Tutorial de PHP y MySQL COMPLETO

© José Antonio Rodríguez 2000.

http://es.tldp.org/Manuales-LuCAS/manual_PHP/manual_PHP/

1. **Instalación de Apache+PHP+MySQL**
   - Instalación en Windows
   - Instalación en Linux/Unix
2. **Sintaxis en PHP**
   - Mi primer script
   - Variables y Operadores
   - Seriables y Operadores
   - Sentencias de Control
   - Las Tablas
   - Las Funciones
   - Include() y require()
   - Tiempo y fecha
   - Las Clases en PHP
3. **Formularios**
   - Los Formularios
   - Descarga de archivos desde un formulario
4. **Ficheros**
   - Funciones de acceso a ficheros
5. **Comenzando con MySQL**
   - MySQL
   - Funciones PHP de acceso a MySQL
   - Conectar a MySQL desde PHP
   - Creación de una Base de Datos en MySQL
   - Importar bases de datos desde MS Access
   - Mostrar los datos de una consulta
   - Un buscador para nuestra base de datos
6. **Operaciones con registros**
   - Añadir registros
   - Modificar registros
   - Borrar registros
   - Todo a la vez
7. **Conexión a MySQL con ODBC**
   - Instalación de MyDOBC
   - Conexión remota a MySQL con MS Access
   - Exportar tablas desde MS Access a MySQL
   - Importar tablas desde MySQL a MS Access

Instalación de Apache+PHP+MySQL en Windows

En este capítulo describiremos el proceso de instalción de la base de datos MySQL, de un servidor web Apache con PHP, en una máquina con sistema operativo Windows.

Lo primero que debemos hacer es conseguirnos los programas necesarios, y que mejor para ello que diriguirnos a las páginas web (o cualquiera de sus mirros) de los programas en cuestión:

cualquiera de sus mirros) de los programas en cuestión:

- **Apache**: www.apache.org
  - apache_1_3_x_win32.exe
- **MySQL**: www.mysql.com
  - mysql-shareware-3.22.34-win.zip
- **PHP**: www.php.net
  - php-3.0.x-win32.zip

NOTA: La versión para sistemas Windows de MySQL no es gratuita. Por lo que usaremos la versión shareware que está limitada a 30 días.

La instalación de estos programas es muy fácil, PHP y MySQL vienen comprimidos en formato ZIP y sólo los tenemos que descomprimir en una carpeta, mientras que Apache es autoejecutable:

- Descomprimimos PHP en "C:\php3"

- Descomprimimos MySQL en "C:\mysql"
- Hacemos "doble click" en el fichero de Apache y aceptamos el directorio de instalación por defecto "C:\Archivos de Programas\Apache Group\Apache".

Ya tenemos instalados los programas, ahora sólo nos queda hacer unos pequeños ajuste de configuración:

APACHE

de configuración:

APACHE

Editamos el fichero de configuración **http.conf** que se halla en C:\Archivos de Programas\Apache Group\Apache\conf\"

Buscamos la línea donde pone:

```
#ServerName new.host.name
```

Quitamos el comentario (#) y la cambiamos por:

```
ServerName http://localhost
```

Indicamos el directorio de PHP:

```
ScriptAlias /php3 "C:\php3"
```

Definimos la extención de los script PHP:

```
AddType application/x-httpd-php3 .php3
AddType application/x-httpd-php3 .php
AddType application/x-httpd-php3 .phtml
```

Y asignamos la aplicación para las extensiones PHP:

```
Action application/x-httpd-php3 "/php3/php.exe"
```

Por defecto los ficheros que son accesibles desde el navegador se encuentran el la carpeta **htdocs** del directorio de Apache, pero la podemos cambiar:

```
DocumentRoot  "C:\www"
<Directory "C:\www">
......
</Directory>
```

PHP

Para configurar PHP, primero buscamos el fichero php3.ini-distphp3.ini-dist y lo renombramos a php.ini, después lo editamos y le hacemos los siguientes cambios:

Buscamos la expresión "extension_dir" y la cambiamos por:

```
extension_dir = C:\php3
```

Para añadir el soporte para MySQL busca la línea:

```
; extension = php3_mysql.dll
```

Cámbiala por:

```
extension = php3_mysql.dll
```

Copia el fichero php3.ini en "C:\windows\"

Ejecución de los programas:

Pues bien, ya solo nos queda arrancar los programas:

```
C:\Archivos de Progrmas\Apache Group\Apache\apache.exe
C:\mysql\bin\mysqld.exe
#Para la versión shareware
C:\mysql\bin\mysqld-shareware.exe
```

También podemos arrancar el servidor Apache desde el menú de inicio:

```
Inicio->Progrmas->Apache Web Server->Start
```

Para comprobar nuestra instalación crea un fichero llamado test.php3 con la siguiente linea:

```
<?php phpinfo() ?>
```

Colócalo en el directorio de documentos de Apache y llámalo desde el navegadentos de Apache y llámalo desde el navegador. Si lo hemos hecho todo bien nos saldrá una página con todas las variables de PHP.

NOTA:

Cabe destacar que lo que hemos echo es una instalación básica, por lo que recomendamos leer los manuales de las distintas aplicaciones para obtener más detalles sobre la instalación de éstas.

Instalación de Apache+PHP+MySQL en Linux/Unix

En este capítulo describiremos el proceso de instalción de la base de datos MySQL, de un servidor web Apache con PHP, en una máquina con sistema operativo Linux o Unix.

Lo primero que debemos hacer es conseguirnos los paquetes necesarios, y que mejor para ello que diriguirnos a las páginas web (o cualquiera de sus mejor para ello que diriguirnos a las páginas web (o cualquiera de sus mirros) de los programas en cuestión:

- **Apache**: www.apache.org
  - apache-1.3.x.tar.gz
- **MySQL**: www.mysql.com
  - mysql-3_22_22_tar.gz
- **PHP**: www.php.net
  - php-3.0.x.tar-gz

Para poder realizar todo el proceso de instalación has de tener acceso como **root** a la máquin Linux.

Lo primero que debemos hacer un directorio de instalación, aunque lo normal sería que lo hicieramos en **/usr/local**, **/urs/src**, o bien en **/opt**. Como hay que escoger uno, yo voy ha escoger el primero, **/usr/local**, aunque el proceso sería el mismo si nos declináramos por cualquier otro.

Supongamos que ya nos hemos conseguido los paquetes y los tenemos en el directorio **/root/install**, lo primiero que hacemos es descomprimirlos:

```
cd /usr/local
tar zxvf /root/install/apache-1.3.x.tar.gz
tar zxvf /root/instal/mysql-3.22.x.tar.gz
toot/instal/mysql-3.22.x.tar.gz
tar zxvf /root/instal/php-3.0.x.tar-gz
```

Creamos enlaces sencillos (blandos) a código fuente

```
ln -s /usr/local/apache-1.3.x  /usr/local/apache
ln -s /usr/local/mysql-3.22.x /usr/local/mysql
ln -s /usr/local/php-3.0.x /usr/local/php
```

Preparamos la fuentes par al compilación de Apache

```
cd /usr/loca/apache
./configure --prefix=/usr/local/apache
```

Compilamos e instalamos MySQL

```
cd /usr/local/mysql
./configure --witout-debug --prefix=/usr/local/mysql
make
make install
cp /usr/local/support-files/mysql.server /etc/rc.d/init.d/mysql
chmod 755 /etc/rc.d/init.d/mysql
```

Creamos la bases del datos del sistema MySQL

```
/usr/local/mysql/bin/mysql_install_db
```

Arrancamos el servidor MySQL

```
/etc/rc.d/init.d/mysql start/etc/rc.d/init.d/mysql start
```

Asignamos la password del adminstrador (root) de MySQL

```
/usr/local/mysql/bin/mysqladmin -u root password "clave"
```

Ya hemos terminado con MySQL, ahora compilaremos PHP como módulo de Apache.

```
cd /usr/local/php
./configure --with-mysql=/usr/local/mysql \
       --with-apache=/usr/local/apache \
       --enable-track-vars
make
make install
#cp php3.ini-dist /usr/local/lib/php3.ini
```

Compilamos Apache

```
cd /usr/local/apache
 ./configure --prefix=/usr/local/apache \
       --activate-module=src/modules/php3/libphp3.a
       # si hemos compilado PHP4 utilizaremos
       #--activate-module=src/modules/php4/libphp4.a
       # quitar los comentarios para habilitar el módulo de proxy
       #--activate-module=src/modules/proxy/libproxy.a< proxy modules>
make
make install
```

Para definir las extenciones de los scripts PHP, hay que añadir las siguientes lineas en el fichero de configuración de apache (**`httpd.conf`**):

```
AddType application/x-httpd-php3 .php3
AddType application/x-httpd-php3 .php
AddType application/x-httpd-php3 .phtml
```

Ahora ya sólo nos queda arrancar el servidor, pero primero copiamos el script de arranque en /etc/rc.d/init.d

```
cp /usr/local/apache/bin/apachecte /etc/rc.d/init.d/apache
/etc/rc.d/init.d/apache start
```

Para comprobar nuestra instalación crea un fichero llamado test.php3 con la siguiente linea:

```
<?php phpinfo() ?>
```

Colócalo en el directorio de documentos de Apache y llámalo desde el navegador. Si lo hemos hecho todo bien nos saldrá una página con todas las variables de PHP.

NOTA:

Cabe destacar que lo que hemos echo es una **instalación básica**, por lo que recomendamos leer los manuales de las distintas aplicaciones para obtener más detalles sobre la instalación de éstas.

Mi primer script

Una vez que ya tenemos instalados **PHP** y **MySQL**, y el servidor **Apache** configurado para usarlos, podemos comenzar a escribir nuestro primer script en PHP.

Ejemplo script php

```
<html>
<body>
<?php
$myvar = "Hola. Este es mi primer script en PHP \n";
//Esto es un comentario
es mi primer script en PHP \n";
//Esto es un comentario
echo $myvar;
?>
</body>
</html>
```

Una vez escrito esto lo salvamos en un fichero con la extensión **php3** (la nueva versión de PHP, la 4, utiliza la extensión **php**), y lo colocamos en nuestro servidor, **http://mi_servidor/php/test.php3**. Ahora si ponemos esta URL en nuestro navegador veremos una línea con el texto **"Hola. Este es mi primer script en PHP"**.

Lo primero que apreciamos en el script son sus delimitadores. En la primera línea del script vemos **<?php** que nos indica que comienza un script en PHP, y en la última colocamos **?>** para indicar el final del script. Hay que destacar que todas las líneas que se encuentre entre estos delimitadores deben acabar en **punto y coma**, excepto las sentencias de control (if, swicht, while, etc.).

Como en toda programación, es importante poner muchos comentarios, para lo cual si queremos comentar una sola línea tenemos que poner al principio de la línea **//**, si lo que queremos es comentar varias utilizaremos los delimitadores **/* - */**.

Para que el servidor envié texto utilizaremos la instrucción **echo**, aunque también podemos utilizar **printf** de uso similar al del **C** o **Perl**.

Finalmente, vemos que la palabra **myvar** comienza con el signo dólar (**$**) . Este símbolo le indica a **PHP** que es una variable. Nosotros le hemos asignado un texto a esta variable, pero también pueden contener números o tablas (arrays). Es importante recordar que todas las variables comienza con el **signo dólar**. También habréis observado que el texto que le asignamos a la variable termina con **\n**, esto no se imprime sirve para indicarle al navegador una nueva línea.

Variables y operadores

Ahora antes de seguir, vamos a ver un poco de teoría, la sintaxis en PHP.

Variables:

Como vimos antes todas la variables deben precedidas por **signo dólar** ($), y le asignamos contenido con el **signo igual** (=). Con las variables, PHP **distingue** entre mayúsculas y minúsculas, por lo que bles, PHP **distingue** entre mayúsculas y minúsculas, por lo que no es lo mismo `$myvar` que `$Myvar`, éstas son dos variables totalmente distintas.

```
<html>
<body>
<?php
$myvar = "SEVILLA \n";
$Myvar = "MADRID \n";
//Esto imprimirá SEVILLA
echo $myvar;
//Esto imprimirá MADRID
ECHO $Myvar;
?>
</body>
</html>
```

Como veis he utilizado dos formas de escribir **echo**, en mayúsculas y en minúsculas, para indicaros que **PHP no las distingue** a la hora de usar funciones o sentencias del lenguaje.
El uso de la barra invertida, como en `\n`, no es obligatorio, pero ayuda a la depuración del código que enviamos al navegador, además del `\n` existen otros usos:

`\"` Carácter dobles comillas
`\\` Carácter barra invertida
`\n` Nueva línea
`\r` Retorno de carro

`\t` **Tabulador horizontal**

Constantes:

Las constantes son similares a las variables, con la salvedad de que no llevan el signo dólar delante, y sólo la podemos asignar una vez. Para definir una constantes usaremos la función **define** como sigue:

```
<html>
<body>
<?php
define ("CONSTANTE", "Hola Mundo");
printf (CONSTANTE);
?>
</body>
</html>
```

PHP crea diversas constantes al arrancar, como PHP_VERSION que contiene la versión de PHP, TRUE que le asigna 1 o FALSE que le asigna 0.

Operadores Aritméticos:

`$a + $b` **Suma**
`$a - $b` **Resta**
`$a * $b` **Multiplicación**

```
$a / $b &ss=codigoenlinea>$a / $b   División
$a % $b  Resto de la división de $a por $b
$a++      Incrementa en 1 a $a
$a--      Resta  1 a $a
```

Operadores de Cadenas:

El único operador de cadenas que existen es el de concatenación, el punto. Pero no os asustéis, PHP dispone de toda una batería de funciones que os permitirán trabajar cómodamente con las cadenas.

```
$a = "Hola";
$b = $a . "Mundo";   / / Ahora $b contiene "Hola Mundo"
```

En este punto hay que hacer una distinción, la interpretación que hace PHP de las simples y dobles comillas. En el segundo caso PHP interpretará el contenido de la cadena.

```
$a = "Mundo";
echo = 'Hola $a';   / /Esto escribirá "Hola $a"
echo = "Hola $a";   / /Esto escribirá "Hola Mundo&q; //Esto escribirá "Hola Mundo"
```

Operadores de Comparación:

```
$a <  $b  $a menor que $b
$a >  $b  $a mayor que $b
$a <= $b  $a menor o igual que $b
$a >= $b  $a mayor o igual que $b
$a == $b  $a igual que $b
$a != $b  $a distinto que $b
```

Operadores Lógicos:

```
$a AND $b  Verdadero si ambos son verdadero
$a &&  $b  Verdadero si ambos son verdadero
$a OR  $b  Verdadero si alguno de los dos es verdadero
$a !!  $b  Verdadero si alguno de los dos es verdadero
$a XOR $b  Verdadero si sólo uno de los dos es verdadero
!$a        Verdadero si $a es falso, y recíprocamente
```

Operadores de Asignación:

```
$a =  $b  Asigna a $a el contenido de $b
$a += $b  Le suma a $b a $a
$a -= $b  Le resta a $b a $a
$a *= $b  Multiplica $a por $b y lo asigna a $a
$a /= $b  Divide $a por $b y lo asigna a $a
$a .= $b  Añade la cadena $b a la cadena $a
```

Sentencias de control

Las sentencias de control permiten ejecutar bloque de códigos dependiendo de unas condiciones. Para PHP el 0 es equivalente a Falso y cualquier otro número es Verdadero.

