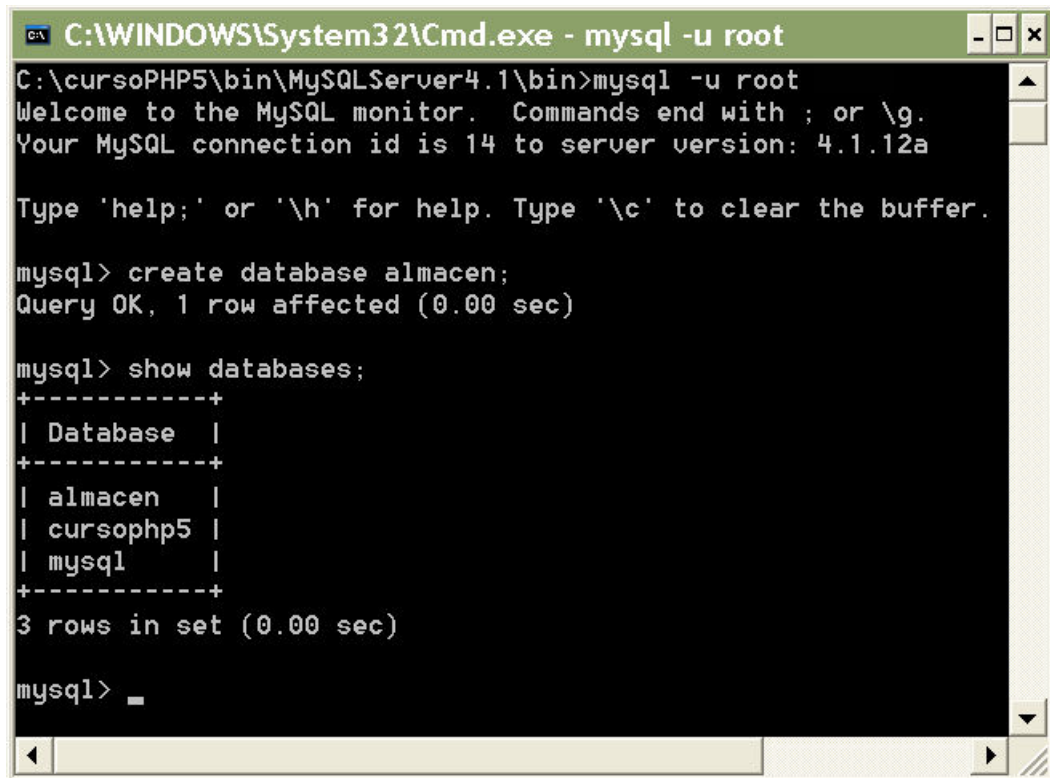
Hemos explicado esto para que el alumno trate de comprobar el resultado de las sentencias que escriba y vea cómo se comportan. Si creamos esta base de datos desde la aplicación **Cliente MySQL** pulsando con el botón derecho sobre **Schemata** en la opción **Create New Schema**, se pregunta en un cuadro de diálogo el nombre de la base de datos.

Como puede ocurrir que ya exista un base de datos con el mismo nombre que la se pretende crear, en cuyo caso se perdería la anterior con todos los datos, lo mejor es usar la sintaxis condicional `create database if not exists almacen;`. De esta forma, si la base de datos existe, no se crea una nueva destruyendo la anterior.



## Mostrar todas las bases de datos

La sentencia `show databases;` se usa para mostrar todas las bases de datos que haya en el servidor MySQL. Podemos observarlo en la ventana siguiente:



```

C:\WINDOWS\System32\Cmd.exe - mysql -u root
C:\cursoPHP5\bin\MySQLServer4.1\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14 to server version: 4.1.12a

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database almacen;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| almacen  |
| cursophp5 |
| mysql    |
+-----+
3 rows in set (0.00 sec)

mysql> _

```

Recordamos que las bases de datos **mysql** y **cursophp5** son imprescindibles para el funcionamiento correcto del sistema y del curso, por lo cual no deben borrarse nunca.

## Utilizar una base de datos

Para poder usar una base de datos, por ejemplo para acceder a sus tablas, es preciso abrirla antes, operación que se realiza con la sentencia `use <nombre de la base de datos>`. Por ejemplo, si queremos abrir la base de datos “**almacen**”, escribiremos `use almacen` y, si se ha podido abrir, el sistema muestra el mensaje de confirmación `Database changed`. A partir de aquí, ya podemos crear tablas en su interior o acceder a los registros y campos de las mismas. Si usamos la aplicación **Cliente MySQL** basta con hacer doble clic sobre el nombre de la base de datos que queremos usar; veremos que su texto cambia a negrita indicándonos que está activa.

