Conexión con el servidor de bases de datos

Antes de empezar a trabajar con bases de datos es **imprescindible** que ambos servidores –Apache y MySQL– estén **activos**.

Como paso inmediato hemos de **interconexionar** *ambos servidores* de forma que sea posible tranferir información de uno a otro.

Para ello es necesario utilizar siempre una función PHP con la siguiente sintaxis:

$c=mysql_connect(*h, u, p***)**

donde **$c** es la variable que recoge el **identificador** del enlace, **h** es la dirección del servidor de bases de datos, (**"localhost"**) , **u** es el nombre de uno de los usuarios registrados en la tabla **user** (**"pepe"** o **"root"**) y **p** la contraseña (en nuestro caso **"pepa"** ó **""**) .

Estos tres valores –cadenas de texto– deben ir escritos entre comillas.

Para **cerrar** la conexión, tenemos que insertar:

$c=mysql_close (***$c*)**

donde **$c** es el nombre de la variable en la que se recogió el **indentificador** del enlace en el momento de la apertura.

Aquí tienes el código de un *script* que realiza la apertura de una conexión y después la cierra.

Y aquí puedes comprobar el funcionamiento del *script* anterior.

Si realizáramos una segunda conexión (con los mismos argumentos) sin haber cerrado la anterior **no se efectuará un nuevo enlace** sino que nos devolverá el ya abierto.

Sintaxis alternativa

Otra forma de efectuar la conexión es utilizar los valores registrados en el fichero **mysql.inc.php** –lo hemos creado en la página anterior– y eso requiere que insertemos un **include("c:...")**, indicando la ruta completa de **seguridad** y el nombre del fichero en el que hemos guardado las claves y que era **mysql.inc.php**.

Automatizar la conexión

Con nuestros conocimientos sobre PHP ya estamos en condiciones de hacer más cómoda la conexión. Creemos una función que realice de forma automática la conexión con MySQL y guardémosla en nuestro fichero **mysql.inc.php**

```
<?
# estas son las variables anteriores
$mysql_server="localhost";
$mysql_login="pepe";
$mysql_pass="pepa";

# creemos una nueva variable $c sin asignarle ningún valor
# para que pueda recoger el identificador de conexión
# una vez que se haya establecido esta

$c;
# escribamos la función que hace la conexión
# como pretendemos que el valor del identificador
# sea usado fuera de la función, para recuperar su valor
# pasaremos ese valor por referencia anteponiendo & al
# nombre de la variable
function conecta1(&$c){
# para usar las variables anteriores en la funcion
# hemos de definirlas como globales
      global $mysql_server, $mysql_login, $mysql_pass;
     if($c=mysql_connect($mysql_server,$mysql_login,$mysql_pass)){
          print "<br>Conexión establecida<br>";
       }else{
         print "<br>No ha podido realizarse la conexión<br>";
      # el exit lo incluimos para que deje de ejecutarse
      # el script si no se establece la conexión
          exit();
      }
}

# esta función asignará a $c el valor del identificador

# repetimos la misma función con otro nombre
# ahora quitaremos el mensaje de conexión establecida
# consideraremos que si no hay mensaje se ha establecido
# asi quedará limpia nuestra página

function conecta2(&$c){
global $mysql_server, $mysql_login, $mysql_pass;
     if($c=mysql_connect($mysql_server,$mysql_login,$mysql_pass)){
       }else{
         print "<br>No ha podido realizarse la conexión<br>";
          exit();
      }
}
?>
```

Si sustituyes el contenido de tu **mysql.inc.php** por el que tienes aquí arriba –puedes eliminar la líneas de comentario al hacerlo– estaremos en disposición de ejecutar scripts como este.

En este ejemplo utilizaremos la primera función:

Ver código fuente	Ejecutar ejemplo

y ahora haremos uso de la segunda

Ver código fuente	Ejecutar ejemplo

En este supuesto como valores de los parámetros **h**, **u** y **p** pondremos los nombres de las variables:
$mysql_server
$mysql_login y
$mysql_pass
sin encerrarlas entre comillas.

Aquí tienes el código de un *script* que realiza la apertura de una conexión y después la cierra.

Desde este enlace puedes comprobar su funcionamiento.

La instrucción OR DIE

Es esta una buena ocasión para hablar de una instrucción PHP que no hemos mencionado hasta el momento.