IF...ELSE

La sentencia **IF...ELSE** permite ejecutar un bloque de instrucciones si la condición es **Verdadera** y otro bloque de instrucciones si ésta es **Falsa**. Es importante tener en cuenta q instrucciones si ésta es **Falsa**. Es importante tener en cuenta que la condición que evaluemos ha de estar encerrada entre **paréntesis** (esto es aplicable a todas la sentencias de control).

```
if (condición) {
    Este bloque se ejecuta si la condición es VERDADERA
} else {
    Este boque se ejecuta si la condición es FALSA
}
```

Existe una forma sencilla de usar la sentencia IF cuando no tenemos que usar el ELSE y solo tenenemos que ejecutar una línea de código.

```
if ($a > 4) echo "$a es mayor que 4";
```

IF...ELSEIF...ELSE

La sentencia IF...ELSEIF...ELSE permite ejecuta varias condiciones en cascada. Para este caso veremos un ejemplo, en el que utilizaremos los operadores lógicos.

```
<?php
if ($nombre == ""){
    echo "Tú no tienes nombre";
} elseif (($nombre=="eva") OR ($nombre=="Eva")) {
    echo "
    echo "Tu nombre es EVA";<
} else {
    echo "Tu nombre es " . $nombre;
}
```

SWITCH...CASE...DEFAULT

Una alternativa a IF...ELSEIF...ELSE, es la sentencia SWITCH, la cuál evalúa y compara cada expresión de la sentencia CASE con la expresión que evaluamos, si llegamos al final de la lista de CASE y encuentra una condición Verdadera , ejecuta el código de bloque que haya en DEFAULT. Si encontramos una condición verdadera debemos ejecutar un BREAK para que la sentencia SWITCH no siga buscando en la lista de CASE. Veamos un ejemplo.

```
<?php
switch ($dia) {
    case "Lunes":
        echo "Hoy es Lunes";
        break;
    c
    case "Martes":
        echo "Hoy es Martes";
        break;
    case "Miercoles":
        echo "Hoy es Miercoles";
        break;
    case "Jueves":
        echo "Hoy es Jueves";
        break;
    case "Viernes":
        echo "Hoy es Viernes";
        break;
    case "Sábado":
        echo "Hoy es Sábado";
        break;
    case "Domingo":
        echo "Hoy es Domingo";
        break;
    default:
```

default:

```
        echo "Esa cadena no corresponde a ningún día de la semana";
```

```
}
?>
```

WHILE

La sentencia WHILE ejecuta un bloque de código mientras se cumpla una determinada condición.

```
<?php
$num = 1;
while ($num < 5) {
    echo $num;
    $num++
}
?>
```

Podemos romper un bucle WHILE utilizando la sentencia **BREAK**.

```
<?php
$num = 1;
while ($num < 5) {
    echo $num;
    if ($num == 3){
        echo "Aquí nos salimos \n";
        break
    }
    $num++
}
?>
```

DO...WHILE

Esta sentencia es similar a WHILE, salvo que con esta sentencia primero ejecutamos el bloque de código y después se evalúa la condición, por lo que el bloque de código se ejecuta siempre al menos una vez.

```
<?php
$num = 1;
do {
    echo $num;
    if ($num == 3){
        echo "Aquí nos salimos \n";
        break
    }
    $num++
} while ($num < 5);
?>
```

FOR

El bucle FOR no es estrictamente necesario, cualquier bucle FOR puede ser sustituido fácilmente por otro WHILE. Sin embargo, el bucle FOR resulta muy útil cuando debemos ejecutar un bloque de código a condición de que una variable se encuentre entre un valor mínimo y otro máximo. El bucle FOR también se puede romper mediante la sentencia **BREAK**.

```
<?php
for ($num = 1; Snum <=5; $num++){
    echo $num;
    if ($num == 3){
        echo "Aquí nos salimos \n";
        break
    }
```

```
}
?>
```

Las tablas

Las tablas (o array en inglés), son muy importantes en PHP, ya que generalmente, las funciones que devuelven varios valores, como las funciones ligadas a las bases de datos, lo hacen en forma de tabla.

En PHP disponemos de dos tipos de tablas. El primero sería el clásico, utilizando índices:

```
<?php
$ciudad[] = "París";
```

$ciudad[] = "París";

```
$ciudad[] = "Roma";
$ciudad[] = "Sevilla";
$ciudad[] = "Londres";
print ("yo vivo en " . $ciudad[2] . "<BR>\n");
?>
```

Esta es una forma de asignar elementos a una tabla, pero una forma más formal es utilizando la función **array**

```
<?php
$ciudad = array("París", "Roma", "Sevilla", "Londres");
//contamos el número de elementos de la tabla
$numelentos = count($ciudad);
//imprimimos todos los elementos de la tabla
for ($i=0; $i < $numelentos; $i++)
{
   print ("La ciudad $i es $ciudad[$i] <BR>\n");
}
?>
```

Sino

```
?>
```

Sino se especifica, el primer índice es el **cero**, pero podemos utilizar el operador **=>** para especificar el indice inicial.

```
$ciudad = array(1=>"París", "Roma", "Sevilla", "Londres");
```

Un segundo tipo, son las **tablas asociativas**, en las cuáles a cada elemento se le asigna un valor (key) para acceder a él.

Para entenderlo, que mejor que un ejemplo, supongamos que tenemos una tabla en al que cada elemento almacena el número de visitas a nuestra web por cada día de la semana.

Utilizando el método clásico de índices, cada día de la semana se representaría por un entero, 0 para lunes, 1 para martes, etc.

```
$visitas[0] = 200;
$visitas[1] = 186;
```

si usamos las tablas asociativas sería

```
$visitas["lunes"] = 200;
$visitas["martes"] = 186;
```

o bien,

```
$visitas = array("luodigo">$visitas = array("lunes"=>200; "martes"=>186);
```

Ahora bien, recorrer una tabla y mostrar su contenido es sencillo utilizando los índices, pero ¿cómo hacerlo en las tablas asociativas?. La manipulación de las tabas asociativas se a hace través de funciones que actúan sobre un puntero interno que indica la posición. Por defecto, el puntero se situa en el primer elemento añadido en la tabla, hasta que es movido por una función:

current - devuelve el valor del elemento que indica el puntero
pos - realiza la misma función que **current**
reset - mueve el puntero al **primer** elemento de la tabla
end - mueve el puntero al **último** elemento de la tabla
next - mueve el puntero al elemento **siquiente**
prev - mueve el puntero al elemento **anterior**
count - devuelve el número de elementos de una tabla.

Veamos un ejemplo de las funciones anteriores:

```
<?php
$semana = array("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domindo");
echo count($semana); //7
//situamos el puntero en el primer elemento
reset($semana);
echo current($semana); //lunes
next($semana);
echo pos($semana); //martes
end($semana)
echo pos($semana); //domingo
prev($semana);
echo current($semana); //sábado
?>
```

Recorrer una tabla con las funciones anteriores se hace un poco lioso, para ello se recomienda utilizar la función **each()**.

```
<?.
<?php
$visitas = array("lunes"=>200, "martes"=>186, "miércoles"=>190, "jueves"=>175);
reset($visitas);
while (list($clave, $valor) = each($visitas))
{
       echo "el día $clave ha tenido $valor visitas<BR>";
}
?>
```

La función **each()**devuelve el valor del elemento actual, en este caso, el valor del elemento actual y su clave, y desplaza el puntero al siguiente, cuando llega al final devuelve **FALSO**, y termina el bucle **while()**.

Tablas multidimensionales

Las tablas multidimensionales son simplemente tablas en las cuales cada elemento es a su vez otra tabla.

```
<?php
$calendario[] = array (1, "enero", 31);
$calendario[] = array (2, "febrero", 28);
$calendario[] = arra
$calendario[] = array (3, "marzo", 31);
$calendario[] = array (4, "abril", 30);
$calendario[] = array (5, "mayo", 31);

while (list($clave, $valor ) = each($calendario)){
{
```

```
        $cadena = $varlor[1];
        $cadena .= " es el mes número " . $valor[0];
        $cadena .= "y tiene " . $varlor[2] . " días<BR>";
        echo $cadena;
}
?>
```

La función **list()** es más bien un operador de asignación, lo que hace es asignar valores a unas lista de variables. En este caso los valores son extraidos de una tabla por la función **each()**.

Las funciones

Muchas veces, cuando trabajamos en el desarrolo de una aplicación, nos surge la necesidad de ejectar un mismo bloque de código en diferentes partes de nuestra aplicación. Una **Función** no es más que un bloque de código al que le pasamos una serie de parámetros y nos devuelve un valor. Como todos los lenguaje de programación, PHP trae una gran cantidad de funciones para nuestro uso, pero las funciones más gran cantidad de funciones para nuestro uso, pero las funciones más importantes son las que nosotros creamos.

Para declara una funcion debemos utilizar la instrucción **function** seguido del nombre que le vamos a dar, y después entre parentesis la lista de argumentos separados por comas, aunque también habrá funciones que no recogan ningún argumento.

```
function nombre_de_funcion (arg_1, arg_2, ..., arg_n)
{
    bloque de código
}
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo (lo que antes hemos llamado bloque de código) de una función, incluso otras funciones y definiciones de clases.

En PHP no podemos redefinir una función previamente declarada, y además en PHP3, las funciones deben definirse siempre antes de que se invoquen, en PHP4 este requerimiento ya no existe.

La instrucción RETURN

Cuando invocamos una función, la ejecución del programa pasa a ejecutar las líneas de código que contenga la función, y una vez terminado, el programa continua su ejecución desde el punto en que fué llamada la función.

Existe una manera de terminar la ejecución de la función aunque aún haya código por ejecutar, mediante el u haya código por ejecutar, mediante el uso de la instrución **return** terminamos la ejecución del código de una función y devolvemos un valor. Podemos tener varios **return** en nuestra función, pero por lo general, cuantos más **return** tengamos menos reutilizable será nuestra función.

```
<?php
function mayor ($x, $y)
{
    if ($x > $y) {
        return $x." es mayor que".$y;
    } else {
        return $y." es mayor que".$x;
    }
}
?>
```

Aunque quedariá mejor:

```
<?php
function mayor ($x, $y)
{
    $msg = "";
```

```
        if ($x > $y) {
            $msg = $x." es mayor que".$y;
        } else {
            $msg = $y." es mayor que".$x;
        }
        return $msg;
    }
    ?>
```

Con la instrución **return** puede devolverse cualquier tipo de valor, incluyendo tablas y objetos. PHP solo permite a las funciones devolver un valor, y para solventar este pequeño problema, si queremos que nuestra función devuelva varios tenemos que utilizar una tabla (array).

Parámetros de las funciones

Existen dos formas de pasar los parámetros a una función, por **valor** o por **referencia**.

Cuando pasamos una variable por **valor** a una funcion, ocurra lo que ocurra en ésta en nada modificará el contenido de la variable. Mientras que si lo hacemos por **referencia**, cualquier cambio acontecido en la función sobre la variable lo hará para siempre.

E variable lo hará para siempre.

En PHP, por defecto, las variables se pasan por valor. Para hacerlo por referencia debemos anteponer un **ampersand** (**&**) a la variable.

```
<?php
function suma ($x, $y)
{
    $x = $x + 1;
    return $x+$y;
}
$a = 1;
$b = 2;

//parámetros por valor
echo suma ($a, $b);    // imprimirá 4
echo $a;    // imprimirá 1

//parámetros por referencia
echo suma (&$a, $b);    // imprimirá 4
echo $a;      //imprimirá 2
?>
```

Si queremos que un parámetro de una función se pase siempre por referencia debemos anteponer un ampersand (**&**) al nombre del parámetro en la definición de la función.

En PHP podemos definir valores por defecto para los parámetro de una función. Estos valores tienen que ser una expresión constante, y no una variable o miembro de una clase. Además cuando usamos parámetros por defectos, éstos deben estar a la derecha de cualquier parámetro sin valor por defecto, de otra forma PHP nos devolverá un error.

```
<?php
function suma ($x=1, $y)
{
    $x = $x + 1;
    return $x+$y;
}
?>
```

Si ejecutarmos esta función nos daría error, ya que hemos dado a **$x** el valor **1** por defecto y la hemos colocado a la izquierda de un parámetro que no tiene valor por defecto. La forma correcta es:

```
<?php
```

```
<?php

    function suma ($y, $x=1)
    {
        $x = $x + 1;
        return $x+$y;
    }
    ?>
```

Cabe destacar que PHP3 no soporta un número variables de parámetros, pero PHP4 sí.

Llegados a este punto, damos un paso atrás y volvemos a las variables, para distinguir entre variables estáticas (**static**) y globales (**global**). Las variables estáticas se definen dentro de una función, la primera vez que es llamada dicha función la variable se inicializa, guardando su valor para posteriores llamadas.

```
<?php
function contador ()
{
    static $count = 0;
    $count = $count + 1;
    return $count;
}
echo contador()."<BR>"; // imprimirá 1
echo contador()."<BR>"; // imprimirá 2
echo contador()."<BR>"; // imprimirá 3
?>
```

Las variables globales, no se pueden declarar dentro de una función, lo que hacemos el llamar a una variable que ya ha sido declarada, tomando el valor que tenga en ese momento, pudiendo se modificado en la función.

```
<?php
var $a = 1;
function ver_a()
{
    global $a;
    echo $a."<BR>"; // imprimirá el valor de $a

    $a += 1; // sumamos 1 a $a
}
echo ver_a(); // imprimirá 1
echo ver_a(); // imprimirá 2
$a = 7;
echo ver_a(); // imprimirá 7
echo ver_a(); // imprimirá 8
?>
```

Funciones Variable

PHP soporta el concepto de **funciones variables**, esto es significa que si una variable tiene unos parentesis añadidos al final, PHP buscará un función con el mismo nombre que el contenido de la variable, e intentará ejecutarla.

```
<?php
function imprime($texto) {
    echo $texto . "\n";
}
function imprimeNegrilla($texto){
    echo "<B>$texto</B>\n";
}
```

```
$MiFunc = "imprime";
$MiFunc("Hola");  //imprimirá Hola
$MiFunc = "imprimeNegrilla";
$MiFunc("Hola");  //imprimirá Hola
?>
```

Recursión

PHP t>Recursión

PHP también permite la recursión, es decir, una función se puede llamar así misma. Para aclarar el concepto de recursión, vamos a crear una función que comprueba si un número es entero o no.

Un número que no sea entero (7'4), tiene una parte entera y otra decimal (comprendida entre 0 y 1), lo que vamos a hacer para comprobar si un número es entero o no, será restarle 1 al número en cuestión hasta que nos que demos sin parte entera, y entoces comprobaremos si tiene parte decimal (un poco lioso todo ésto).

```
<?php
function esEntero($numero) {
    if ($numero > 1) {
        return (esEntero($numero -1));

    } elseif ($numero < 0) {
        /* como los núm. son simétricos
        chequeamos lo convertimos a positvo */
        return (esEntero((-1) * $numero -1));

    } elseif (($numero > 0) AND ($numero < 1)) {
        return ("NO");
    } else {
        /* el cero es entero por definición */
        return ("SI");
    }

} //fin function

echo "¿Es  0  un número entero? ".esEntero(0)."\n";
echo "¿Es  3.5 un número entero? ".esEntero(3.5)."\n";
echo "¿Es  -7  un número entero? ".esEntero(-7)."\n";
echo "¿Es -9.2 un número entero? ".esEntero(9.2)."\n";

?>
```

Cómo ahorrarnos líneas de código

En las lecciones anteriores hemos aprendido el uso básico de las funciones de PHP para trabajar con MySQL. En esta lección y sucesivas vamos a ver nuevas funciones que nos facilitan y potencian nuestras páginas web.

Por lo general, todos nuestros script tienen partes de código iguales, las funciónes include() y **require()** nos van ahorrar muchas de estas lineas de código. Ambas funciones hacen una llamada a un determinado fichero pero de dos maneras diferentes, con **include()**, insertamos lo que contenga el fichero que llamemos de manera literal en nuestro script, mientras que con **require()**, le decimos que el script necesitará parte de código de se encuentra en el fichero que llama **require()**.

Como todo esto es un poco lioso, veamos unos ejemplos que nos lo aclará.

```
<?php
include ("header.inc");
echo "Hola Mundo";
include ("footer.inc");
?>
```

Si tenemos encuenta que el fichero **header.inc** contiene:

```
<html>
<body>
```

y el fichero **footer.inc** contiene:

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

Nuestro script sería equivalente a:

```
<html>
<body>
<?php
<?php
echo "Hola Mundo";
?>
</body>
</html>
```

Ahora veamos el script de ejemplo para la función **require()**:

```
<?php
require ("config.inc");
include ("header.inc");
echo $cadena;
include ("footer.inc");
?>
```

Donde el fichero **config.inc** tendría algo como ésto:

```
<?php
$cadena = "Hola Mundo";
?>
```

Tiempo y fecha

En esta lección vamos a ver como algunas funciones relacionadas con el tiempo y la fecha, así como algunos ejemplos prácticos.

time

Devuelve el numero de segundos transcurridos desde el 1 de Enero de 1970. A esta forma de expresar fecha y hora se le denomina **tmestamp.**

date(formato, timestamp)

La funció **date** devuelve una cte(formato, timestamp)

La funció **date** devuelve una cadena formateada según los código de formato. Si no le pasamos la variable timestamp nos devuelve la cadena formateada para la fecha y la hora actual.

Los códigos de formato para la función date son:

CODIGO	DESCRIPCIÓN
a	am o pm
A	AM o PM
d	Día del mes con ceros
D	Abreviatura del día de la semana (inglés)
F	Nombre del mes (inglés)

h	Hora en formato 1-12
H	Hora en formato 0-23
i	Minutos
j	Día del mes sin ceros
l	Dia de la semana
m	Número de mes (1-12)
M	Abreviatura del mes (inglés)
s	Segundos
y	Año con 2 dígitos
Y	Año con 4 dígitos
z	Dia del año (1-365)

Para ver algunos ejemplos supongamos que ahora es el 7 de abril de 200

Para ver algunos ejemplos supongamos que ahora es el 7 de abril de 2000 a las 14 horas 30 minutos y 22 segundos:

- `date("d-m-Y") ->  07-04-2000`
- `date("H:i:s") ->  14:30:22`
- `date("Y")     ->  2000`
- `date("YmdHis") -> 20000407143022`
- `date("d/m/y H:i a") -> 07/04/00 14:30 pm`
- `date(d-m-Y H:i, time()) -> el momento actual`

mktime(hora, min, seg, mes, dia, año)

La funcion mktime devuelve una variable de tipo timestamp a partir de las coordenadas dadas. La principal utilidad de esta funcion es la de añadir o quitar una determinada cantidad de fecha u horas a una dada.