## Crear una tabla

Una vez que hemos abierto la base de datos “**almacen**”, vamos a crear tres tablas sencillas dentro de la misma: “**productos**”, “**compras**” y “**clientes**”. Para crear la tabla “**productos**”, damos la orden siguiente:

```

create table if not exists productos (
codigo INT PRIMARY KEY AUTO_INCREMENT,
nombre VARCHAR(30) NOT NULL,
precio_euros DECIMAL(8,3) NOT NULL,
stock TINYINT NOT NULL,
descuento SET('S','N') NOT NULL DEFAULT 'S',

```

```
caduca DATE,
notas TEXT);
```

Con esta orden hemos creado, dentro de la base de datos “**almacen**”, la tabla “**productos**”, que tiene esta estructura:

<u>Nombre del campo</u>	<u>Tipo de campo</u>	<u>Longitud</u>	<u>Propiedades</u>
codigo	INT	11	PRIMARY KEY AUTO INCREMENT
nombre	VARCHAR	30	NOT NULL
precio	DECIMAL	8,3	NOT NULL
stock	TINYINT	4	NOT NULL
descuento	SET		NOT NULL DEFAULT "S"
caduca	DATE		
notas	TEXT		

La misma información puede obtenerse con la sentencia

```
show columns from productos;
```

como puede verse en esta ventana:

```
mysql> show columns from productos;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| codigo     | int(11)       |      | PRI | NULL    | auto_increment |
| nombre     | varchar(30)   |      |      |          |                 |
| precio_euros | decimal(8,3)  |      |      | 0.000   |                 |
| stock      | tinyint(4)    |      |      | 0        |                 |
| descuento  | set('S','N')  |      |      | S        |                 |
| caduca     | date          | YES  |      | NULL    |                 |
| notas      | text          | YES  |      | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> _
```

Conviene hacer algunas observaciones que el alumno debe tener en cuenta:

1. En la tabla “**productos**” **hemos creado seis campos** cuyo identificador debe ser válido: **codigo**, **nombre**, **precio\_euros**, **stock**, **descuento**, **caduca** y **notas**.



2. **A cada campo le hemos asignado su tipo, longitud y características adecuadas** , según el contenido que vamos a introducir en ellos:

- El campo `codigo` es de tipo numérico entero `INT` y tiene una longitud de 11 dígitos (incluido el signo – si lo hay); no puede ser nulo, es decir, estar vacío; es el índice primario, es decir, el valor único (uno distinto para cada registro) por el que MySQL va a ir ordenando los registros a medida que se den de alta; y se autoincrementa cada vez que se añade un registro nuevo. Observe el alumno que en el tipo de campo no ha sido preciso escribir la longitud, pues MySQL asume por defecto que el tipo `INT` es de 11 posiciones. Los tipos numéricos enteros pueden ser más largos o más cortos, según el número máximo que pretendamos introducir en ese campo: `TINYINT` asigna 4, `SMALLINT` asigna 6, `MEDIUMINT` asigna 9, `BIGINT` asigna 20.
- El campo `nombre` es de tipo `VARCHAR` , tiene una longitud de 30 y no puede ser nulo. Si se observa ahora la sintaxis de la sentencia, se verá que hemos puesto la longitud de este campo entre paréntesis. El valor máximo posible es de 255. El tipo `CHAR` se comporta igual que `VARCHAR` , con la diferencia de que `CHAR` guarda la longitud completa indicada, aunque parte del campo esté vacía, como ya hemos explicado en un apartado anterior.
- El campo `precio_euros` es de tipo `DECIMAL` , lo cual indica que el número que se puede guardar en él tendrá parte entera y parte decimal. Tampoco puede ser nulo. Hay que tomar nota de la sintaxis `precio_euros DECIMAL(8,3)` : entre paréntesis se ponen dos números separados por una coma. El primero ( 8 ) es la longitud del número completo y el segundo ( 3 ) indica el número de decimales que tiene. Para tratar campos en los que haya que escribir números reales MySQL dispone también de diferentes tipos: `FLOAT` (de coma flotante) y `DOUBLE` , que no exigen poner longitud. El primero soporta hasta 11 posiciones incluidos el signo -, el punto decimal, la parte entera y la decimal. El segundo soporta hasta 20 posiciones.
- El campo `stock` es de tipo `TINYINT` , tiene una longitud de 4 y no puede ser nulo. En el mismo sólo podremos registrar como máximo el valor 127 para saber las existencias de un producto.
- El campo `descuento` es de tipo `SET` , admite sólo los valores “S” o “N”, por defecto toma “S” y no puede ser nulo. No precisa ponerle longitud. El tipo `ENUM` se comporta de una forma similar al tipo `SET` y tiene la misma sintaxis. Hay que observar cómo debe escribirse el conjunto de valores que puede contener este campo: entre paréntesis, cada valor entre comillas y separados unos valores de otros por una coma.
- El campo `caduca` es de tipo `DATE` , es decir que va a contener una fecha. No hay que poner longitud. Para este tipo de datos MySQL dispone además de los tipos `TIME` para la hora, minutos y segundos, `DATETIME` , para la fecha y la hora y `TIMESTAMP` para el número de segundos transcurridos desde el 1 de enero de 1970 (época Unix).
- El campo `notas` es de tipo `TEXT` . Este tipo de datos permite introducir gran cantidad de información, de bastantes Mb, al igual que el tipo `BLOB` . No hay que poner longitud.