Es una opción alternativa a **exit()** que, como acabamos de ver en un ejemplo, interrumpe la ejecución de un script en el momento de ser ejecutada.

Cuando se produce un error en la ejecución de un script –no poder establecer conexión con MySQL, por ejemplo– no tiene sentido seguir ejecutándolo. Lo razonable será *interrumpir* el proceso y advertir del error.

Si añadimos a la instrucción **$c=mysql_conect(***'h','u','p'***)** (sin paréntesis, ni comas, ni punto y coma, sólo separado por un espacio):

or die ('*mensaje***')**

y ponemos el *punto y coma* de fin de instrucción después de cerrar este último paréntesis, en el caso de que se produzca un error se interrumpirá la ejecución del script y apare- cerá en la ventana del navegador el texto incluido en *mensaje*.

Este es el código fuente de un script que produce un error –la contraseña es incorrecta– y que utiliza esta *nueva sintaxis*. Pero si lo ejecutas verás que aparece un mensaje de error generado por PHP.

Este tipo de mensajes pueden deshabilitarse haciendo una modificación en **php.ini**. Pero hay una técnica mucho más fácil. Bastará con insertar delante de la función *una arroba* (**@**) para evitar que aparezcan. En este otro script lo hemos incorporado y puedes comprobarlo aqui.

Lista de bases de datos existentes

Antes de **crear** y/o **borrar** una *base de datos* puede ser conveniente y útil comprobar si ya existe.

Tipos de campos en MySQL

MySQL tiene habilitados diversos tipos de **campos** que en una primera aproximación podrían clasificarse en tres grupos:
Campos numéricos
Campos de fecha
Campos de cadenas de caracteres

Campos numéricos

MySQL soporta los tipos numéricos **exactos**(INTEGER, NUMERIC, DECIMAL, y SMALLINT) y los tipos numéricos **aproximados** (FLOAT, DOUBLE precision y REAL).

Los campos que contienen **números enteros** admiten el parámetro **UNSIGNED**, que implica que **no admita signos**, por lo que solo aceptaría **enteros positivos**.

Todos los campos **numéricos** admiten el parámetro **ZEROFILL** cuya función es completar el campo **con ceros a la izquierda** hasta su longitud máxima.

Tipos de campos numéricos enteros

Estos son los distintos tipos de campos numéricos enteros que admite MySQL. Los parámetros señalados entre corchetes son opcionales.

TINYINT *[(M)] [UNSIGNED] [ZEROFILL]*

Número entero *muy pequeño*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **255**. En caso contrario, puede estar comprendido entre **-128** y **127**.

El parámetro **ZEROFILL** sólo tiene sentido junto con la opción **UNSIGNED** ya que *no es habitual* rellenar los números negativos con ceros a la izquierda del signo.

El valor por defecto de parámetro **M** (número de cifras) es **4** si no está activada la opción **UNSIGNED**. Si esta opción estuviera activada el valor por defecto sería **M=3**. Para valores de **M >valor por defecto** reajusta el tamaño al **valor por defecto**.

Si se asigna a M un valor **menor que cuatro** limita el número de caracteres al tamaño especificado considerando el signo sólo en los números negativos.

Por ejemplo, si **M=3** admitiría **148**, pero si intentamos insertar **-148** recortaría por la izquierda y solo insertaría **-14**.

Si intentamos insertar un valor **fuera de rango** registraría **el valor dentro del rango más próximo a él**.
P. ej.: Si tratamos de insertar el valor **437** escribiría **127** ó **255**, este último en el caso de tener la opción **UNSIGNED**.
Si pretendiéramos insertar **-837** con la opción **UNSIGNED** escribiría **0** y sin ella pondría **-128**.

El tamaño de un campo TINYINT es de *1 byte*.

SMALLINT *[(M)] [UNSIGNED] [ZEROFILL]*

Número entero *pequeño*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **65 535**. En caso contrario, puede estar comprendido entre **-32 768** y **32 767**.

Son válidos los comentarios hechos para **TINYINT**, excepto los relativos a los valores **por defecto** de **M** que en este caso serían **6** ó **5**. Su tamaño es de *2 bytes*.

MEDIUMINT *[(M)] [UNSIGNED] [ZEROFILL]*

Número entero *mediano*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **16 777 215**. En caso contrario, puede estar comprendido entre **-8 388 608** y **8 388 607**.