```
<?PHP
function restarDias($numdias, $date) {
    if (isset($date)) {
        $date = time();
    }
    list($hora, $min, $seg, $dia, $mes, $anno) = explodest($hora, $min, $seg, $dia, $mes, $anno)
    = explode( " ", date( "H i s d m Y"));

    $d = $dia - $numdias;
    $fecha = date("d-m-Y", mktime($hora, $min, $seg, $mes, $d, $anno));
    return $fecha;
    }
    echo                                                                    restarDias(5)."<BR>";
    echo restarDias(10)."<BR>";
?>
```

chekdate (mes, dia, año)

La función chekdate comprueba si una fecha es válida, si es así devuelve TRUE y si no lo es FALSE. Una fecha se considera válida si el año está entre 1900 y 32767, el mes entre 1 y 12, y el dia es menor o igual que número de días total del mes en cuestión.

```
<?PHP
if (checkdate(31, 2, 2000)) {
    echo "La fecha es correcta";
} else {
    echo "La fecha es incorrecta";
}
?>
```

Para el ejemplo anterior nos daría que la fecha es incorrecta, febrero nunca tiene un día 31.

Las clases

Las Clases son máximo exponente de la Programación Orientada a Objetos (POO). PHP no es un lenguaje orientad a objeto, pero implementa las características que permiten definir las clases.

Pero, ¿qué son las Clases y para que sirven?, empecemos por los segundo, sirven hacer el código más legible, y lo que es más importante, reutilizable. Escribir una Clase es sin duda más largo que escribir el código directamente, pero a la larga es más rentable por su portabilidad a otras , pero a la larga es más rentable por su portabilidad a otras aplicaciones y su mantenimiento.

Las Clases no son más que una serie de variables y funciones que describen y actúan sobre algo. Por ejemplo, vamos a crear la clase **automóvil**, la cual tendrá diversas variables, **$color**, **$modelo**, **$marca**, **$potencia**, **$matricula** y habrá una serie de funciones que actuarán sobre la clase **automóvil** como **Precio()**, **Acelerar()**, **Frenar()**, **Girar()**y **Reparar()**.

Como ejemplo vamos a crear la clase **mysql**, que nos servirá para realizar consultas a las bases de datos MySQL.

```
<?php
class DB_mysql {

/* variables de conexión */
var $BaseDatos;
var $Servidor;
var $Usuario;
var $Clave;


/* identificador de conexión y consulta */
var $Conexion_ID = 0;
var $Consulta_ID = 0;

/* número de error y texto error */
var $Errno = 0;
var $Error = "";


/* Método Constructor: Cada vez que creemos una variable
de esta clase, se ejecutará esta función */
function DB_mysql($bd = "", $host = "localhost", $user = "nobody", $pass = "") {
    $this->BaseDatos = $bd;
    $this->Servidor = $host;
    $this->Usuario = $user;
    $this->Clave = $pass;
}

/*Conexión a la base de datos*/
function conectar($bd, $host, $user, $pass){

    if ($bd != "") $this->BaseDatos = $bd;
    if ($host != "") $this->Servidor = $host;
    if ($user != "") $this->Usuario = $user;
    if ($pass != "") $this->Clave = $pass;

    // Conectamos al servidor
    $this->Conexion_ID = mysql_connect($this->Servidor, $this->Usuario, $this->Clave);
    if (!$this->Conexion_ID) {
        $this->Error = "Ha fallado la conexión.";
        return 0;
    }

    //seleccionamos la base de datos
    if (!@mysql_select_db($this->BaseDatos, $this->Conexion_ID)) {
        $this->Error = "Imposible abrir ".$this->BaseDatos ;
        return 0;
    }
```

```
        /* Si hemos tenido éxito conectando devuelve
        el identificador de la conexión, sino devuelve 0 */
        return $this->Conexion_ID;
    }

    /* Ejecuta un consulta */
    function consulta($sql = ""){

        if ($sql == "") {
            $this->Error = "No ha especificado una consulta SQL";
            return 0;
        }

        //ejecutamos la consulta
        $this->Consulta_ID = @mysql_query($sql, $this->Conexion_ID);

        if (!$this->Consulta_ID) {
            $this->Errno = mysql_errno();
            $this->Error = mysql_error();
        }
        /* Si hemos tenido éxito en la consulta devuelve
        el identificador de la conexión, sino devuelve 0 */
        return $this->Consulta_ID;
    }

    /* Devuelve el número de campos de una consulta */
    function numcampos() {
            return mysql_num_fields($this->Consulta_ID);
    }

    /* Devuelve el número de registros de una consulta */
    function numregistros(){
            return mysql_num_rows($this->Consulta_ID);
    }

    /* Devuelve el nombre de un campo de una consulta */
    function nombrecampo($numcampo) {
            return mysql_field_name($this->Consulta_ID, $numcampo);
    }

    /* Muestra los datos de una consulta */
    function verconsulta() {

        echo "<table border=1>\n";

        // mostramos los nombres de los campos
        for ($i = 0; $i < $this->numcampos(); $i++){
            echo "<td><b>".$this->nombrecampo($i)."</b></td>\n";
        }
        echo "</tr>\n";
        // mostrarmos los registros

        while ($row = mysql_fetch_row($this->Consulta_ID)) {
            echo "<tr> \n";
            for ($i = 0; $i < $this->numcampos(); $i++){
                echo "<td>".$row[$i]."</td>\n";
            }
            echo "</tr>\n";
        }

    }


    } //fin de la Clse DB_mysql
    ?>
```

Como habreis observado, para crear una clase utilizamos la sentencia **class**, y además hemos creado una función con el mismo nombre que la clase, a esa función se le llama **constructor** y se ejecutará cada vez que definamos una

variable de esa clase. No es obligatorina variable de esa clase. No es obligatorio crear un **constructor** en una definición de clase.

Otra cosa importante en las clases es el operador **->**, con el que indicamos una variable o método (parte derecha del operador) de una clase (parte izquierda del operador). Para hacer referencia a la clase que estamos creando dentro de su definición, debemos utilizar **this**.

Y ahora veamos un ejemplo de la clase que hemos creado, y supongamos que el código anterior lo hemos guardado en un fichero llamado **clase_mysql.inc.php**.

```
<body>
<html>
<?php
    require ("clase_mysql.inc.php");
    $miconexion = new DB_mysql ;
    $miconexion->conectar("mydb", "localhost", "nobody", "");
    $miconexion->consulta("SELECT * FROM agenda");
    $miconexion->verconsulta();
?>
</body>
</html>
```

Los formularios

Los Formularios no forman parte de PHP, sino del lenguaje estánder de Internet, HTML, pero como éstos van a aperecer muchas veces durante el curso, vamos a dedicar esta algunas líneas a ellos. Lo que viene a continuación es HTML y no PHP.

Todo formulario comienza con la etiqueta **<FORM ACTION="lo_que_sea.php" METHOD="post/get">**. Con . Con **ACTION** indicamos el script que va procesar la información que recogemos en el formulario, mientras que **METHOD** nos indica si el usuario del formulario va ha enviar datos (**post**) o recogerlos (**get**). La etiqueta **<FORM>** indica el final del formulario.

A partir de la etiqueta **<FORM>** vienen los campos de entrada de datos que pueden ser:

Cuadro de texto:

```
<input type="text" name="nombre" size="20" value="jose">
```

Cuadro de texto con barras de desplazamiento:

```
<textarea rows="5" name="descripcion" cols="20">Es de color rojo</textarea>
```

Casilla de verificación:

```
<input type="checkbox" name="cambiar" value="ON">
```

Botón de opción:

```
<input type="radio" value="azul" checked name="color">
```

Menú desplegable:

```
<select size="1&qu class="codigo"><select size="1" name="dia">
<option selected value="lunes">lunes</option>
<option>martes</option>
<option value="miercoles">miercoles</option>
</select>
```

Boton de comando:

```
<input type="submit" value="enviar" name="enviar">
```

Campo oculto:

```
<input type="hidden" name="edad" value="55">
```

Este último tipo de campo resulta especialmente útil cuando que remos pasar datos ocultos en un formulario.

Como habrás observado todos lo tipos de campo tienen un modicador llamado **name**, que no es otro que el nombre de la variable con la cual recogeremos los datos en el script indicado por el modificador **ACTION** de la etiqueta **FORMFORM,** con value establecemos un valor por defecto.

A continuación veamos un ejemplo, para lo cual crearemos un formulario en HTML como el que sigue y lo llamaremos formulario.htm:

```
<HTML>
<BODY>
<FORM METHOD="post" ACTION="mis_datos.php">
    <input type="hidden" name="edad" value="55">
    <p>Tu nombre <input type="text" name="nombre" size="30" value="jose"></p>
    <p>Tu sistema favorito
    <select size="1" name="sistema">
        <option selected value="Linux">Linux</option>
        <option value="Unix">Unix</option>
        <option value="Macintosh">Macintosh</option>
        <option value=&qmargin-left: 75"><option value="Windows">Windows</option>
    </select></p>
    <p>¿Te gusta el futbol ? <input type="checkbox" name="futbol" value="ON"></p>
    <p>¿Cual es tu sexo?</p>
    <blockquote>
        <p>Hombre<input type="radio" value="hombre" checked name="sexo"></p>
        <p>Mujer <input type="radio" name="sexo" value="mujer"></p>
    </blockquote>
    <p>Aficiones</p>
    <p><textarea rows="5" name="aficiones" cols="28"></textarea></p>
    <p><input type="submit" value="Enviar datos" name="enviar">
    <input type="res-left: 50"> <input type="reset" value="Restablecer" name="B2"></p>
</FORM>
</BODY>
<HTML>
```

Y ahora creemos el script PHP llamado desde le formulario mis_datos.php:

```
<?PHP;
if ($enviar) {
    echo "Hola <b>" . $nombre . "</b> que tal estás<BR>\n";
    echo "Eres " . $sexo . "<BR>\n";
    echo "Tienes " . $edad . "<BR>\n";
    echo "Tu sistema favorito es " . $sistema . "<BR>\n";
    if ($futbol) {
        echo "Te gusta el futbol <BR>\n";
    } else odigo" style="margin-left: 50">} else {
        echo "NO te gusta el futbol <BR>\n";
    }
    if ($aficiones != "") {
        echo "Tus aficiones son: <BR>\n";
        echo nl2br($aficiones);
    } else {
        echo "NO tienes aficiones <BR>\n";
    }
}
echo "<a href='formulario.htm'>VOLVER AL FORMULARIO</a>"
?>
```

Una vez rellenados los datos del formulario, pulsamos el botón Enviar datos, con lo que le campo enviar toma lo que su etiqueta **value** indica, es decir enviar="Enviar datos". En nuestro script lo primero que evaluamos es que se haya enviado el formulario, y para ello nada mejor que comprobar que la variable **$enviar** no está vacia. Le ponemos el

signo dolar delante a enviar, ponemos el signo dolar delante a enviar, ya que en PHP todas las variables se les refiere con este signo.

Pero y fusionaramos el código de ambos fichero, nos ahorrariamo uno. Si la variable **$enviar** está vacia, enviamos el formulario.

```
<?PHP;
if ($enviar) {
    echo "Hola <b>" . $nombre . "</b> que tal estás<BR>\n";
    echo "Eres " . $sexo . "<BR>\n";
    echo "Tienes " . $edad . "<BR>\n";
    echo "Tu sistema favorito es " . $sistema . "<BR>\n";
    if ($futbol) {
        echo "Te gusta el futbol <BR>\n";
    } else {
        echo "NO te gusta el futbol <BR>\n";
    }
    if ($aficiones != "") {
        echo "Tus aficiones son: <BR>\n";
        echo nl2br($aficiones);
    } else {
        echo "NO tienes aficiones <BR>\n";
    }
    echo "<a href='$PHP_SELF'>VOLVER AL FORMULARIO</a>"

} else {
    <HTML>
    <BODY>
    <FORM METHOD="post" ACTION="<?PHP echo $PHP_SELF ?>">
        <input type="hidden" name="edad" value="55">
        <p>Tu   nombre   <input   type="text"   name="nombre"   size="30"   nombre"   size="30"
        value="jose"></p>
        <p>Tu sistema favorito
        <select size="1" name="sistema">
            <option selected value="Linux">Linux</option>
            <option value="Unix">Unix</option>
            <option value="Macintosh">Macintosh</option>
            <option value="Windows">Windows</option>
        </select></p>
        <p>¿Te gusta el futbol ? <input type="checkbox" name="futbol" value="ON"></p>
        <p>¿Cual es tu sexo?</p>
        <blockquote>
            <p>Hombre<input type="radio" value="hombre" checked name="sexo"></p>
            <p>="codigo"   style="margin-left:   100"><p>Mujer   <input   type="radio"   name="sexo"
            value="mujer"></p>
        </blockquote>
        <p>Aficiones</p>
        <p><textarea rows="5" name="aficiones" cols="28"></textarea></p>
        <p><input type="submit" value="Enviar datos" name="enviar">
        <input type="reset" value="Restablecer" name="B2"></p>
    </FORM>
    </BODY>
    </HTML>

<?PHP
} //fin IF
?>
```

La variable de entorno **$PHP_SELF**, es una variable de entorno que nos devuelve el nombre del script que estamos ejecutando. Y por último, hacer notar el uso de la función **nl2br()**, **nl2br()**, con la cuál sustituimos los retornos de carro del texto, los cuáles no reconocen los navegadores, por la etiqueta **
**.

Descarga de archivos

Vamos a ver un caso especial, como descargar un archivo desde un formulario. Para ello utilizaremos una etiqueta **INPUT** de tipo **FILE**, soportada a partir de las versiones de los navegadores Nestcape Navigato 2.0 e Internet Explorer 4.0.

El formulario debe usar el método **post**, y el atributo **post,** y el atributo enctype debe tener el valor multipart/form-data. Además al formulario debemos añadirle un campo oculto de nombre MAX_FILE_SIZE, al cuál le daremos el valor en byte del tamaño máximo del archivo a descargar.

```
<FORM ENCTYPE="multipart/form-data" ACTION="7-3.php3" METHOD="post">
    <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="100000">
    <INPUT NAME="archivo" TYPE="file">
    <INPUT TYPE="submit" VALUE="Descargar Archivo">
</FORM>
```

Cuando el formulario es enviado, PHP detectarará automaticamente que se está descargando un archivo y lo colocará en un directorio temporal en el servidor. Dicho directorio será que el que esté indicado en el archivo de configuración php3.ini, o en su defecto en el directorio temporal del sistema.

Cuando PHP dectecta que se está descargando un archivo crea varias variables con el prefijo del nombre del archivo pero con distintas terminaciones. La variable terminada en **_name** contiene el nombre original del archivo, la terminad en **_size** el tamaño en bytes de éste, y la variable terminada en **_type** nos indicará el tipo de archivo si éste es ofrecido por el navegador.

Si el proceso de descarga no ha sido correcto la variable archivo tomará el valor none y **_size** será **0**, y si el proceso ha sido correcto, pero la variable teminada en **_size** da **0**, quiere decir que el archivo a descarga supera el tamaño máximo indicado por **MAX_FILE_SIZE**.

Una vez descargado el archivo, lo primero que debemos hacer es moverlo a otro lugar, pues sino se hace nada con él, cuando acabe la ejecución de la página se borrará.

Veamos un ejemplo de todo lo dicho.