3. Si escribimos la sentencia de creación de la tabla desde el *prompt* de `MySQL monitor` , dado que es muy larga, podemos pulsar *Intro* antes de escribir ; y en este caso aparece el signo -> en una línea inferior, para indicarnos que la línea de la sentencia sigue en la fila inferior. Sólo cuando pongamos ; al final de la sentencia, ésta intentará ejecutarse.

4. **Utilizar esta forma de trabajar resulta demasiado pesada**, debido sobre todo a que el *prompt* de **MySQL monitor** sólo permite escribir y borrar hacia atrás, por lo cual resulta desesperante si uno se equivoca y, además, no se guarda nada de lo escrito ni siquiera en el bufer del teclado.

Por eso, **lo más cómodo y productivo es crear ficheros de texto**, es decir, *script* MySQL, que se guardan donde nos convenga con la extensión **sql**. En ellos se escriben las sentencias MySQL que deseamos se ejecuten cuando sea necesario.

En las prácticas de esta Unidad cada alumno debe guardar estos ficheros en su carpeta de trabajo. Para crearlos puedes utilizar el **Cliente MySQL** para escribirlo y guardarlo usando la opción del *Menú* **File/Save as...** eligiendo tipo de fichero **SQL Script**. También puede escribir el *script* MySQL con el **Editor dev-PHP** del curso (con \*.\* en la ventana de diálogo **Abrir** se pueden ver todos los ficheros, no sólo los que sean de tipo php) o con cualquier otro procesador de texto plano.

Una vez escrito y guardado el fichero con la extensión **sql**, se puede ejecutar de dos formas:

- Desde una ventana del DOS, ejecutando `mysql.exe -u root` y redireccionando hacia él el fichero **sql**. Por ejemplo, supongamos que hemos creado el *script* `creatabla.sql` y que lo hemos guardado en `C:\cursoPHP5\curso\alumnos`. Nos situamos en una ventana del DOS en el camino `C:\cursoPHP5\bin\MySQL4.1\bin` y escribimos:

```
mysql -u root <C:\cursoPHP5\curso\alumnos\creatabla.sql      (Intro)
```

Si las sentencias son correctas, se ejecutarán y producirán los efectos pretendidos. Si el fichero ejecutable estuviera en un camino y el fichero **sql** en otro, hay que indicar el camino para que el sistema pueda hallar los ficheros indicados.

Si las sentencias del fichero **sql** son muchas y se ejecutan seguidas de forma que el resultado ocupa más de una pantalla, puede utilizarse el comando **More** del DOS para detener la ejecución en cada pantalla. Escribir, entonces,

```
mysql -u root <C:\cursoPHP5\curso\alumnos\creatabla.sql | More      (Intro)
```

De nuevo, reconocemos que este procedimiento no deja de ser pesado y engorroso. Por eso, aconsejamos utilizar el siguiente.

- Desde el **Cliente MySQL** en la opción del *Menú* **File/Open Scripts** seleccionando en Tipo de fichero **SQL Script File ANSI**, abrimos el fichero **sql** y lo ejecutamos. Si no funciona correctamente, podemos arreglarlo en el editor según los mensajes de error que se aparecen en la parte inferior de la ventana.