También son válidos los comentarios hechos para **TINYINT**, excepto los relativos al valor **por defecto** de **M** que en este caso serían **8**. Su tamaño es de *3 bytes*.

INT *[(M)] [UNSIGNED] [ZEROFILL]*

Número entero. Con la opción **UNSIGNED** puede tomar valores entre **0** y **4 294 967 295**. En caso contrario, puede estar comprendido entre **-2 147 483 648** y **2 147 483 647**.

Son válidos todos los comentarios de los casos anteriores. Su tamaño es de *4 bytes*.

INTEGER *[(M)] [UNSIGNED] [ZEROFILL]*

PHP dispone de herramientas para conocer el **número** de bases de datos existentes en el servidor, así como sus nombres. Todas ellas requieren que se haya establecido una conexión con el servidor.

$p=mysql_list_dbs(*$c*)

La variable **$p** es un **nuevo identificador** *imprescindible* y *previo* a la determinación del número y los nombres de las bases de datos existentes en el enlace abierto (identificado por **$c**).

$n=mysql_num_rows(*$p*)

Esta función devuelve el *número de bases de datos existentes* en el servidor.

Utiliza como parámetro (**$p**) el resultado obtenido mediante la función anterior.

Ese número puede recogerse en una variable (en este caso **$n**).

mysql_db_name(*$p, i*)

Esta nueva función devuelve el nombre de una de las bases de datos, identificada por un número **i** que debe pertenecer al intervalo **[0,$n)**.

Fíjate que **i** tiene que ser **i<$n** porque si, por ejemplo, **$n=5** los cinco valores posibles de **i** serían: 0,1,2,3 y 4.

Una **lista completa** de todas las bases de datos existentes en el servidor podría hacerse mediante el siguiente proceso:

• *Abrir* la conexión.
• *Invocar* a mysql_list_dbs.
• *Contar* el número de bases de datos con *mysql_num_rows*
• Insertar un bucle:
for ($i=0;$i<$num,$i++)
• Presentar la *lista de nombres* mediante un bucle que lea los diferentes valores de **$i** en:
mysql_db_name($p,$i)

Aquí tienes el código fuente de un ejemplo completo y desde aquí puedes ejecutarlo

Crear una base de datos

La creación de **una base de datos** también requiere una conexión previa y utiliza la siguiente sintaxis:

mysql_query
("CREATE DATABASE *nom*")

donde **nom** es el nombre de la **nueva base de datos**.

Esta función devuelve TRUE si la base de datos es creada, y FALSE si no es posible hacerlo.

Es un sinónimo de **INT**

BIGINT *[(M)] [UNSIGNED] [ZEROFILL]*

Número entero *grande*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **18 446 744 073 709 551 615**. En caso contrario, puede estar comprendido entre **-9 223 372 036 854 775 808** y **21 474 839 223 372 036 854 775 807 647**, pero al usarlo desde PHP estará sujeto a las limitaciones máximas de los valores numéricos de este.

Son válidos todos los comentarios de los casos anteriores. Su tamaño es de *8 bytes*.

Números de coma flotante

Por la estructura binaria de los microprocesadores y habida cuenta de que algunos números **no enteros** -sin ir más lejos, el **0.1**- requerirían *infinitos caracteres binarios* para su representación exacta, se hace necesario introducir un **redondeo** en su tratamiento informático y como consecuencia de ello asumir que se generan **errores de medida**.

Esta circunstancia obligó al tratamiento de los números decimales mediante el llamado **Standar de Aritmética de Punto Flotante**, un algoritmo definido por la **IEEE** (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers) que unificó los procesos de representación de números en ordenadores con lo que son uniformemente controlables los errores introducidos.

El **Standar de Aritmética de Punto Flotante** estableció **dos niveles de precisión**:
Precisión **Simple**, en la que **todo número debe ser almacenado en 32 bits**
(4 bytes)
Doble precisión, en la que los **números se almacenan en 64 bits** (8 bytes).
MySQL admite los siguientes tipos de números de coma flotante:

FLOAT(x) *[ZEROFILL]*

Número de **coma flotante**. Ignora la opción **UNSIGNED**, pero sí acepta **ZEROFILL**, por lo que debe prestarse atención a estas opciones ya que no sería demasiado habitual una presentación como esta: **000-3.47**

El valor de **x** especifica la *precisión*. Si **x<=24** será de **precisión simple**. cuando **24 <x <=53** lo convertirá automáticamente a **doble precisión**.