```
<HTML>
<BODY>
<?PHP
if ($enviar) {if ($enviar) {
    if ($archivo != "none" AND $archivo_size != 0){
        echo "Nombre: $archivo_name <BR>\n";
        echo "Tamaño: $archivo_size <BR>\n";
        echo "Tipo: $archivo_type <BR>\n";
        /* para Windows
        if (! copy ($archivo, "C:\\TEMP\\".$archivo_name)) {
            echo "<h2>No se ha podido copiar el archivo</h2>\n";
        }
        */

        /* para Linux/Unix */
        if (! copy ($archivo, "/tmp/".$archivo_name)) {
            echo "<h2>No se ha podido copiar el archivo</h2>\n";
        }

    } elseif ($archivo != "none" AND $archivo_size == 0) {
        echo "<h2>Tamaño de arcft: 75">echo "<h2>Tamaño de archivo superado</h2>\n";
    } else {
        echo "<h2>No ha escogido un archivo para descargar</h2>\n";

    }
    echo "<HR>\n";
}

?>

<FORM ENCTYPE="multipart/form-data" ACTION="<?php echo $PHP_SELF ?>" METHOD="post">
    <INPUT type="hidden" name="MAX_FILE_SIZE" value="100000">
    <p><b>Archivo a descargar<b><br>
```

```
    <INPUT type="file" name="archivo" size="35"></p>
    <p><INPUT type="submit" name="enviar" value="Aceptar"></p>
</FORM>
</BODY>
</HTML>
```

Funciones de acceso a ficheros

Posiblente durante nuestra tarea de programación nos surga la necesidad de obtener datos de un fichero, o bien, de crear uno. PHP nos provee de una extensa gama de funciones de acceso a ficheros.

En esta lección sólo vamos a las funciones básicas, abrir (fopen), cerrar (fclose), leer (fgets) y escribir (fputs). Estas cuatro nos solventaran la mayoria de problemas queibir (fputs). Estas cuatro nos solventaran la mayoria de problemas que nos surgan con respecto al acceso a ficheros.

fopen (archivo, modo)

Con esta función abrimos un fichero, bien sea local o una dirección de internet (http:// o ftp://).

La función **fopen** nos devuelve un valor numérico (indicador de archivo) de tipo **integer** que nos servirará para hacer referencia al archivo abierto.

Con fopen podemos abrir un archivo de los siguientes modos:

r solo lectura

r+ lectura y escritura

w solo escritura. Sino existe el archivo lo crea, si ya existe lo machaca.

w+ lectura y escritura. Sino existe el archivo lo crea, si ya existe lo machaca.

a solo lectura. Sino existe el archivo lo crea, si ya existe empieza a escribir al final del archivo.

a+ lectura y escritura. Sino existe el archivo lo crea, si ya existe empieza a escribir al final del archivo.

```
<?PHP
//abreo"><?PHP
//abre un archivo utilizando el protocolo HTTP
if ( ! fopen("http://www.ciberaula.com/", "r")) {
    echo "El archivo no se puede abrir\n";
    exit;
}
?>
```

Los modos **r**, **r+** , **w** , **w+** colocan el puntero de lectura/escritura a principio del fichero, los modos **a** , **a+** lo colocan al final.

fgets (indicador_archivo, longitud)

La función **fgets** nos devuelve una cadena con la longitud especifica del fichero al que apunta el indicador de archivo.

```
<?PHP
//abre un archivo e imprime cada linea
$archivo = fopen("dlass="codigo">$archivo = fopen("data.txt" , "r");
if ($archivo) {
    while (!feof($archivo)) {
        $linea = fgets($archivo, 255);
        echo $linea;
```

```
        }
    }
    fclose ($archivo);
    ?>
```

La función **feof** devuelve TRUE si puntero de lectura/escritura se encuetra al final del fichero, y FALSE en caso contrario.

fputs (indicador_archivo, cadena)

La función **fputs** escribe una cadena en el fichero indicado. Para escribir en una archivo este debe haber sido previamente abierto. La función **fputs** devuelve TRUE si se ha escrito con éxito, en caso contrario devuelve FALSE.

```
    <?PHP
    //abre un archivo y escribe en él
    $archivo = fopen("data.txt" , "w");
    if ($archivo) {
        fputs ($archivo, "Hola Mundo");
    }
    fclose ($archivo);
    ?>
<);
    ?>
```

fclose (indicador_archivo)

Con esta función cerramos el fichero que nos marca el indicador de archivo, devuelve TRUE si el fichero se cierra correctamente y FALSE sino se ha podido cerrar.

file_exists (fichero)

Esta función devuelve TRUE si el archivo especificado existe, y FALSE en caso contrario.

```
    <?PHP
    if (file_exists("data.txt")) {
        echo "El fichero existe";
    } else {
        echo "El fichero NO existe";
    }
    ?>
```

copy (origen, destino)

La función copy copia un fichero de un lugar (origen) a otro (destino), devuelve TRUE si la copia a tenido éxito y FALSE en caso contrario.

```
    <?PHP
    if (copy("data.txt", "/tmp/data.txt")) {
        echo "El fichero ha sido copiado con éxito";
    } else {
        echo "El fichero NO se higo" style="margin-left: 50">echo "El fichero NO se ha podido
        copiar";
    }
    ?>
```

MySQL

En esta lección vamos a hacer un pequeño recorrido por **MySQL**, por su estructura y forma de trabajar. Para ello suponemos que ya hemos conseguido (por fin...) instalar MySQL en nuestra máquina.

Lo primero que debemos hacer es arrancar MySQL (ver capitulo de instalación):

- Linux: `./mysqld start`
- Windows: `mysqld-shareware.exe`

Todo el sistema de permisos de acceso al servidor, a las bases de datos y sus tablas, MySQL lo almacena en una tabla llamada `mysql`, que como todas estará en el directorio **/data**, a menos que hallamos especificado otro directorio.

En Windows esta tabla se crea con la instalación, pero en Linux/Unix debemos crearla con:

```
/usr/local/mysql/bin/mysql_install_db
```

En la base de datos `mysql` es donde se guadaran todos los permisos y restricciones a los datos de nuestras bases de datos. La principal herramienta de MySQL es `mysqladmin`, la cuál como parece indicar su nombre es la encargada de la administración.

MySQL crea por defecto al usuario **root** con todos los perimsos posibles habilitados, podemos utilizar este usuario como administrador o crear otro, por ejemplo **mysqladmi**. Como el usuario **root** lo crea sin clave de acceso, lo primero que debemos hacer es asignarle una:

```
mysqladmin -u root password "miclave"
```

A partir de ahora cualquier operación que hagamos como root deberemos especifica la clave. Hay que destacar que enter el modificador -p y la clave no debe habeficador -p y la clave no debe haber espacios.

```
mysqladmin -u root -pmiclave
```

Pues bien, ya estamos preparado para crear una base de datos

```
mysqladmin -u root -pmiclave create mibasededatos
```

Para borrarla

```
mysqladmin -u root -pmiclave drop mibasededatos
```

La estructura de MySQL

En el directorio /benc encontraremos ejemplos de script y SQL. En el directoio /share están los mensajes de error del servidor para los distinos idiomas. Los directorios /include y /lib contiene los fichero *.h y las librerias necesarias, en /bin estan los ficheros ejecutables y en /data encontraremos como subdirectorio cada una de las bases de datos que hayamos creado.

Como hemos dicho, para cada base de datos que nostros creamos, MySQL crea un directorio con el nombre que le hemos asignado a la bade de datos. Dentro de este directorio, por cada tabla que definamos MySQL va ha crear tres archivos: **mitabla.ISD, mitabla.ISM, mitabla.frm**

El abla.frm

El archivo con extensión **ISD**, es el contine los datos de nuestra tabla, el **ISM** contiene información acerca de las claves y otro datos que MySQL utiliza para buscar datos en el fichero **ISD**. Y el archivo **frm** contine la estructura de la propia tabla.

Dado que las bases de datos de MySQL son simples ficheros de un directorio, para realizar copias de seguridad, podremos utilizar las herramientas de compresión que habitualmente usamos en nuestro sistema y luego copiarlo a otro lugar, o simplemente esto último.

Seguridad

Como comentamos anteriormente, todo el sistema de permisos MySQL lo guarda en una base de datos llamada `mysql`, la cuál se componen de cinco tablas: **host, user, db, tables_priv, colums_priv**.

La tabla **user** contiene información sobre los usuarios, desde que máquinas pueden acceder a nuestro servidor MySQL, su clave y de sus diferentes permisos. La tabla host nos informa sobre que máquinas podran acceder a nuestro sistema, así como a las bases de datos que tendrán acesso y sus diferentes permisos. Finalmente, las tablas **db, tables_priv, columns_priv** nos proveen de un control individual de las bases de datos, tablas y columnas (campos).

Tabla **user**

CAMPO	TIPO	POR DEFECTO
Host	char(60)	
User	char(16)	
Password	char(16)	
Select_priv	enum('N','Y')	N
Insert_priv	enum('N','Y')	N
Update_priv	enum('N','Y')	N
Delete_priv	enum('N','Y')	N
Create_priv	enum('N','Y')	N
Drop_priv	enum('N','Y')	N
Reload_priv	enum('N','Y')	N
Shutdown_priv	enum('N','Y')	N
Process_priv	enum('N','Y')	N
File_priv	enum('N','Y')	N
Grant_priv	enum('N','Y')	N
References_priv	enum('N','Y')	N
Index_priv	enum('N','Y')	N
Alter_priv	enum('N','Y')	N

Tabla **host**

CAMPO	TIPO	POR DEFECTO
Host	char(60)	
Db	char(32)	
Select_priv	enum('N','Y')	N
Insert_priv	enum('N','Y')	N
Update_priv	enum('N','Y')	N

Delete_priv	enum('N','Y')	N
Create_priv	enum('N','Y')	N
Drop_priv	enum('N','Y')	N
Grant_priv	enum('N','Y')	N
References_priv	enum('N','Y')	N
Index_priv	enum('N','Y')	N
Alter_priv	enum('N','Y')	N

Tabla **db**

CAMPO	TIPO	POR DEFECTO
Host	char(60)	
Db	char(32)	
User	char(16)	
Select_priv	enum('N','Y')	N
Insert_priv	enum('N','Y')	N
Update_priv	enum('N','Y')	N
Delete_priv	enum('N','Y')	N
Create_priv	enum('N','Y')	N
Drop_priv	enum('N','Y')	N
References_priv	enum('N','Y')	N
Index_priv	enum('N','Y')	N
Alter_priv	enum('N','Y')	N

He aquí una breve descripción de los diferentes permisos:

- **Select_priv**: Permite utilizar la sentencia SELECT
- **Insert_priv**: Permite utilizar la sentencia INSERT
- **Update_priv**: Permite utilizar la sentencia UPDATE
- **Delete_priv**: Permite utilizar la sentencia DELETE
- **Create_priv**: Permite utilizar la sentencia CREATE o crear bases de datos
- **Drop_priv**: Permite utilizar la sentencia DROP o eliminar bases de datos
- **Reload_priv**: Permite recargar el sistema mediante *mysqladmin reload*
- **Shutdown_priv**: Permite parar el servidor mediante *mysqladminPermite parar el servidor mediante mysqladmin shutdown*
- ***Process_priv**: Permite manejar procesos del servidor*
- ***File_priv**: Permite leer y escribir ficheros usando comando como SELECT INTO OUTFILE y LOAD DATA INFILE*
- ***Grant_priv**: Permite otorgar permisos a otros usuarios*
- ***Index_priv**: Permite crear o borrar índices*
- ***Alter_priv**: Permite utilizar la sentencia ALTER TABLE*

Si dejamos en blanco los campos **user**, **host** o **db**, haremos referencia a cualquier usario, servidor o base de datos. Conseguiremos el mismo efecto poniendo el símbolo **%** en el campo.

Funciones PHP de acceso a MySQL

En esta leccion vamos a ver todas la funciones que provee PHP para el manejo de bases de datos MySQL. Los ejemplos del manejo de las funciones, los veremos a lo largo del curso.

mysql_affected_rows

```
int mysql_affected_rows(int [link_identifier] );
```

mysql_affected_rows devuelve el número de filas afectado en el último SELECT, UPDATE o DELETE pregunta en el servidor asociado con el identificador de conexión especificado. Si no se e especifica un identificador de conexión, se asume le de la última conexión abierta.

Este orden no es eficaz para las instrucciones SELECT, sólo en instrucciones que modifican archivos. Para recuperar el número de filas vuelto de un SELECT, usa mysql_num_rows.

---

mysql_close

```
int mysql_close(int [link_identifier] );
```

Devuelve: TRUE si se ha cerrado correctamente, FALSE en caso de errror.

mysql_close closes la conexión a la base de datos MySQL asociada al idenficador de conexión especificado. Si no se e especifica un identificador de conexión, se asume le de la última conexión abierta.

Note que esta función no es normalmente necesaria en conexiones no-persistentes (abiertas con mysl_connect) ya que ést se cerrará automáticamente al final de la ejecución del script o página. La función mysql_close no cierra una coneción persistente (abierta con mysql_pconnect()).

Ver tambien: mysql_connect y mysql_pconnect.

---

mysql_connect

```
int mysql_connect(string [hostname] , string [username] , string [password] );
```

Devuelve: un identificador de conexión, o FALSE en caso de error.

sql_connect establece una conexión a un servidor de MySQL. Todos los argumentos son optativos, y si no se especifican, los valores por defecto son (' el localhost', nombre del usuario del usuario que posee el proceso del servidor, la contraseña vacía). La cadena hostname también puede incluir un número del puerto, "hostname:port".

En caso de realizar una segunda llamada a mysql_connect con los mismos argumentos, no se establecerá ninguna nueva conexión, sino se devolverá el identificador de conexión de la ya existente.

La conexión al servidor se cerrará en cuanto la ejecución del script acabe, a menos que la cerremos antes con la función mysql_close.

Ver también: mysql_pconnect y mysql_close.

---

mysql_create_db

```
int mysql_create_db(string database name, int [link_identifier] );
```

La función mysql_create_db intenta crear una nueva base de datos en el servidor asociado con el identificado de conexión especificado.

Ver también: mysql_drop_db.

---

mysql_data_seek

```
int mysql_data_seek(int result_identifier, int row_number);
```

Devuelve: TRUE si toda ha ido bien, y FALSE en caso de error.

La función mysql_data_seek mueve el puntero que indica la fila actual al número de fila de la consulta que indica el identificador. La próxima llamada al mysql_fetch_row o mysql_fetch_array devolvería esa fila.

Ver también: mysql_data_seek.

---

mysql_dbname

```
string mysql_dbname(string result, int i);
```

mysql_dbname devuelve el nombre de la base de datos guardado en posición i de los resultados del indicador de consulta devuelto por la función del mysql_list_dbs. La función del mysql_num_rows puede usarse para determinar cuántos nombres de bases de datos están disponibles.

---

mysql_db_query

```
int mysql_db_query(string database, string query, int link_identifier);
```

Devuelve: un identificador de conexión, o FALSE en caso de error.< caso en FALSE o conexión,>

Ejecuta una consulta en una base de datos. Si el identificador no se especifica, la función intenta encontrar una conexión abierta con el servidor. Si no encuentra una conexión, intentará crear una (similar a mysql_connect( ) sin argumentos).

See also mysql_connect.

---

mysql_drop_db

```
int mysql_drop_db(string database_name, int [link_identifier] );
```

Devuelve: TRUE si toda ha ido bien, y FALSE en caso de error.

Elimina una base de datos del servidor asociado al identificador de conexión.

Ver también: mysql_create_db

---

mysql_errno

```
int mysql_errno();
```

Devuelve el número de error asociado a la última operación realizada.

Ver también: mysql_error

---

mysql_error

```
string mysql_error();
```

Devuelve el texto asociado al error producido en la última operación realizada por la base de datos.

```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_err_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

Ver también: mysql_errno

---

mysql_fetch_array

```
array mysql_fetch_array(int result);
```

Devuelve un array con la información correspondiente al resultado de una consulta especificado por su identificador o 'false' si ya no hay más filas.

Es una versión extendida de mysql_fetch_row ( ). Además de almacenar los datos a través de índices numéricos del array, también lo hace a través de índices asociativos, utilizando los nombres de los campos como claves. Si dos o más columnas del resultado tienen el mismo nombre de campo, la última es la que tiene preferencia. Para acceder a las demás es necesario utilizar el índice numérico o construir un alias para la columna:

select t1.f1 as foo t2.f1 as bar from t1, t2

as foo t2.f1 as bar from t1, t2

Esta función no es más lenta que 'mysql_fetch_row( )'.

Example 1. mysql fetch array

```
<?php
mysql_connect($host,$user,$password);
$result = mysql_db_query("database","select * from table");
while($row = mysql_fetch_array($result)) {
echo $row["user_id"];
echo $row["fullname"];
}
mysql_free_result($result);
?>
```

---

mysql_fetch_field

```
object mysql_fetch_field(int result, int [field_offset] );
```

Devuelve un objeto que contiene la información de los campos que componen un resultado de una consulta. Si no se especifica 'offset', devuelve información sobre el siguiente campo que todavía no ha sido devuelto.