En el **Ejemplo 1** de esta Unidad puede verse el fichero *script* MySQL con el que hemos creado la base de datos “almacen” y sus tres tablas “productos”, “compras” y “clientes”. **Como no es un fichero de PHP, no puede interpretarse ni generar una página web, sino sólo sirve para crear esta base de datos**. En la Unidad siguiente aprenderemos a hacerlo con funciones de PHP.

Con los ejemplos y ejercicios de esta Unidad, referidos todos a MySQL, **sólo pretendemos que el alumno asimile bien los principales contenidos sobre las bases de datos y sus tablas**.

## Introducir registros en una tabla

Podemos insertar un registro en la tabla “**productos**” con la sentencia siguiente:

```
insert into productos values (0,"Pala",10.55,100,"S",
                             "2001-12-16","Fuerte y barata");
```

Otra forma de hacerlo es escribiendo en un fichero de tipo `txt` la información anterior, por ejemplo, en el fichero `datos_productos.txt`, e incorporando los datos de esta forma:

```
load data local infile "datos_productos.txt"
into table productos;
```

Tanto en la tabla “**productos**” como en “**compras**” y en “**clientes**” hemos introducido tres registros. Conviene que el alumno mire en el **Ejemplo 2** el código para ver cómo pueden completarse los valores de los diferentes campos, sobre todo los de fecha y los numéricos, tanto enteros como reales. Si ejecuta este *script* varias veces seguidas, se añadirán tres registros nuevos cada vez con los mismos datos. Para comprobar que se han cargado los datos correctamente en las tablas desde el **Cliente MySQL** hay que hacer doble clic sobre el nombre de la tabla y luego pulsar el botón **Execute** (Control + Intro).

## Seleccionar o consultar registros dentro de una tabla

La sentencia **select** permite buscar y mostrar los datos de una tabla. Su estructura es bastante rica y compleja. Por ello, aconsejamos a los alumnos que consulten algún libro o dirección de Internet sobre el lenguaje SQL. Aquí sólo vamos a abordar algunas sentencias, las más importantes, para poder incorporar estas órdenes dentro de los *scripts* PHP en la **Unidad** siguiente. En concreto, dentro del propio **Cliente MySQL** puede obtenerse ayuda en inglés sobre los diferentes comandos que admite MySQL. Siempre que lo necesite, el alumno puede consultarlo si no dispone de otro material que le resulte más asequible.

Por ejemplo, en la sección **14.1.7 SELECT Syntax**, se indica la sintaxis siguiente:

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr, ...
  [INTO OUTFILE 'file_name' export_options
   | INTO DUMPFILE 'file_name']
  [FROM table_references
   [WHERE where_definition]
   [GROUP BY {col_name | expr | position}
    [ASC | DESC], ... [WITH ROLLUP]]
   [HAVING where_definition]
   [ORDER BY {col_name | expr | position}
    [ASC | DESC] , ...]
   [LIMIT {[offset,] row_count | row_count OFFSET offset}]
   [PROCEDURE procedure_name(argument_list)]
   [FOR UPDATE | LOCK IN SHARE MODE]]
```

Veamos cómo se usa esta orden en sus cláusulas más importantes y frecuentes:

Aplicada a la base de datos “**productos**”, la sentencia

```
select codigo,nombre,caduca from productos;
```

muestra estos campos de todos los registros de la tabla citada.

Si queremos que la búsqueda sólo se realice entre los 50 primeros registros de la tabla , podemos utilizar la cláusula **limit** así:

```
select codigo,nombre,caduca from productos limit 0,50;
```

Para comprobar cómo funciona esta orden, hay que ejecutarla desde una ventana del DOS poniendo los caminos adecuados. Por ejemplo, suponiendo que hemos creado el *script muestra* y que lo hemos guardado en `C:\cursoPHP5\bin\MySQL4.1\bin` , nos situamos en una ventana del DOS en ese directorio y escribimos:

```
mysql<muestra.sql | More      (Intro)
```

La órdenes siguientes pueden incorporarse en un fichero *script* MySQL de texto e ir ejecutándose una a una como hemos indicado (**File/New Query Tab** en el *Menú* del programa) para poder ver el resultado. En el **Ejemplo 3** hemos colocado varias sentencias **select** con sintaxis variadas para que el alumno o alumna pueda apreciar la selección de registros que se hace según las cláusulas que integren su estructura.

Con la orden anterior se muestran estos datos:

codigo	nombre	caduca
1	Pala	2001-12-16
2	Azadón	2001-08-01
3	Cuerda de pita 100 metros	2001-08-01

Si queremos que aparezcan todos los campos, usamos la sintaxis