Cuando no se especifica el valor de **x** considera el campo como de **precisión simple**. Su tamaño es de *4 bytes* si **x<=24** y de *8 bytes* cuando **24 <x <=53**

FLOAT *[(M,D)] [ZEROFILL]*

Número de **coma flotante** de **precisión simple**. Son válidos los comentarios relativos a las opciones **UNSIGNED** y **ZEROFILL** del caso anterior.

Toma valores en los intervalos siguientes:
-3.402823466E+38 a **-1.175494351E-38**
0 y
1.175494351E-38 a **3.402823466E+38**.
M es la **anchura máxima de visualización** y **D** es el número de **decimales**. Si **M > 24** se convierte automaticamente a **doble precisión**
FLOAT sin argumentos representa un número de **coma flotante** y **precisión simple**.

DOUBLE *[(M,D)] [ZEROFILL]*

Número de **coma flotante** de **doble precisión**. Siguen siendo válidos los comentarios relativos a las opciones **UNSIGNED** y **ZEROFILL** del caso anterior.

Toma valores en los intervalos siguientes:
-1.7976931348623157E+308 a **-2.2250738585072014E-308**
0 y
2.2250738585072014E-308 a **1.7976931348623157E+308**
M es la **anchura máxima de visualización** y **D** es el número de **decimales**.

DOUBLE sin argumentos representa un número de **coma flotante** y **precisión doble**.

REAL *[(M,D)] [ZEROFILL]*

Es sinónimo de **DOUBLE**.

DECIMAL *[(M[,D])] [ZEROFILL]*

Si intentamos crear una base de datos **con un nombre ya existente** la función nos devolverá FALSE.

Aquí tienes el código de un ejemplo de creación de una base de datos. Si lo ejecutas **dos veces** podrás comprobar que en la segunda oportunidad te aparece el mensaje diciendo que no ha sido posible crearla.

Borrar una base de datos

Para borrar una base de datos se requiere el uso de la siguiente función PHP:

mysql_query
("DROP DATABASE *nom***")**

donde **nom** es el nombre de la base de datos y debiendo ponerse toda la cadena del paréntesis entre comillas.

Esta función devuelve TRUE cuando se ejecuta con éxito, y FALSE en el caso contrario.

Este es el código de un *script* que puede borrar la base creada anteriormente.

Igual que ocurría al tratar de crearla, si intentamos borrar una base de datos inexistente la función **mysql_drop_db** nos devolverá FALSE.

Depurando los procesos de creación y borrado de bases de datos

Cuando intentamos crear una base de datos ya existente o borrar una inexistente las funciones **mysql_query** nos devuelven FALSE pero esa respuesta no nos *dice* la causa por la que *no ha sido posible* la ejecución de la instrucción.

Sería mucho más interesante **comprobar** la **existencia** o **inexistencia** de una base de datos antes de ejecutar esas instrucciones y que después de la comprobación se nos presentara un mensaje informativo.

MySQL dispone de una sentencia para este fin, pero –aunque la vamos ver más adelante– olvidémosnos de su existencia e intentemos crear nuestro propio script de comprobación.

Combinando las instrucciones anteriores no resulta difícil hacerlo. Aquí tienes un ejemplo de código para efectuar esa comprobación al crear una base de datos y este otro código es para el caso de borrado.

Experimenta con estos scripts, sustitúyelos por otros propios en los

Es un número de **coma flotante y doble precisión** que se almacena como un campo de tipo **CHAR**.

El valor es guardado como una cadena donde cada carácter representa una cifra. La *coma* y el *signo menos de los números negativos* no son tenidos en cuenta en el valor de **M** -anchura máxima de visualización- aunque si se reserva -automáticamente- espacio para ellos en campo.
Si **D** vale **0** no tendrá parte decimal. Los números toman valores en el mismo intervalo especificado para **DOUBLE**.