Propiedades del objeto devuelto:

- name - nombre del campo
- table - nombre de la tabla a la que pertenece el campo
- max_length - longitud máxima que puede tomar el campo
- not_null - 1 si el campo no puede tomar valores nulos
- primary_key - 1 si el campo es una clave principal (primary key)
- unique_key - 1 si el campo tiene restricción de unicidad
- multiple_key - 1 si el campo no tiene rest
- multiple_key - 1 si el campo no tiene restricción de unicidad
- numeric - 1 si el campo es numérico
- blob - 1 si el campo es BLOB
- type - tipo de dato (del campo)
- unsigned - 1 si el campo es 'unsigned'
- zerofill - 1 si el campo es rellenado con ceros

See also mysql_field_seek

---

mysql_fetch_lengths

```
int mysql_fetch_lengths(int result);
```

Devuelve: una tabla o FALSE si hay error.

mysql_fetch_lengths almacena en una tabla la longitud de cada campo de una consulta realizada con mysql_fetch_row o mysql_fetch_array. El índice de la tabla comienza en 0.

Ver también: mysql_fetch_row.

---

mysql_fetch_object

```
int mysql_fetch_object(int result);
```

Devuelve: un objeto o FALSE en caso de error.

Esta función es similar a'mysql_fetch_array, solo que los resultados de una consulta, en lugar de una tabla, los devuelve como un objeto. En este caso, sólo se puede acceder a los datos a través de los nombres de sus campos. La velocidad de ejecución es idéntica a la de 'mysql_fetch_array. Para referenciar el valor de un campo debemos utilizar el operador típicos de los objetos (->).< objetos típicos>

Ver también: mysql_fetch_array and mysql_fetch_row.

---

mysql_fetch_row

```
array mysql_fetch_row(int result);
```

Devuelve: una tabla o FALSE si hay error.

Devuelve un tabla con los valores de los campos de la fila actual de la cosulta, la que especificar el indicador (result) , y mueve el puntero interno que marca la fila actual a la siguiente fila, si no hay mas filas devuelve FALSE. El índice de la tabla comienza en 0.

Ver también: mysql_fetch_array, mysql_fetch_object, mysql_data_seek, mysql_fetch_lengths, and mysql_result.

---

mysql_field_name

```
string mysql_field_name(string result, int i);
```

Devuelve el nombre del campo especificado por el índice.

---

mysql_field_seek

```
int mysql_field_seek(int result, int field_offset);
```

Mueve el puntero del campo actual hacia adelante las posiciones actual hacia adelante las posiciones indicadas por 'offset'.

Ver también: mysql_fetch_field.

---

mysql_field_table

```
string mysql_field_table(int result, int field_offset);
```

Devuelve el nombre de la tabla que almacena el campo especificado por el índice ('field_offset').

---

mysql_field_type

```
string mysql_field_type(string result, int field_offset);
```

Devuelve el tipo del campo del indice especificado.

---

mysql_field_flags

```
string mysql_field_flags(string result, int field_offset);
```

Devuelve los especificadores (flags) del campo especificado como una cadena de texto en la que cada especificador se corresponde con una palabra, y éstas van separadas mediante un espacio simple. Se puede analizar la cadena utilizando explode( )

Los especificadores son:

"not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment", "timestamp".

---

mysql_field_len

```
int mysql_field_len(string result, int field_offset);
```

Devuelve la longitud del campo especificado

---

mysql_free_result

```
int mysql_free_result(int result);
```

Sólo debería ser utilizada si la cantidad de memoria utilizada para almacenar el resultado de una consulta es muy grande. Cuando se ejecuta esta función, toda la memoria asociada al resultado se libera.

---

mysql_insert_id

```
int mysql_insert_id(void);
```

Esta función devuelve el ID (identificador) generado para los campos autonuméricos (AUTO_INCREMENTED). El ID devuelto es el correspondiente al de la última operación INSERT.

---

mysql_list_fields

```
int mysql_list_fields(string database, string tablename);
```

Devuelve información sobre la tabla. El valor resultante puede ser utilizado con mysql_field_flags(), mysql_field_len(), mysql_field_name(), and mysql_field_type().

El identificador que devuelve es un entero positivo o '-1' en caso de error. El texto que describe el error se encuentra en $phperrmsg.

---

mysql_list_dbs

```
int mysql_listdbs(void);
```

Devuelve un puntero que contiene las bases de datos disponibles para el servidor actual (msql daemon). Este valor se utiliza con la función mysql_dbname( ).

---

mysql_list_tables

```
int mysql_list_tables(string database, , int [link_identifier]);
```

Devuelve un identificar, el cual pasaremos a la función mysql_tablename para extraer el nombre de las tablas de la base de datos especificada.

---

mysql_num_fields

```
int mysql_num_fields(int result);
```

mysql_num_fields devuelve en número de campos de una consulta.

Ver también: mysql_db_query, mysql_query, mysql_fetch_field, mysql_num_rows.

---

mysql_num_rows

```
int mysql_num_rows(string result);
```

Devuelve el número de filas del resultado de una consulta.

Ver también: mysql_db_query, mysql_query and, mysql_fetch_row.query, mysql_query and, mysql_fetch_row.

---

mysql_pconnect

```
int mysql_pconnect(string [hostname] , string [username] , string [password] );
```

Devuelve: A positive MySQL persistent link identifier on success, or false on error

Devuelve un identificador de conexión persistente o 'false' en caso de error. Las diferencias con respecto a mysql_connect( ) son:

- Primero, la función intenta encontrar una conexión persistente que ya esté abierta con la misma máquina, usuario y password. Si es encontrada, devuelve el identificador de la misma, en lugar de crear una nueva conexión.
- Segundo, la conexión al servidor SQL no será cerrada cuando finalice la ejecución del script, sino que permanece abierta para un uso posterior.

La función mysql_close no cierra una conección abierta con mysql_pconnect. Las conecciones abierta con esta función se llaman "persistentes".

---

mysql_query

```
int mysql_query(string query, int [link_identifier] );
```

Ejecuta una consulta a la base de datos activa en el servidor asociado al identificador de conexión. Si no se especifica, se utiliza la última conexión abierta.fica, se utiliza la última conexión abierta. Si no hay conexiones abiertas la función intenta establecer una.

Esta función devuelve TRUE o FALSE para indicar si las operaciones UPDATE, INSERT o DELETE han tenido éxito. Para la operación SELECT devuelve un nuevo identificador de resultado.

Ver también: mysql_db_query, mysql_select_db, and mysql_connect.

---

mysql_result

```
int mysql_result(int result, int row, mixed field);
```

Devuelve el contenido de la celda de un resultado. El argumento 'field' puede ser un índice o el nombre del campo correspondiente o el nombre del campo de la forma: tabla.campo. Si la columna tiene un alias ('select foo as bar from...') se utiliza el alias en lugar del nombre de la columna.

En lugar de esta función es preferible usar mysql_fetch_row(), mysql_fetch_array(), and mysql_fetch_object(), con la que obtendremos mejor rendimiento.

---

mysql_select_db

```
int mysql_select_db(string database_name, int [link_identifier] );
```

Devuelve: true on success, false on error

Establece la base de datos activa en el servidor. Si no se especifica identificador de conexión se utiliza la última conexión abierta. Si no hay conexiones anexión abierta. Si no hay conexiones activas, la función intenta establecer una. A partir de la llamada a mysql_select_db las llamadas a mysql_query( ) actúan sobre la nueva base de datos activa.

Ver también: mysql_connect, mysql_pconnect, and mysql_query

---

mysql_tablename

```
string mysql_tablename(int result, int i);
```

Toma como argumento un puntero devuelto por la función mysql_list_tables( ) y devuelve el nombre de la tabla asociada al indice i. La función mysql_num_rows( ) puede ser utilizada para determinar el número de tablas.

```
<?php
    mysql_connect ("localhost:3306");
    $result = mysql_listtables ("basededatos");
    $i = 0;
    while ($i < mysql_num_rows ($result)) {
        $tb_names[$i] = mysql_tablename ($result, $i);
        echo $tb_names[$i] . "<BR>";
        $i++;
    }
?>
```

Conectar a MySQL desde PHP

Ya tenemos datos en nuestra Base de Datos (BD), así que con el siguiente script nos conectaremos a la BD del servidor **MySQL** para obtener los datos de un registro.

Conexión a MySQL

```
<html>
<body>
<?php
$linkp>
<?php
$link = mysql_connect("localhost", "nobody");
mysql_select_db("mydb", $link);
$result = mysql_query("SELECT * FROM agenda", $link);
echo "Nombre: ".mysql_result($result, 0, "nombre")."<br>";
echo "Dirección: ".mysql_result($result, 0, "direccion")."<br>";
echo "Teléfono :".mysql_result($result, 0, "telefono")."<br>";
echo "E-Mail :".mysql_result($result, 0, "email")."<br>";
?>
</body>
</html>
```

En la primera línea del script nos encontramos con la función **mysql_connect()**, que abre una conexión con el servidor MySQL en el **Host** especificado (en este caso la misma máquina en la que está alojada el servidor MySQL, **localhost**). También debemos especificar un usuario (nobody, root, etc.), y si fuera necesario un password para el

usuario indicado **(mysql_connect("localhost", "root", "clave_del_root"))**. Si la conexión ha tenido éxito, la función **mysql_connect()** devuelve un identificar de dicha conexión (un número) que es almacenado en la variable **$link**, **sino** ha tenido éxito, devuelve **0** (FALSE).

Con **mysql_select_db()** PHP le dice al servidor que en la conexión **$link** nos queremos conectar a la base de datos **mydb**. Podríamos establecer distintas conexiones a la BD en diferentes servidores, pero nos conformaremos con una.

La siguiente función **mysql_query()**, es la que hace el trabajo duro, usando el identificador de la conexión (**$link**), envía una instrucción SQL al servidor MySQL para que éste la procese. El resultado de ésta operación es almacenado en la variable **$result**.

Finalmente, **mysql_result()** es usado para mostrar los valores de los campos devueltos por la consulta (**$result**). En este ejemplo mostramos los valores del registro 0, que es el primer registro 0, que es el primer registro, y mostramos el valor de los campos especificados.

Creación de una Base de Datos en MySQL

Antes de seguir con **PHP**, vamos a preparar la base de datos (BD) que vamos a utilizar como ejemplo. Como servidor de BD, usaremos **MySQL** un pequeño y compacto servidor de BD, ideal para pequeñas y medianas aplicaciones. **MySQL** soporta el estándar **SQL** (ANSI), y además está disponible para distintas plataformas, incluido las "windows"disponible para distintas plataformas, incluido las "windows".

Una vez instalado **MySQL**, vamos a crear nuestra BD ejemplo. **MySQL** utiliza una tabla de permisos de usuarios, por defecto, en la instalación crea el usuario **root** sin password. Debes crear distintos usuarios con distintos permisos. Entre ellos, el usuario administrador de **MySQL**, con todos los permisos, y como recomendación de seguridad, el usuario **nobody** sólo con el permiso de consultar (**SELECT**), que es el que utilizaremos para conectarnos al servidor de BD en nuestros script.

Vamos a ver dos formas de crear una base de datos y sus tablas. Para ello vamos a crear la base de datos que nos servirá de ejemplo en capitulos siguientes:

1. Linea de comandos

Para crear nuestra base de datos en sistemas Linux/Unix, debemos ser el administrador de **MySQL** o tener el permiso pertinente para crear bases de datos, para ello haremos lo siguiente:

```
mysqladmin create mydb
```

Ya hemos creado una BD, ahora le añadiremos una tabla y algunos registros, para lo cual copia el siguiente texto y sálvalo en un archivo, que podríamos llamar **mydb.dump**.

Crear tabla mybd

```
CREATE TABLE agenda (id INT NOT NULL AUTO_INCREMENT, nombre CHAR(50),direccion CHAR(100),
telefono CHAR(15), email CHAR(50), KEY (id) )\g
INSERT INTO agenda VALUES (0, 'Juan Pérez', 'C/ Laguna, 15. Sevilla', '95.455.55.55',
'juan@agenda.com' )\g
INSERT INTO agenda VALUES (1, 'Luis García', 'C/ Betis, 22. Cádiz', '95.655.66.33',
'luis@agenda.com' )\g
INSERT INTO agenda VALUES (2, 'Carlos Rodríguez', 'C/ Sevilla, 6. Huelva', '95.113.22.77',
'carlos@agenda.com' )\g
```

Debemos tener en cuenta que los comandos de arriba debe escribirse cada uno en una sola línea. Se han separado para aumentar la legibilidad del código.

Ahora desde la línea de comandos ejecuta:

```
cat mydb.dump | mysql mydb
```

Cabe destacar el campo id, que no puede estar vacío, y además es autoincrementable, lo cuál deberemos tener en cuenta a la hora de actualizar y añadir registros. Si no hemos cometido ningún error, ya tenemos nuestra base de datos de ejemplo en el servidor **MySQL**.

MySQL.

2. Script

Una seguna forma de crear las bases de datos y tablas es utilizar las funciones que para ello nos da PHP. Para crear una base de datos tenemos dos opciones, una utilizar la función **mysql_create_db()**, o bien enviado una consulta SQL con la intrucción "**CREATE DATABASE mydb**". Como ejemplo vamos a crear un script **crear_my.php** que creará la anterio base de datos.

```
<html>
<body>
<?PHP
define ("CONSTANTE", "Hola Mundo");
printf (CONSTANTE);
?>
</body>
</html>
$basedatos = "mydb";
//conectamos con el servidor
$link = @mysql_connect("localhost", "root", "");

// comprobamos que hemos estabecido conexión en el servidor
if (! $link){
    echo "<h2 align='center'>ERROR: Imposible establecer conección con el servidor</h2>";
    exit;
}
// obtenemos una lista de las bases de datos del servidor
$db = mysql_list_dbs();

// vemos cuantas BD hay
$num_bd = mysql_num_rows($db);

//comprobamos si la BD que quermos crear exite ya
$existe = "NO" ;
for ($i=0; $i<$num_bd; $i++) {
    if (mysql_dbname($db, $i) == $basedatos) {
        $existe = "SI" ;
        break;
    }
}

// si no existe la creamos
if ($existe == "NO") {
    /* manera 1 */
    if (! mysql_create_db($basedatos, $link)) {
        echo "<h2 align='center'>ERROR 1: Imposible crear base de datos</h2>";
        exit;
    }
    /* class="codigo" style="margin-left: 50"> /* manera 2
    if (! mysql_query("CREATE DATABASE $basedatos", $link)){
        echo "<h2 align='center'>ERROR2: Imposible crear base de datos</h2>";
        exit;
    } */
}

// craamos la tabla
$sql = "CREATE TABLE agenda (";
$sql .= "id INT NOT NULL AUTO_INCREMENT, ";
$sql .= "nombre CHAR(50), ";
$sql .= "direccion CHAR(100), ";
$sql .= "telefono CHAR(15), ";
$sql .= "email CHAR(50), ";
$sql .= "KEY (id) ) ";
```

```
if (@mysql_db_query($basedatos, $sql, $link)) {
    echo "<h2 align='center'>La tabla se ha creado con éxito</h2>";
} else {
    echo "<h2 align='center'>No se ha podido crear la tabla</h2>";
}

?>

</body>
</html>
```

Importar bases de datos desde MS Access

Un caso muy común, como en mi caso, al comenzar a utilizar MySQL, necesitaba migrar mis bases de datos Access de mi sistemas Windows a MySQL en **Linux**. La solución a este problema, nos la aporta un módulo creado por Pedro Freire de CYNERGI.

Los pasos que debemos seguir para instalar este módulo, y su posterior uso son:

1. Abre el archivo de Access **.mdb** que deseas exportar.
2. En la ventana de objetos de la BD selecciona "**Módulos**", y después en "**Nuevo**".
3. Entonces se te abrirá una ventana nueva, borra todo texto (código) que haya escrito.
4. Copia todo el texto del archivo de Pedro Freire y pégalo en el nuevo módulo.
5. Cierra la ventana de código del módulo, selecciona que "**Sí**" desea guardar los cambios y nombra el módulo (p.e. "MexportSQL"). El módulo es ahora parte de tu base de datos Access.
6. Vuelve a abrir el módulo, o pincha con el ratón en "**Diseño**" con nuestro nuevo módulo seleccionado. Mueve el cursor hasta donde aparezca la primera palabra "**Function**", y presiona F5 o selecciona "**Ejecutar**" en el menú.

La ejecución del módulo nos creará dos archivos (esql_add.txt y esql_del.txt) en el directorio C:/temp (el que trae por defecto, pero lo podemos cambiar). A nosotros el archivo que nos interesa es **esql_add.txt**, el cuál como mejor nos parezca deberemos llevárnoslo a nuestra máquinaezca deberemos llevárnoslo a nuestra máquina Linux.