```
select * from productos;
```

Si queremos que aparezca el valor del campo `caduca` sin repeticiones, podemos usar la orden

```
select count(distinct caduca) from productos;
```

**Detrás de la cláusula `select` podemos indicar**, como hemos visto, **varios argumentos diferentes** para seleccionar los registros de la consulta:

1. El nombre de uno o de varios campos de una o de varias tablas.
2. El signo `*` para indicar que aparezcan todos los campos.
3. El operador `distinct` para eliminar de la consulta los registros duplicados.
4. Las funciones de agrupamiento `count()` para contar, `sum()` para suma y `avg()` para hallar la media de un campo numérico.

5. Otras funciones, como `max()` para hallar el valor máximo numérico y `min()` para buscar el mínimo.

En el **Ejemplo 3** puede verse su sintaxis y los resultados de la cláusula `select` aplicando diferentes sintaxis.

La cláusula `from` (desde) indica el origen de las tablas en las que está archivada la información. Detrás de ella debe escribirse el nombre de la tabla o tablas, separadas por comas, de las que se quiere obtener la información. Dentro de ésta puede asignarse un alias o nombre más sencillo, breve y manejable con el que deseamos referirnos en adelante a cada tabla. Por ejemplo, en las tablas “**productos**” y “**clientes**” hay un campo denominado “**nombre**”. Para poder identificar cada uno dentro de una misma cláusula `select`, debemos escribir:

```
select productos.nombre,clientes.nombre
from productos,clientes;
```

De una forma más breve, podíamos haber usado un **alias** para cada tabla así:

```
select P.nombre,C.nombre from productos P,clientes C;
```

La cláusula `where` (donde) se usa para seleccionar sólo los registros de una tabla que cumplan una determinada condición. Por ejemplo, la instrucción

```
select * from productos where nombre="Pala";
```

mostrará todos los campos de la tabla “**productos**” que lleven como contenido “**Pala**” en el campo **nombre**.

La condición puede ser compleja usando expresiones en las que intervengan nombres de campos, valores constantes y operadores, como

```
select * from productos
where nombre="Pala" and stock>50 and precio_euros>10;
```

En este caso se mostrarán los registros de la tabla “**productos**” que contengan “**Pala**” en el campo **nombre** y de los que haya en **stock** más de 50 ejemplares cuyo **precio\_euros** sea superior a 10 euros. Se pueden usar también alias para referirnos a las tablas e incluir en la condición campos de diferentes tablas.

Pueden utilizarse, igualmente, funciones dentro de las expresiones, como

```
select nombre,tipo
from clientes where year(alta)>2000;
```

donde mostramos sólo el nombre y el tipo de los registros de la tabla “**clientes**” cuyo año de la fecha de alta sea posterior a 2000.

La cláusula `group by` (agrupados por) permite hacer una consulta sumaria, es decir, en lugar de mostrarse todos los registros, se agrupan los que tengan unas mismas características y se muestra sólo cuántos son. Por ejemplo, la instrucción

```
select descuento,count(*)
  from productos group by descuento;
```

muestra sólo el número de registros de la tabla “**productos**” que tienen descuento y el número de los que no lo tienen.

La cláusula **having** (**si hay**) **permite limitar el número de grupos** poniendo una condición a la integración de los grupos. Debe usarse sólo detrás de **group by** para insertar la condición. Por ejemplo, la instrucción

```
select descuento,count(*)
  from productos group by descuento
  having count(*)>1;
```

muestra sólo el número de registros de la tabla “**productos**” que tienen descuento y que no lo tienen si son más de 1. Así pues, no aparece ahora el producto que no tiene descuento, pues sólo hace un grupo de 1.

La cláusula **order by** (**ordenar por**) **hace aparecer en la consulta los registros ordenados por el campo que se indique** . Por ejemplo, la instrucción

```
select * from productos order by nombre;
```

muestra los registros de la tabla “**productos**” ordenados por el campo nombre. Por defecto, se ordenan de forma ASCendente (de menor a mayor en el código ASCII), pero podemos ordenarlos también al revés añadiendo la cláusula DESC. Si ahora escribimos

```
select * from productos order by nombre desc;
```

los registros de mostrarán ordenados de mayor a menor (en el código ASCII).

También es posible mostrar los registros ordenados por un segundo campo cuando hay coincidencia de dos o más registros en el primero. Por ejemplo, la instrucción