Los valores por defecto de **M** y **D** son respectivamente **10** y **0**.
Ocupan *M+2 bytes* si *D > 0*; *M+1 bytes* si *D = 0* ó *D+2 bytes* si *M < D*

NUMERIC(M,D) *[ZEROFILL]*

Se comporta de forma idéntica a **DECIMAL**

Campos de fecha

MySQL dispone de campos específicos para el almacenamiento de fechas. Son los siguientes:

DATE

Recoge una *fecha* dentro del intervalo **01-01-1000** a **31-12-9999**. MySQL guarda los valores DATE con formato **AAAA-MM-DD** (año-mes-día) . Su tamaño es de *3 bytes*.

DATETIME

Recoge una combinación de *fecha* y *hora* dentro del intervalo **00:00:00 del día 01-01-1000** y las**23:59:59 del día 31-12-9999**. MySQL guarda los valores DATETIME con formato **AAAA-MM-DD HH:MM:SS** (año-mes-día hora:minutos:segundos) . Su tamaño es de *8 bytes*.

TIME

Recoge una *hora* dentro del intervalo **-838:59:59** a **838:59:59**. MySQL guarda los valores TIME con formato **HH:MM:SS** (horas:minutos:segundos) . Su tamaño es de *3 bytes*.

YEAR 0 YEAR(2) o YEAR(4)

Recoge un *año* en formato de *cuatro cifras* (**YEAR** o **YEAR(4)**) o en formato de *dos cifras* (**YEAR(2)**)dentro del intervalo **1901** a **2155** en el caso de cuatro cifras o de **1970** a **2069** si se trata de dos cifras. Su tamaño es de *1 byte*.

TIMESTAMP *[(M)]*

Recoge un *tiempo UNIX*. El intervalo válido va desde **01-01-1970 00:00:00** a cualquier fecha del año **2037**.

El parámetro **M** puede tomar los valores: **14** (valor por defecto), **12**, **8**, o **6** que se corresponden con los formatos **AAAAMMDDHHMMSS**, **AAMMDDHHMMSS**, **AAAAMMDD**, o **AAMMDD**.
Si se le asigna la opción **NUL** guardará la hora actual. Cuando se asigna **8** o **14** como parámetros es considerado como **un número** y para las demás opciones como **una cadena**.

Independientemente del valor del parámetro, un campo TIMESTAMP siempre ocupa *4 bytes*.

Campos tipo *cadena de caracteres*

CHAR (M) *[BINARY]*

Es una **cadena de tamaño fijo** que se **completa a la derecha por espacios** si es necesario.

El parámetro M puede valer de **1** a **255** caracteres.

Los espacios finales son suprimidos cuando la cadena es insertada en el registro.

Los valores de tipo CHAR son elegidos y comparados **sin tener en cuenta Mayúsculas / Minúsculas** y utilizan el juego de carácteres *por defecto*.

Se puede utilizar el operador **BINARY** para hacer la cadena **sensible a Mayúsculas / Minúsculas**.

Se puede utilizar un campo tipo **CHAR(0)** con el atributo **NULL** para almacenar una valor booleano. En este caso ocupará un solo byte y podrá tener **únicamente dos valores**: **NUL** ó **""**.

Su tamaño es de *M bytes* siendo 1 <= M <= 255 .

VARCHAR(M) *[BINARY]*

que utilices las funciones que hemos incluido –a la derecha tienes el código fuente– dentro del fichero **mysql.inc.php** y comprueba, listando las bases de datos existentes, que sólo queden: **mysql** y **test**.

Tipos de campos MySQL

Conocidos los procesos de creación, listado y borrado de bases de datos ya estamos en disposición en empezar a tratar lo relativo a las **tablas**.

Es muy necesario conocer los *diferentes tipos* de **campos** que pueden contener las **tablas** de MySQL. Aquí a la derecha los tienes.

Conocer las posibilidades de cada uno será fundamental a la hora de diseñar una tabla. En ese momento tendremos que decidir qué campos son necesarios, cuál es tipo requerido, cuáles han de ser sus dimensiones y también cuáles de ellos requerirán ser tratados como índices.

Tipos de campo bien elegidos y un tamaño adecuado a las necesidades reales de nuestro proyecto son las mejores garantías para optimizar el tamaño de la tabla y para hacerla realmente eficaz.

El *grado de eficiencia* de una base de datos suele ser **directamente** proporcional al *tiempo invertido* en el análisis de la estructura de sus tablas.