Ahora solo tenemos que seguir los paso que explicamos en el capítulo anterior (Comenzando con MySQL). Primero creamos la base de datos:

```
mysqladmin create mybd
```

Y después volcamos los datos y en la nueva base de datos:

```
cat esql_add.txt | mysql mybd
```

Mostrar los datos de una consulta

Mostrar los datos de una consulta

Ahora que ya sabemos conectar con el servidor de BD, veremos como mostrar los datos por pantalla.

Consulta de la BD

```
<html>
<body>
<?php
$link = mysql_connect("localhost", "nobody"odigo">$link = mysql_connect("localhost", "nobody");
mysql_select_db("mydb", $link);
$result = mysql_query("SELECT nombre, email FROM agenda", $link);
echo "<table border = '1'> \n";
echo "<tr> \n";
echo "<td><b>Nombre</b></td> \n";
echo "<td><b>E-Mail</b></td> \n";
echo "</tr> \n";
while ($row = mysql_fetch_row($result)){
```

```
        echo "<tr> \n";
        echo "<td>$row[0]</td> \n";
        echo "<td>$row[1]</td> \n";
        echo "</tr> \n";
    }
    echo "</table> \n";
    ?>
    </body>
    </html>
```

En este script hemos introducido dos novedades, la más obvia es la sentencia de control while(), que tiene un funcionamiento similar al de otros lenguajes, ejecuta una cosa mientras la condición sea verdadera. En esta ocasión while() evalúa la función mysql_fetch_row(), que devuelve un array con el contenido del registro actual (que se almacena en $row) y avanza una posición en la lista de registros devueltos en la consulta SQL.

La función **mysql_fetch_row()** tiene un pequeño problema, es que el array que devuelve sólo admite referencias numéricas a los campos obtenidos de la consulta. El primer campo referenciado es el 0, el segundo el 1 y así sucesivamente. En el siguiente script solucionaremos este pequeño inconveniente.

Consulta modificada de BD

```
    <html>
    <body>
    <?php
    $link = mysql_connect("localhost", "nobody&ost", "nobody");
    mysql_select_db("mydb", $link);
    $result = mysql_query("SELECT nombre, email FROM agenda", $link);
    if ($row = mysql_fetch_array($result)){
        echo "<table border = '1'> \n";
    echo "<tr> \n";
    echo "<td><b>Nombre</b></td> \n";
    echo "<td><b>E-Mail</b></td> \n";
    echo "</tr> \n";
        do {
            echo "<tr> \n";
            echo "<td>".$row["nombre"]."</td> \n";
            echo "<td>".$row["email"]."</td>\n";
            echo "</tr> \n";
        } while ($row = mysql_fetch_array($result));
        echo "</table> style="margin-left: 50">echo "</table> \n";
    } else {
        echo "¡ La base de datos está vacia !";
    }
    ?>
    </body>
    </html>
```

Esencialmente, este script hace lo mismo que el anterior. Almacenamos en **$row** el registro actual con la función **mysql_fetch_array()** que hace exactamente lo mismo que **mysql_fetch_row()**, con la excepción que podemos referenciar a los campos por su nombre (**$row["email"]**), en vez de por un número.

Con la sentencia **if/else**, asignamos a $row el primer registro de la consulta, y en caso de no haber ninguno (**else**) mostramos un mensaje ("No se ha encontrado..."). Mientras que con la sentencia **do/while**, nos aseguramos que se nos muestren todos los registros devueltos por la consulta en caso de haber más de uno.

Hay que destacar la utilización del punto (**.**), como operador para concatenar cadenas.

Un buscador para nuestra base de datos

Vamos a ver una aplicación, un ejemplo, de todo lo visto hasta ahora. Escribiremos un script que sirva para buscar una determinada cadena (que recibiremos de un formulario, y la almacenamos en la variable **$buscar**), dentro de nuestra base de datos, concretamente dentro del **campo** "nombre".

campo "nombre".

En primer lugar escribiremos el texto HTML de la página web que nos servirá como formulario de entrada, la llamaremos formulario.htm.

Formulario entrada

```
<html>
<body>
<form method = "POST" action = "http://mysevidor/php/buscador.php3">
<strong>Palabra clave:</strong>
<input type="text" name="buscar" size="20"><br><br>
<input type="submit" value="Buscar">
</form>
</body>
</html>
```

El siguiente script de busqueda lo llamaremos **buscador.php3**, y será el encargado de hacer la búsqueda en la BD, y devolver por pantalla los registros encontrados.

Script búsqueda

```
<html>
<body>
<?php
if (!isset($buscar)){
    echo "Debe especificar una cadena a buscar";
    echo &quo
    echo       "<p>Debe       especificar       una       cadena       a       buscar</p>       \n";
    echo "<p><a href=buscador_bd.htm>Volver</p> \n";
    echo "</html></body> \n";
    exit;
}
$link = mysql_connect("localhost", "nobody");
mysql_select_db("mydb", $link);
$sql = "SELECT * FROM agenda WHERE nombre LIKE '%$buscar%' ORDER BY nombre";
$result = mysql_query($sql, $link);
if ($row = mysql_fetch_array($result)){
    echo "<table border = '1'> \n";
    //Mostramos los nombres de las tablas
    echo "<tr> \ndigo" style="margin-left: 50">echo "<tr> \n";
    mysql_field_seek($result,0);
    while ($field = mysql_fetch_field($result)){
        echo "<td><b>$field->name</b></td> \n";
    }
    echo "</tr> \n";
    do {
        echo "<tr> \n";
        echo "<td>".$row["id"]."</td> \n";
        echo "<td>".$row["nombre"]."</td> \n";
        echo "<td>".$row["direccion"]."</td> \n";
        echo "<td>".$row["telefono"]."</td> \n";
        echo "<td><a
        href='mailto:"
        href='mailto:".$row["email"]."'>".
            $row["email"]."</a></td> \n";
        echo "</tr> \n";
    } while ($row = mysql_fetch_array($result));
    echo "<p><a href=buscador_bd.htm>Volver</p> \n";
    echo "</table> \n";
} else {
    echo "<p>¡No se ha encontrado ningún registro!</p>\n";
    echo "<p><a href=buscador_bd.htm>Volver</p> \n";
}
```

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

Lo primero que comprobamos es que el contenido de la variable **$buscar** que recibimos del la página web **formulario.htm** no es una cadena vacia, y esto lo hacemos con la función **isset()** que devuelve '**falso**' si la variable que recibe está vacia. A la función le anteponemos el **signo admiración** (**!**) que es equivalente a un **NOT**, para convertirlo en '**verdadero**' en caso de que la variable esté vacia, y en es caso terminamos la ejecución del script con **exit**.

Lo más importante de este script, es sin duda la sentencia SQL que le enviamos al servidor MySQL, y más concretamente la condición que le imponemos, **WHERE nombre LIKE '%$buscar%'**. Con la sentencia **LIKE** buscamos cualquier ocurrencia de la cadena contenida en **$buscar**, mientras que con los signos de porcentaje (**%**) indicamos el lugar de la coincidencia, por ejemplo, si hubiesemos puesto **nombre LIKE '%$buscar'**, buscariamos cualquier ocurrencia al final del **campo**"nombre", mientras que si hubiesemos puesto **nombre LIKE '$buscar%'**, buscariamos cualquier ocurrencia al principio del campo "nombre".

Las últimas novedade que hemos incorporado, son las funciónes **mysql_fetch_field()**, con el que obtenemos información acerca de las caracteríticas de cada campo, como su nombre, tipo, longitud, nombre de la tabla que los contiene, etc. Pero para ejecutar la función anterior debemos colocar el puntero en el primer campo, y eso lo logramos con la función **mysql_field_seek()**, la cál mueve el puntero interno a la posición indicada.

Añadir registros

En esta lección vamos ver como podemos añadir nuevos registros a nuesta BD. La recogida de datos la vamos a hacer a traves de un interfaz de web. En primer lugar vamos a crear una página web con un simple formulario, con los campos que deseamos.

Formulario inicial añadir BD

```
<html>
<body>
<form method="poodigo"><body>
<form method="post" action="add_reg.php3">
Nombre :<input type="Text" name="nombre"><br>
Dirección:<input type="Text" name="direccion"><br>
Teléfono :<input type="Text" name="telefono"><br>
E-mail :<input type="Text" name="email"><br>
<input type="Submit" name="enviar" value="Aceptar información">
</form>
</body>
</html>
```

Hemos creado un formulario donde recoger los datos, y una vez introducidos ejecutamos un script llamado **add_reg.php3**, pues veamos como es este script.

añadir registros

```
<html>
<body>
<?php
// process form
$link = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$sql = "INSERT INTO agenda (nombre, direccion, telefono, emT INTO agenda (nombre, direccion,
telefono, email) ";
$sql .= "VALUES ('$nombre', '$direccion', '$telefono', '$email')";
$result = mysql_query($sql);
echo "¡Gracias! Hemos recibido sus datos.\n";
</body>
```

```
</html>
```

Como se puede ver, para introducir un nuevo registro, utilizamos la ya conocida función **mysql_query()**, la cúal también usamos para las consultas, y usaremos para las actualizaciones, es decir una señora función. ¡Aaah!, una cosa muy importante, para poder añadir o modificar registros debemos tener permiso para ello en el servidor MySQL, por eso en este caso me conecto como root, pero podría ser cualquier otro usuario.

Para terminar esta lección, una pequeña frivolidad, vamos a combinar la página web de formulario y el fichero de script php3, en un solo fichero que llamaremos **add_reg.php3** (este script no lo comentaré, !algo tendran que hacer ustedes¡o tendran que hacer ustedes¡).

Combinacion de formulario y script

```
<html>
<body>
<?php
if ($enviar) {
    // process form
    $link = mysql_connect("localhost", "root");
    mysql_select_db("mydb",$db);
    $sql = "INSERT INTO agenda (nombre, direccion, telefono, email) ";
    $sql .= "VALUES ('$nombre', '$direccion', '$telefono', '$email')";
    $result = mysql_query($sql);
    echo "¡Gracias! Hemos recibido sus datos.\n";
}else{
    ?>
    <form method="post" action="add_reg.php3">
    Nombre :<input type="Text" name="nombre"><br>
    Dirección:<input type="Text&quo>Dirección:<input type="Text" name="direccion"><br>
    Teléfono :<input type="Text" name="telefono"><br>
    E-mail :<input type="Text" name="email"><br>
    <input type="Submit" name="enviar" value="Aceptar información">
    </form>
    <?php
} //end if
?>
</body>
</html>
```

Modificar registros en MySQL

Primero, para modificar hay que tener permiso para ello en el servidor de BD, el resto nos viene de corrido. Primero seleccionamos el registro que deseamos modificar, y luego, mandamos una consulta con la modificaciones, o ambas cosas a la vez. Suponemos que las modificaciones las recogemos de un formulario como el de la lección anterior .

Modificar registros **opcion A**

```
<html>
<body>
<?php
if (isset($id)){
    // process form
    $link = mysql_connect("localhost", "root");
    mysql_select_db("mydb",$db);
    $sql = "SELECT * FROM agenda WHERE id = $id"
    $result = mysql_query($sql);
    $sql = "UPDATE agenda SET nombre='$nombre',
    direccion='$direccion', telefono='$telefono', email='$email'";
    $result = mysql_query($sql);
}else{
    echo "Debe especificar un 'id'.\n";
}
</body>
</html>
```

O bien, Modificar registros **opcion B**

```
<html>
<body>
<?php
if (icodigo"><?php
if (isset($id)){
    // process form
    $link = mysql_connect("localhost", "root");
    mysql_select_db("mydb",$db);
    $sql = "UPDATE agenda SET nombre='$nombre', direccion='$direccion', ";
    $sql .= "telefono='$telefono', email='$email' WHERE id=$id";
    $result = mysql_query($sql);
}else{
    echo "Debe especificar un 'id'.\n";
}
?>
</body>
</html>
```

Borrar registros

El proceso de borrar un registro es identico al de modificar, solo que en vez de utilizar **UPDATE** utilizamos **DELETE** en la sentenica **SQL**. Por tanto el script quedaría como sigue.

Borrado registros de BD

```
<html>
<body>
<?php

<body>

<?php
if (isset($id)){
    // process form
    $link = mysql_connect("localhost", "root");
    mysql_select_db("mydb",$db);
    $sql = "DELETE FROM agenda WHERE id=$id")
    $result = mysql_query($sql);
}else{
    echo "Debe especificar un 'id'.\n";
}
?>
</body>
</html>
```

Todo a la vez

Como resuemen de todo lo visto hasta ahora, vamos a hacer un script donde se mezcla todo, y algo nuevo. Como ejercico, os dejo que la incorporación del buscador de la lección 5.

```
<html>
<body>
<?php
$link = mysql_connect("localhost", "root");
mysql_select_db("mydb",$link); //Comprobamos si hemos recibido datos del formulario (enviar)
if ($enviar) {
    // Si recibimos un id, modificamos, sino añadimos un registro
    if ($id) {
        $sql = "UPDATE agenda SET nombre='$nombre', direccion='$direccion'";
        $sql.= "telefono='$telefono', email='$email' WHERE id=$id";
        echo "Registro Actualizado<p>";
    } else {
        $sql = "INSERT INTO agenda (nombre, direccion, telefono, email) ";
        $sql .= "VALUES ('$nombre', '$direccion', '$telefono', '$email')";
        echo "Registro Añadido<p>";
```

```
        }
        // Enviamos la sentencia SQL al servidor DB

$result = mysql_query($sql);

    } elseif ($delete) {
        // Borramos un registro
        $sql = "DELETE FROM agenda WHERE id=$id";
        $result = mysql_query($sql);
        echo "Registro Borrado<p>";
    } else {
        /* Esta parte se ejecuta si no hemos
        presionado el boton enviar, es decir no venimos
        de un formulario */
        if (!$id) {
            // Mostramos todos los registros de nuestra BD
            $result = mysql_query("SELECT * FROM agenda",$link)ot;SELECT * FROM agenda",$link);
            while ($myrow = mysql_fetch_array($result)) {
                echo   $myrow["nombre"]."   -   ".$myrow["direccion"]."   -   ".$myrow["Telefono"]."   -
                ".$myrow["email"];
                echo "<a href=\$PHP_SELF?id=".$myrow["id"]."&delete=yes">Borrar</a>";
                echo " - ";
                echo "<a href=\$PHP_SELF?id=".$myrow["id"].">Modificar</a><br>";
            }
        }
        ?>
        <p><a href="<?php echo $PHP_SELF?>">Añadir un registro</a></p>
        <p><form method="post" action="<?php echo $PHP_SELF?>"></p>
        <?php
        if ($id) {
            // editamos el registro seleccionado
            $sql = "SELECT * FROM agenda WHERE id=$id";
            $result = mysql_query($sql);
            $myrow = mysql_fetch_array($result);
            $id = $myrow["id"];
            $nombre = $myrow["nombre"];
            $direccion = $myrow["direccion"];
            $telefono = $myrow["telefono"];
            $email = $myrow["email"];
            // enviamos el id para poder editar el registro
            echo "<input type=hidden name="id" value=$id>";
        }
        ?>
        Nombre:<input type="Text" name="nombre" value="<?php echo $nombre ?>"><br>
        Dirección:<input type="Text" name="direccion" value="<?php echo $direccion ?>"><br>
        Telefono:<input type="Text" name="telefono" value="<?php echo $telefono ?>"><br>
        Email:<input type="Text" name="email" value="<?php echo $email ?>"><br>
        <input type="Submit" name="enivar" value="Enviar Información">
        </form>
        <?php
    } // End If if ($enviar)
    ?>
    </body>
    </html>
```

Cabe destacar el uso de **$PHP_SELF**, esta es una funcion interna de PHP que nos devuelve la dirección del script en el que se ejecuta.

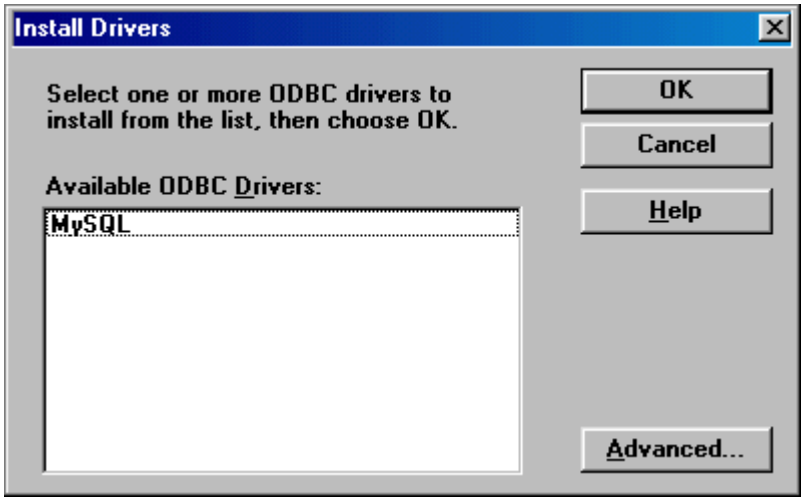
Instalación de MyODBC

En este capitulo describiremos el proceso de instalación del driver MyODBC ODBC 32 bits para Windows 9x y Windows NT. Gracias a este driver podremos acceder desde un cliente Windows, con MS Access por ejemplo, a un servidor remoto MySQL.