```
select * from productos order by nombre,stock;
```

en el caso de que hubiera dos o más productos con el **nombre** “**Azadón**”, mostraría éstos ordenados según el número, de menor a mayor, del campo **stock**.

Desde la versión 4.0 del servidor MySQL es posible también utilizar las Subqueries (subconsultas). Es decir, utilizar el resultado de una consulta dentro de otra. Por ejemplo, la instrucción

```
SELECT * FROM clientes
  WHERE clientes.codigo IN (SELECT compras.codigo_cliente from compras)
```

obtiene todos los registros de la tabla “**clientes**” que tengan dado de alta algún registro en la tabla “**compras**”.

Además, desde la versión 4.0 del servidor MySQL también es posible también utilizar las JOINS (uniones). Las joins son básicamente la combinación de dos o más registros de diferentes tablas usando alguna sentencia de comparación. Por ejemplo, la instrucción

siguiente muestra el mismo resultado que la anterior

```
SELECT * FROM clientes
LEFT JOIN compras ON clientes.codigo = compras.codigo_cliente
```

Si el alumno o alumna desea aumentar sus conocimientos sobre subqueries o joins puede dirigirse al manual de MySQL donde se muestran la sintaxis y más ejemplos.

Aconsejamos al alumno que tenga ejecutándose la utilidad **Cliente MySQL** a la vez que la ventana del DOS desde la que esté ejecutando las sentencias SQL. Así, desde esta última ventana podrá ir añadiendo y modificando el contenido de los registros de la tablas “**productos**”, “**compras**” y “**clientes**”; luego, desde la ventana de **Cliente MySQL** puede ir viendo los resultados de las sentencias SQL.

### Actualizar los registros de una tabla

Para modificar el contenido de los registros de una tabla, hay que utilizar la sentencia **update**, que tiene esta sintaxis:

```
update <nombre de la tabla> set <nombre del campo>
    = <expresión>,... [where <condición>]
```

Por ejemplo, si queremos que en la tabla “**productos**” dentro del campo **stock** se ponga el valor 120 en aquellos registros que contengan el valor 100, podemos usar la instrucción

```
update productos set stock = 120 where stock=100;
```

Si no hubiéramos usado la condición en la cláusula **where**, se habrían cambiado los valores del campo **stock** de todos los registros de la tabla.

También podemos cambiar el contenido de varios campos a la vez y usar funciones y operaciones para modificar los campos. Por ejemplo, aplicada a la tabla “**clientes**” la instrucción

```
update clientes set baja=now(),
    deudas_euros=deudas_euros*1.10;
```

modifica el actual valor **NULL** del campo **baja** con la fecha y hora actuales, e incrementa el campo **deudas\_euros** en el 10 %, en todos los registros

Si desemos modificar los registros de varias tablas a la vez la sintaxis que debemos usar es la siguiente:

```
UPDATE tabla1, tabla2 ...
SET nombre_columna1=expr1, nombre_columna2=expr2 ...
[WHERE condición]
```

En el **Ejemplo 4** puede verse el uso de la sentencia **update** y los resultados aplicando diferentes sintaxis.



## Borrar registros

La sentencia `delete` permite eliminar registros de una tabla. Su sintaxis es

```
delete from <nombre de la tabla> [where <condición>]
```

Si no se incluye al cláusula `where`, que es opcional, se eliminan todos los registros. Por ejemplo, la instrucción

```
delete from clientes;
```

deja la tabla sin registros, pero no la elimina. No es frecuente eliminar todos los registros de golpe. Por eso, casi siempre se utiliza la cláusula `where` para seleccionar los registros que se desea eliminar. Por ejemplo, la instrucción

```
delete from clientes where deudas_euros=0;
```

elimina los registros de la tabla “**clientes**” que no tengan deudas. En cambio,

```
delete from clientes where codigo=1;
```

elimina el registro que tenga el número 1 como **codigo**.

Si desemos llevar a cabo una eliminación múltiple en varias tablas la sintaxis que debemos usar es la siguiente:

```
delete <nombre de la tablas donde borramos los registros>
from <nombre de la tablas sobre la que hacemos consultas > [where <condición>]
```

Por ejemplo, para eliminar los registros cuyo código de cliente es 1 tanto en la tabla `clientes` como en `compras` escribimos:

```
delete compras, clientes from compras, clientes
where clientes.codigo=1 and compras.codigo_cliente=clientes.codigo;
```

En el **Ejemplo 5** puede verse el uso de la sentencia `delete` y los resultados aplicando diferentes sintaxis.

**NOTA:** es importante tener en cuenta que las sentencias `SELECT`, `UPDATE` y `DELETE` pueden ser utilizadas conjuntamente con subconsultas. Además, todas estas sentencias permiten actuar sobre varias tablas a la vez.

## Crear y destruir índices

Los índices están asociados al fichero de los datos y sirven para mantener en orden los registros de una tabla .

Para cada tabla sólo puede haber un **índice primario**, que ordena los registros por uno o más campos al ir introduciendo los registros en una tabla. Un índice primario identifica los registros de una tabla de forma única. Si se observa las tres tablas creadas por nosotros, “**productos**”, “**compras**” y “**clientes**”, puede verse que hemos generado en todos los casos un índice primario por el campo **codigo**, de forma que éste será el orden natural al ir introduciendo los registros.



Este campo se autoincrementa de forma que, según se van añadiendo registros, cada uno va tomando un número consecutivo.

La cláusula **primary key**, incluida dentro de la sentencia **create table**, permite crear un índice **primario**. También puede verse en el **Ejemplo 1** cómo lo hemos hecho en la instrucción siguiente de creación de la tabla “**productos**”:

```
create table if not exists
  productos( codigo INT PRIMARY KEY AUTO_INCREMENT,
             nombre VARCHAR(30) NOT NULL,
             precio_euros DECIMAL(8,3) NOT NULL,
             stock TINYINT NOT NULL,
             descuento SET('S','N') DEFAULT "S" NOT NULL,
             caduca DATE,
             notas TEXT);
```

Ya hemos comentado que los ficheros de índice tienen la extensión **myi** y el mismo nombre que el fichero que contiene los datos de la tabla.

**También pueden crearse ficheros de índice no primario**, tantos como se necesite. Este tipo de índices no identifica cada registro de forma única, sino que dos o más registros pueden llevar la misma clave. Los ficheros de índice no primario se crean tomando como clave de ordenación uno o más campos de la tabla. Su sintaxis es la siguiente:

```
create [unique|fulltext] index <nombre del índice> on <nombre de la tabla>(nombre del campo,... )
```

La opción **unique** debe usarse cuando queremos que sólo haya un registro con la misma clave, de forma que, al introducir otro con la misma, no se permita hacerlo. La opción **fulltext** sólo puede usarse con campos de tipo **varchar** y **text**, e indica que se tome su texto completo como clave de ordenación. Si se usan como claves campos de tipo **char** y **varchar**, se puede indicar una longitud de la clave de ordenación, sin que sea necesario usar el campo completo. Si se usan como claves campos de tipo **blob** y **text**, es imprescindible indicar una longitud de la clave de ordenación.

Por ejemplo, si queremos crear un fichero de índice denominado **c\_nombre** en el que se ordenen los registros de la tabla “**clientes**” por el campo **nombre**, debemos escribir la instrucción

```
create index c_nombre on clientes(nombre);
```

Si hubiéramos querido usar como clave de orden sólo los 10 primeros caracteres del campo **nombre**, deberíamos haber escrito la instrucción

```
create index c_nombre on clientes(nombre(10));
```

Conviene observar que este índice, como el primario, se guarda también en el mismo fichero de tipo **myi** ya mencionado.

La sentencia **drop index** permite eliminar índices. Tiene esta sintaxis

```
drop index <nombre del índice>
on <nombre de la tabla>
```

Por ejemplo, si queremos eliminar el índice `c_nombre`, debemos escribir

```
drop index c_nombre on clientes;
```

En el **Ejemplo 6** se realizan diferentes operaciones de creación de índices, de mostrar los índices asociados a una tabla y de eliminación de los mismos aplicadas a las tablas “`productos`” y “`clientes`”.

## Eliminar tablas y bases de datos

Para eliminar una tabla hay que usar la sentencia `drop table`, que tiene esta sintaxis:

```
drop table [if exists] <nombre de la tabla>,...
```

Por ejemplo, si queremos eliminar la tabla “`productos`”, hay que usar la instrucción

```
drop table if exists productos;
```

Para eliminar una base de datos hay que usar la sentencia `drop database`, que tiene esta sintaxis:

```
drop database [if exists] <nombre de la base de datos>,...
```

Por ejemplo, si queremos eliminar la base de datos “`almacen`”, hay que usar la instrucción