CREACIÓN DE LAS BASES DE DATOS NECESARIAS PARA EL CURSO

Una vez hayas hecho todas las pruebas necesarias con las funciones anteriores, llega el momento de utilizarlas para crear las bases de datos que vamos a utilizar en el Curso.

Pulsa en este enlace para que cree automáticamente la base de datos que va a contener los sucesivos ejemplos que hemos incluido en estos materiales.

Crear base de datos EJEMPLOS

Una vez hecho esto, escribe tu propio script y crea una segunda base de datos como el nombre **practicas**. Esta será la que habrás de utilizar en ejercicios que tendrás que ir haciendo en el resto del curso.

Es una **cadena de caracteres de longitud variable**. Su tamaño máximo -especificado en el parámetro **M**- puede estar comprendido entre **1** y **255** caracteres. Con la opción **BINARY** es capaz de discriminar entre Mayúsculas / minúsculas.

TINYBLOB o TINYTEXT

TINYBLOB y **TINYTEXT** son **cadenas de caracteres de longitud variable** con un tamaño máximo de 255 (2^8 - 1) caracteres.

La diferencia entre ambas es que **TINYBLOB discrimina** entre Mayúsculas / minúsculas, mientras que **TINYTEXT** no lo hace.

Ninguno de los campos **BLOB** y **TEXT** admite **valores por DEFECTO**

Las versiones de **MySQL** anteriores a **3.23.2** permiten utilizar estos campos para indexar.

Si se intenta guardar en un campo de este tipo una cadena **de mayor longitud que la especificada** solamente se guardarán los **M** primeros caracteres de la cadena.

BLOB o TEXT

BLOB y **TEXT** son **cadenas de caracteres de longitud variable** con un tamaño máximo de 65535 (2^{16} - 1) caracteres.

La diferencia entre ambas es que **BLOB** si **discrimina** entre Mayúsculas / minúsculas, mientras que **TEXT** no lo hace.

Ninguno de los campos: **BLOB** y **TEXT** admite **valores por DEFECTO**

MEDIUMBLOB o MEDIUMTEXT

MEDIUMBLOB y **MEDIUMTEXT** son **cadenas de caracteres de longitud variable** con una longitud máxima de 16.777.215 (2^{24} - 1) caracteres.
Son válidas las especificaciones hechas en el apartado anterior.

El tamaño máximo de los campos de este tipo está sujeto a limitaciones externas tales como la memoria disponible y el **tamaño del buffer de comunicación servidor/cliente**.

LONGBLOB o LONGTEXT

Su única diferencia con la anterior es el tamaño máximo de la cadena, que en este caso es 4.294.967.295 (2^{32} - 1) caracteres.

ENUM('valor1','valor2',...)

Es **una cadena de caracteres** que contiene **uno solo** de los valores de la lista (valor1, valor2, etc. etc.).

A la hora de insertar un nuevo registro en una tabla, el valor a especificar para un campo de este tipo ha de ser **una cadena** que contenga **uno de los valores** especificados en la tabla. Si se tratara de insertar un valor distinto de ellos insertaría una cadena vacía.

SET('valor1','valor2','valor3'...)

Es una cadena de caracteres formados por la **unión** de ninguno, uno o varios de los valores de una lista. El máximo de elementos es 64.

Los valores que **deben escribirse** en los **campos** de este tipo han de ser **numéricos**, expresados en forma **binaria** o en forma decimal.

En el supuesto de que contuviera **tres valores** los posibles valores a **insertar** en un campo de este tipo a la hora de añadir un registro serían los siguientes.

Incluir valores			*Código valor*			*Cadena binaria*	*Equiv. decimal*
val1	*val2*	*val3*	*val1*	*val2*	*val3*		
Si	Sí	Sí	1	1	1	111	**7**
Si	Sí	No	1	1	0	011	**3**
Si	No	Sí	1	0	1	101	**5**
No	Sí	Sí	0	1	1	110	**6**
No	No	Sí	0	0	1	100	**4**
No	Sí	NO	0	1	0	010	**2**
Si	No	No	1	0	0	001	**1**
No	No	No	0	0	0	000	**0**

¡Cuidado!

Antes de continuar, es conveniente comprobar desde Windows que el subdirectorio **mysql/data** contiene las bases de datos **ejemplos** y **practicas**