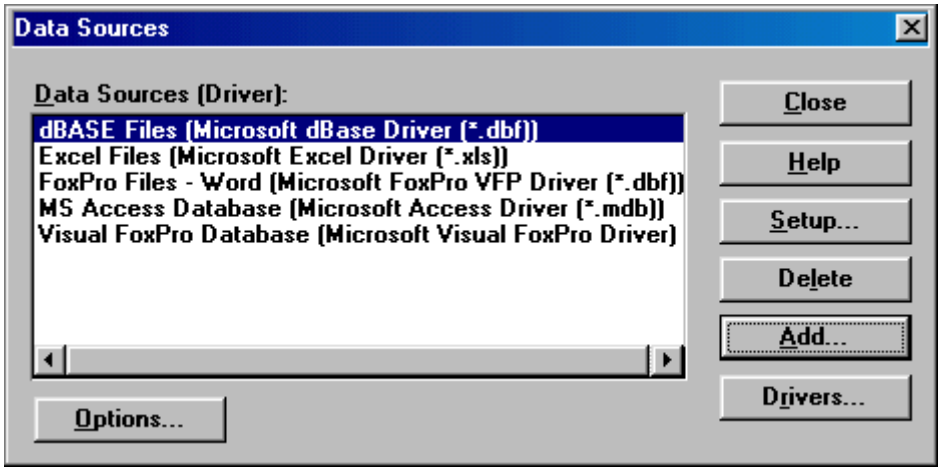
Lo primero que debemos hacer es conseguir el driver MyODBC más reciente, y para ello que mejor que podemos hacer es dirigirnos a la web de MySQL.

Una vez que tenemos el programa en nuestro ordenador (cliente Windows), ejecutamos el programa de instalación, **Setup**.

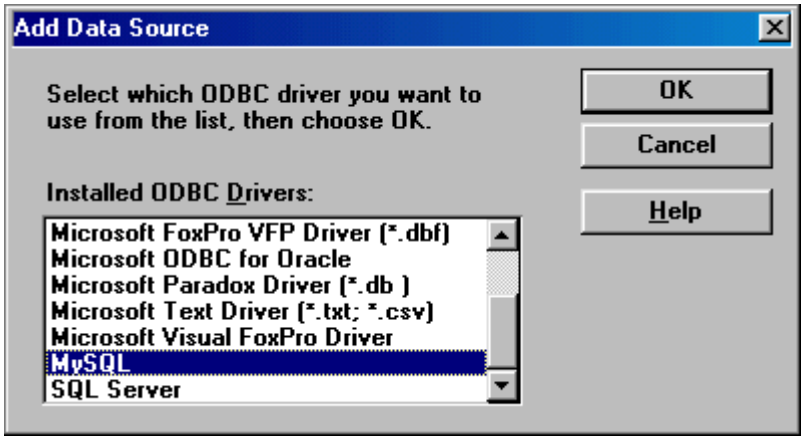
La caja de diálogos de la instalación (**Install Drivers**) permite escoger diversos drivers ODBC para instalar, pero MyODBC solo da como opción el driver para MySQL, el cuál seleccionaremos.



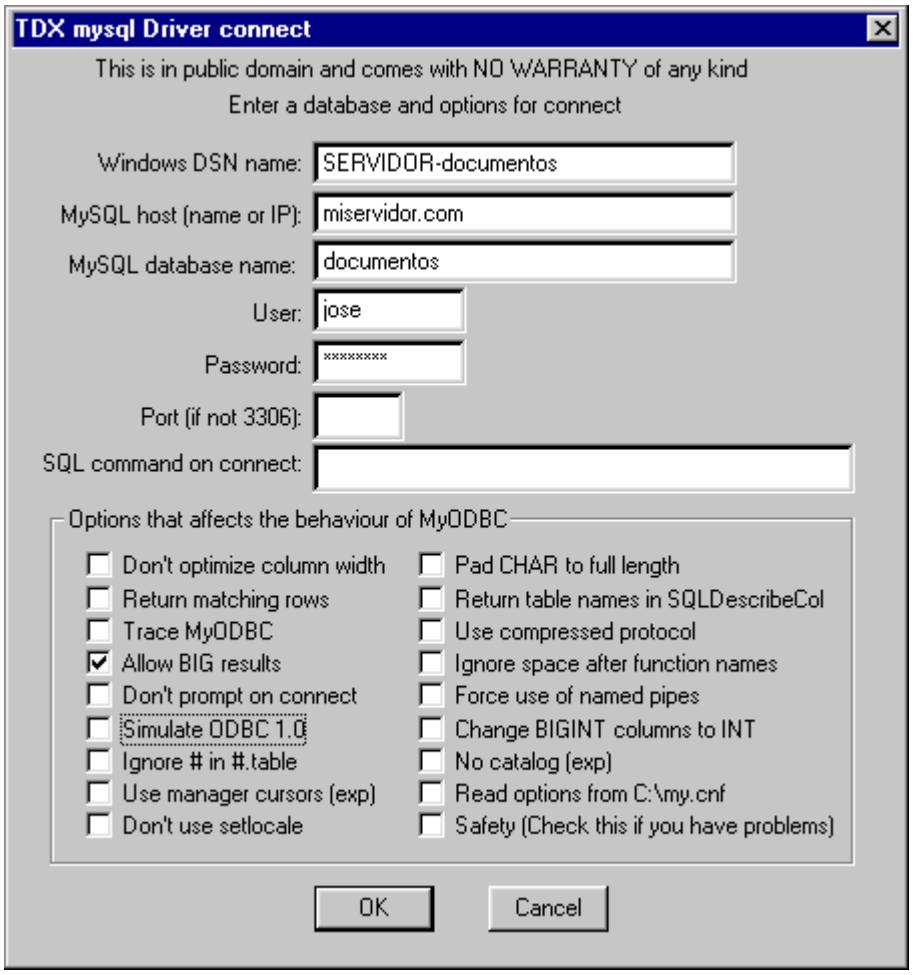
Una vez que el driver ODBC para MySQL ha sido instalado, la siguiente caja de diálogos nos pide configura el origen de la base de datos a conectar. Dentro de la caja de diálogos **Data Source**, elegimos el botón **Add**, para añadir un nuevo origen de datos.



A continuación seleccionamos el driver ODBC correspondiente al de la base de datos a la cuál nos vamos conectar, en nuestro caso MySQL.



Y es este momento cuando nos aparece el cuadro de configuración de nuestra conexión.



Configuración:

- **Windows DNS name**: es el nombre que nosotros le daremos a la conexión.
- **MySQL host (name or IP)**: es el nombre (www.miservidor.com) o dirección IP del servidor al cuál nos queremos conectar. Usar preferentemente la dirección IP.
- **MySQL database name**: nombre de la base de datos MySQL a la que nos queremos conectar.
- **User**: nombre de usuario.
- **Password**: clave del usuario.

- **Port**: puerto que usaremos en la conexión, generalmente es el 3306. Si es otro es necesario especificarlo.

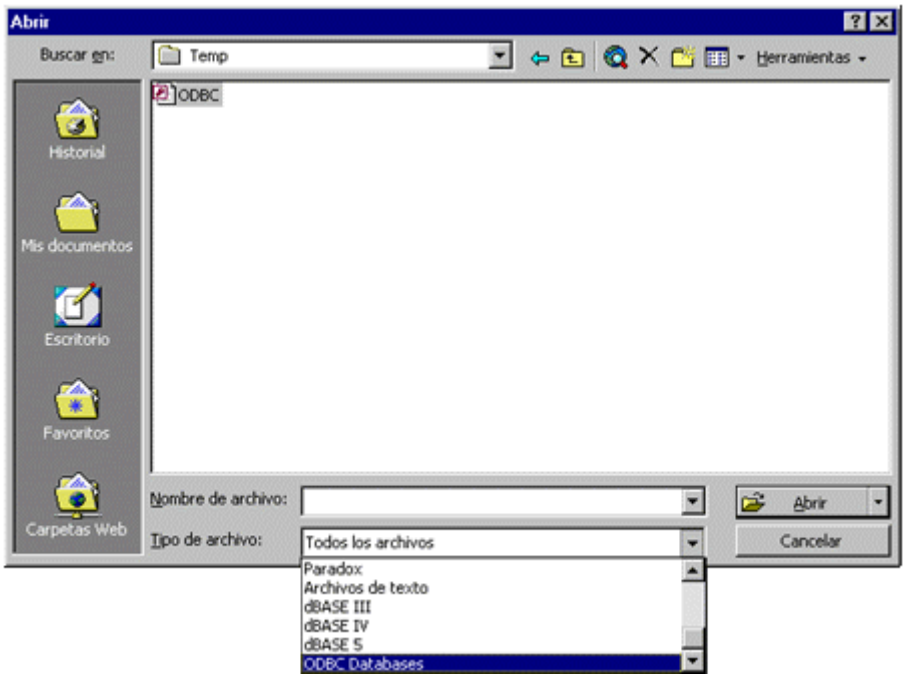
Notas:

- Con MyODBC **no podemos** utilizar los campos Acces de tipo BINARY
- Los parámetrY
- Los parámetros de configuración pueden ser modificados en el menú ODBC del **Panel de Control** de Windows 9x/NT
- Para realizar la conexión a nuestra base de datos MySQL podemos utilizar el programa **MS Access**.
- Es necesario activar la opción "**Allow big results**" para evitar problemas con los campos de tipo TEXT.
- Los usuarios de MS Access 2.0 deben activar la opción "S**imulate ODBC 1.0**"

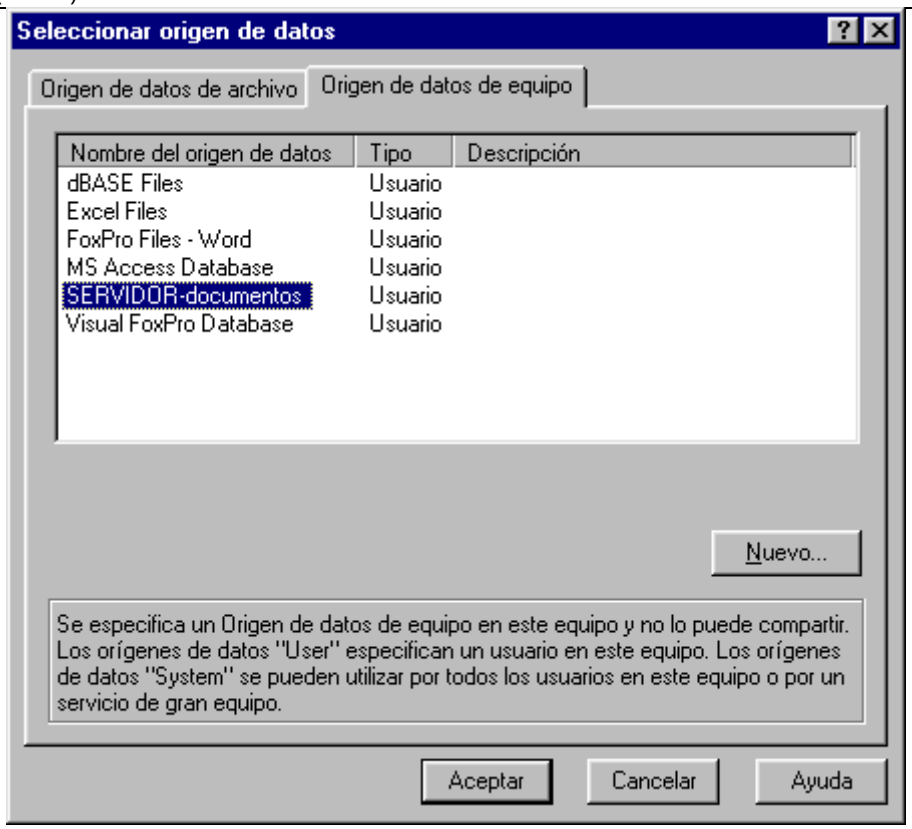
Conexión remota a MySQL con MS Access

Vamos a ver como conectarnos a una base de datos MySQL que está en un servidor remoto mediante un cliente Windows con MS Access. Para ello deberemos tener instalado en nuestro cliente Windows el driver MyODBC.

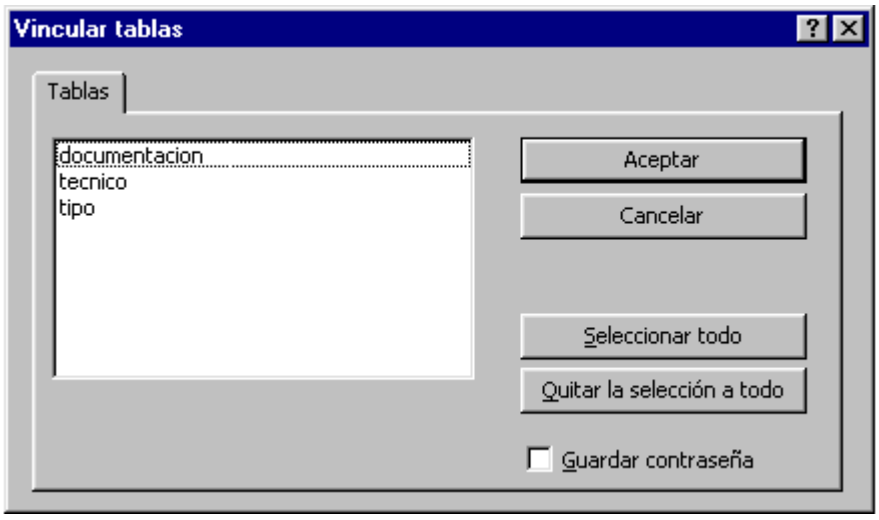
Lo primero, es lo primero, arrancar MS Access, después en el menú "**Archivo**", pinchamos en "**Abrir**", con lo que se nos abre la, pinchamos en "**Abrir**", con lo que se nos abre la caja de diálogos "Abrir" y en campo "**Tipo de archivo**" seleccionamos "**ODBC Databases**".



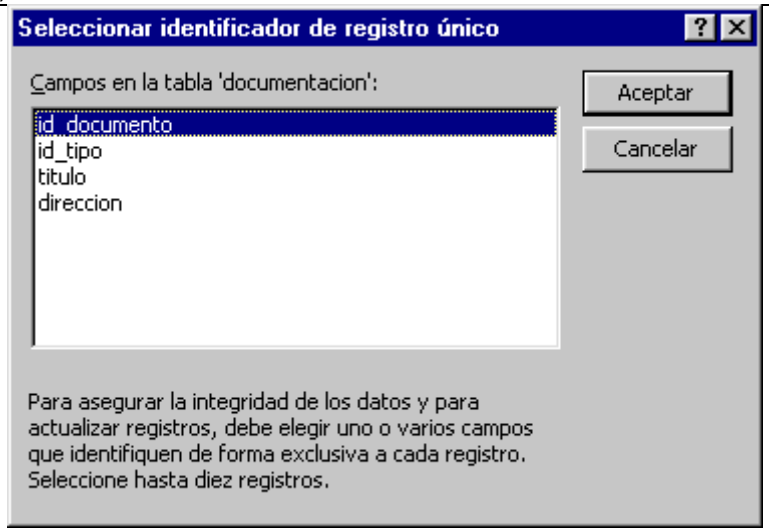
Ahora debemos seleccionar el origen de los datos (la conexión) al que nos vamos a conectar, pinchamos en la lengüeta de "**Origen de datos de equipo**" y escogemos el "**Windows DNS name**" que bien creamos en la instalación de MyODBC o desde el Panel de Windows en "**Control en Fuente de datos ODBC**".