```
drop database if exists almacen;
```

En el **Ejemplo 7** se eliminan las dos tablas creadas en la base de datos “`almacen`” y la propia base de datos. El alumno puede estudiar en ese código `sql` cómo se aplican las instrucciones comentadas anteriormente.

## Notas importantes

1. Los siete ejemplos de este apartado no son código `php`, por lo cual no pueden interpretarse ni generar páginas `web` como hemos hecho hasta ahora. Por eso, en la ventana del curso sólo aparece el botón **Imprimir**, para que el alumno pueda ver y estudiarlo este código SQL.

2. Para ejecutar estos ejemplos, lo mejor es copiar estos ficheros desde la carpeta `c:\cursoPHP5\curso\curso_ini\capitulos\08unidad6\ejemplos` al directorio de trabajo propio y luego ejecutarlos desde una ventana del DOS, como hemos explicado. Por ejemplo, suponiendo que nuestra carpeta sea `c:\cursoPHP5\curso\alumnos\alfer001` y que queremos ejecutar el Ejemplo 1 (fichero `uni6_eje1.sql`, supongamos), podemos abrir una ventana del DOS, ponernos en el camino `C:\cursoPHP5\bin\mysql4.1\bin` y desde ahí escribir

```
mysql<C:\cursoPHP5\curso\alumnos\alfer001\uni6_eje1.sql | more
(Intro)
```

3. Por otra parte, es muy importante ejecutar los ejemplos en orden, pues en ellos se sigue un proceso lógico: primero se crea la base de datos “`almacén`” y sus tres tablas “`productos`”, “`compras`” y “`clientes`” (ejemplo 1); luego, se incorporan tres registros cada una de las veces que se ejecuta el ejemplo 2; en el ejemplo 3 se utilizan diferentes sintaxis de la sentencia `select` para hacer consultas; posteriormente, se eliminan registros (ejemplo 4); a continuación,

se actualizan los datos de los registros (ejemplo 5); seguidamente, se crean índices y se destruyen (ejemplo 6); y, finalmente, se eliminan las tablas y la base de datos (ejemplo 7). De ahí la necesidad de proceder de forma lógica al ir ejecutando los citados ejemplos.

4. Por otra parte, la utilidad **Cliente MySQL** puede ayudarnos a ver los datos y a comprender más intuitivamente lo que pasa en cada proceso. Para ejecutar scripts complejos con varios comandos SQL seguidos de ";" debes usar la opción del **Menú File/New Script Tab**.

5. Finalmente, queremos insistir en que sólo hemos expuesto lo básico del lenguaje SQL, debido a que el aprendizaje del mismo no es objeto de este curso, sino que se supone que los alumnos y alumnas que se han matriculado en este curso de PHP deben tener ya previamente asimilados los conocimientos básicos necesarios de SQL para realizar el curso de PHP.

## 6 - Resumen

### Hay que saber al final de esta unidad

- Cómo es la arquitectura de una aplicación PHP que utilice bases de datos y mediante qué dos tipos de enlaces puede acceder el servidor a las bases de datos.
- Los conceptos fundamentales de las bases de datos relacionales: diferentes tipos de bases de datos relacionales, organización de la información en filas (registros) y columnas (campos), características de cada campo (nombre, tipo, longitud, si es la clave de un índice, etcétera).
- Las ventajas que aportan las bases de datos al almacenamiento y manejo de la información que contienen.
- Cómo se lleva a cabo el diseño correcto y eficaz de una base de datos.
- Arrancar el servidor de datos MySQL, comprobar que está activado y desactivarlo (echarlo abajo) al acabar la sesión de trabajo del curso.
- Arrancar el programa "MySQL monitor" desde una ventana del DOS y escribir desde su prompt las sentencias SQL básicas para tratar las bases de datos y las tablas.
- Arrancar la utilidad "Cliente MySQL", conectarse al servidor de datos y manejar con cierta soltura las principales opciones de este programa que permiten crear bases de datos y tablas, eliminarlas, acceder

a la información de las mismas y tratarla, etcétera.

- Crear, guardar, abrir y ejecutar Queries con la utilidad anterior.
- Aplicar correctamente las principales sentencias SQL para crear bases de datos y tablas, mostrar las que haya creadas, usarlas, introducir registros en las tablas, consultar su información, actualizarlos, eliminarlos, crear y eliminar índices, así como eliminar tablas y bases de datos.

MENTOR - CNICE MEC 2006