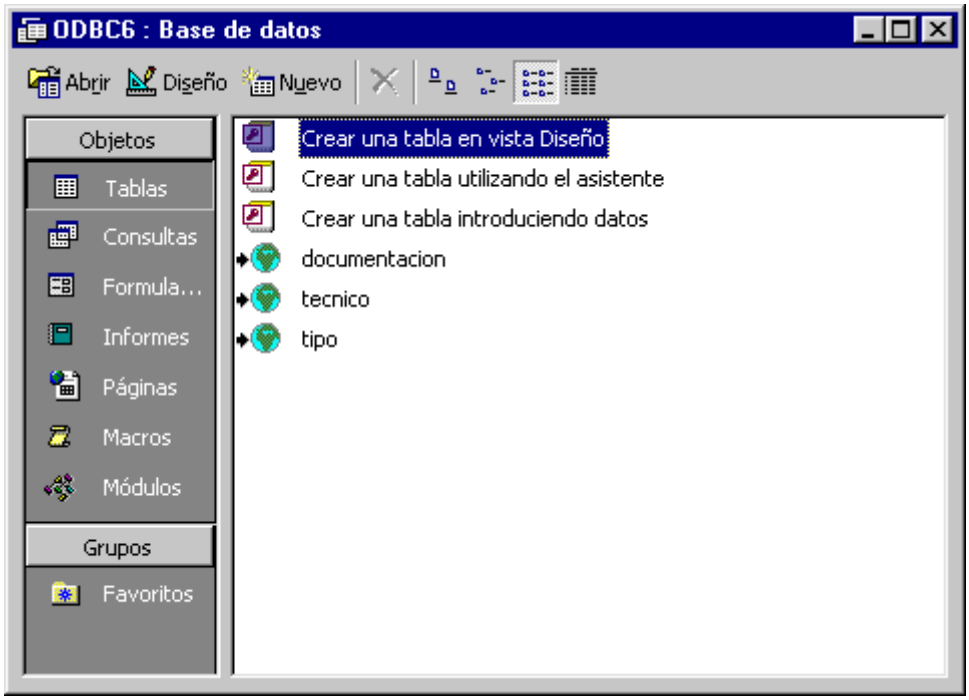
Una vez realizada la conexión a nuestra base de datos remota MySQL, debemos escoger las tablas a vincular.



Si las tablas no tienen una clave primaria, MS Access nos pedirá que elijamos una, sino escogemos ninguna no podremos realizar modificaciones sobre la tabla.



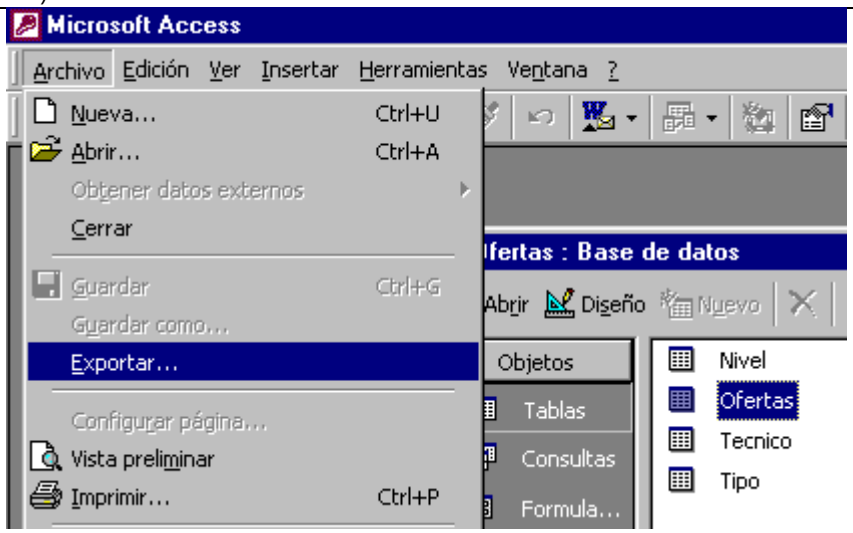
Pues bien ya tenemos nuestra conexión establecida. Cabenemos nuestra conexión establecida. Cabe que recordar que para poder modificar las tablas hay que tener los permisos pertinentes como usuario de MySQL.



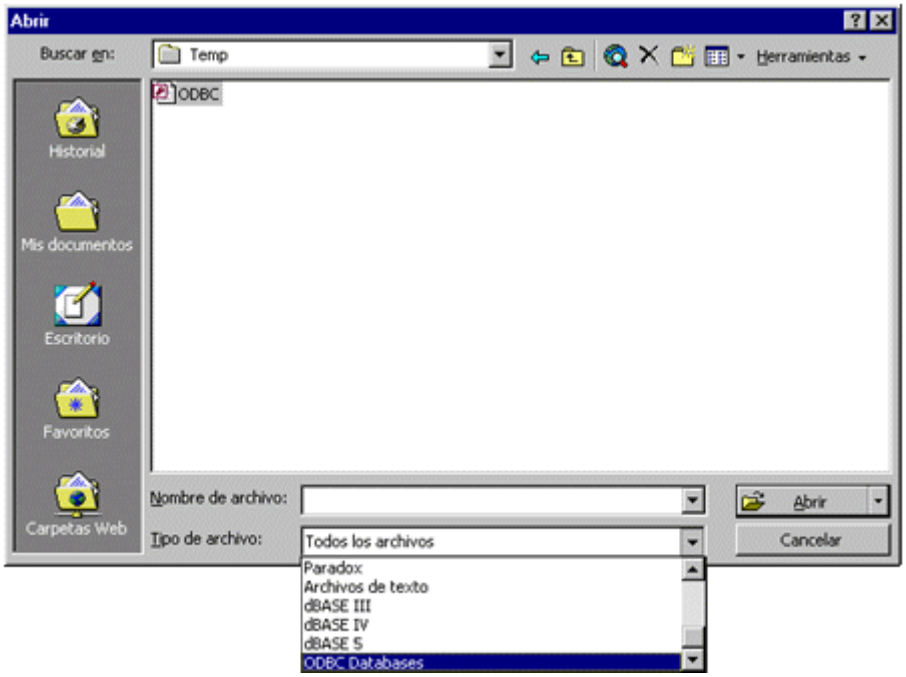
Exportar un tabla MS Access a MySQL

Veamos como exportar una tabla de una base de datos MS Access a otra base de datos remota MySQL mediante ODBC.

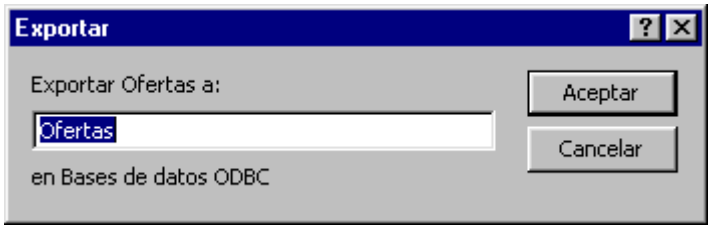
Una vez abierta la base de datos origen con MS Access, seleccionamos la tabla y en el menú **Archivo** pinchamos en **Exportar**.



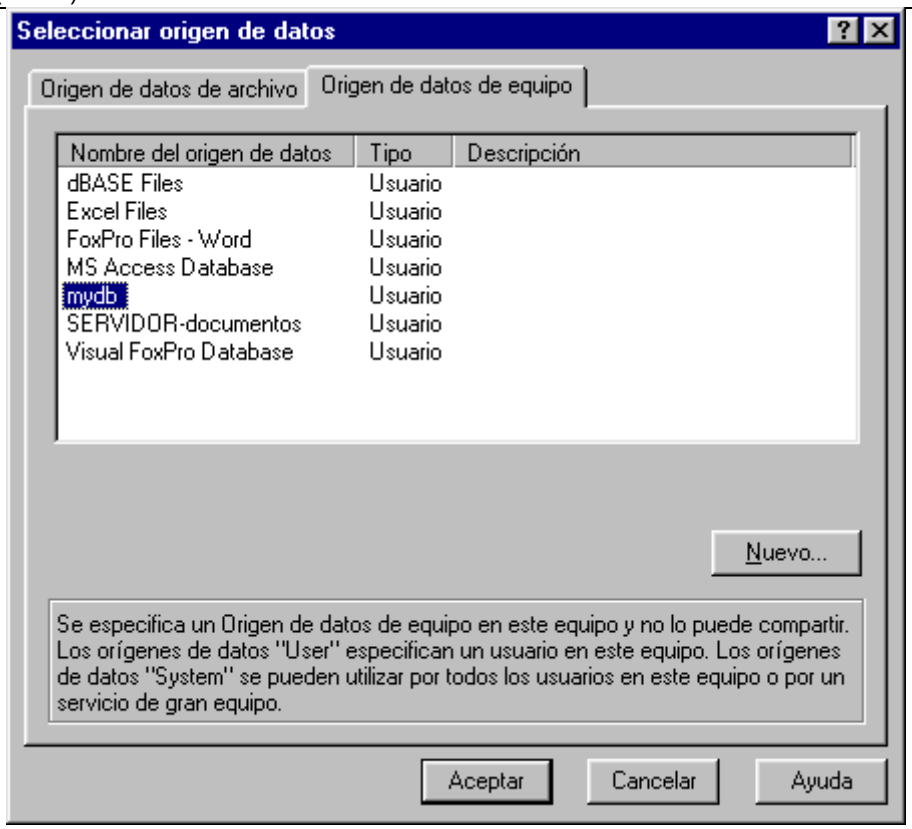
En el siguiente cuadro de diálogo en el campo **Tipo de archivo** seleccionamos **ODBC DataBases**.



Asignamos el nombre que va tomar la tabla exportada en la base de datos MySQL.



Seleccionamos la conexión ODBC a la base de datos MySQL.

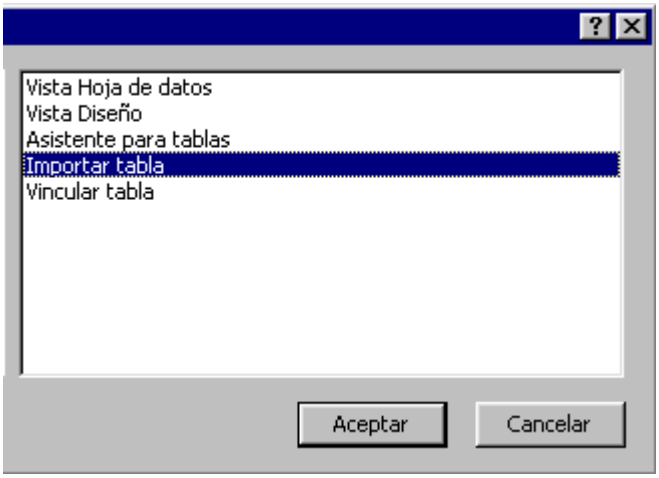


Y ya está todo.

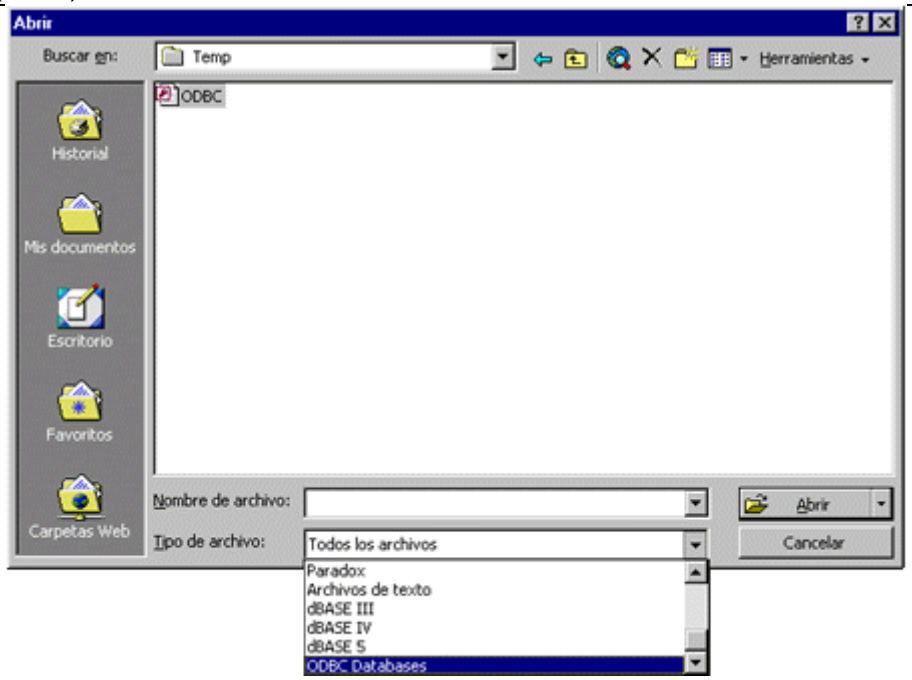
Importar tablas desde MySQL a MS Access

Veamos como hacer para importa una tabla desde una base de datos MySQL en un servidor remoto vía ODBC a una base de datos MS Access.

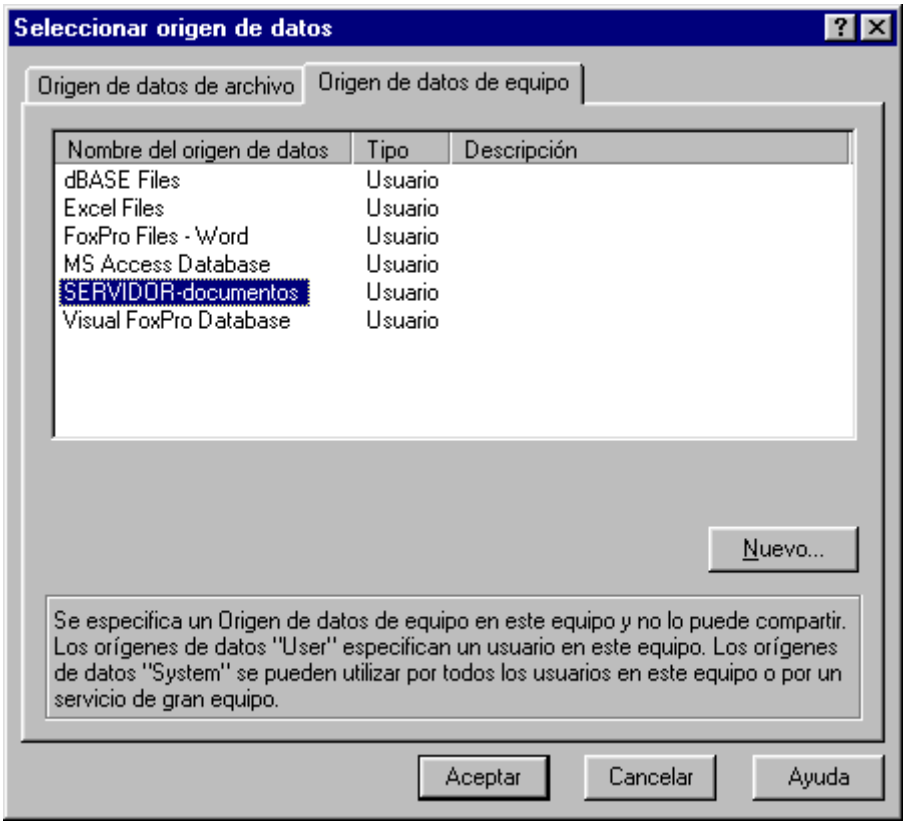
Lo primimero que debemos hacer es abrir con MS Access la base de datos a la cúal queremos importar la tabla. Pichamos el botón de **Nuevo** y en cuadro de dialogo que se nos abre, seleccionamos "**Importar tabla**".



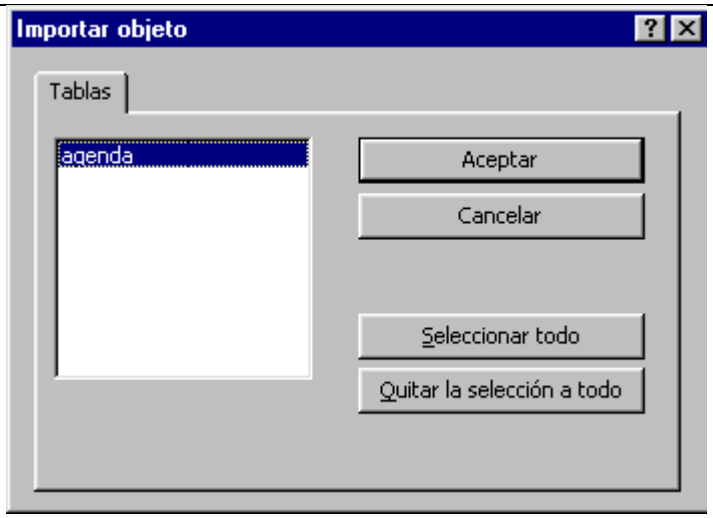
En el siguiente cuadro de diálogo en el campo **Tipo de archivo** seleccionamos **ODBC DataBases**.



Seleccionamos nuestra **conexión ODBC**.



Ya solo nos resta escoger las tablas a importar.


Importar objeto
Tablas
agenda
Aceptar
Cancelar
Seleccionar todo
Quitar la selección a todo