Sintaxis MySQL de selección de registros

Las *sentencias* de selección de registros requieren utilizar -entre otras- *palabras clave* como las que enumeramos a continuación.

Observa que hay dos tipos: *obligatorias* y *opcionales*, y que algunas de las palabras clave son *alternativas* y por lo tanto, incompatibles en una misma sentencia.

El uso de estas palabras clave *requiere* que sean insertadas en un *determinado orden* tal y como se enumera aquí debajo.

Si alteráramos ese orden (p. ejemplo: colocando GROUP BY antes de WHERE) nos daría un **error** y no se ejecutaría la sentencia.

SELECT

Es la **primera palabra** de la sentencia de búsqueda y tiene carácter **obligatorio**.

[STRAIGHT_JOIN]

Es una *palabra clave* de uso **opcional** (la marcamos con corchetes para indicar su condición de opcional) que fuerza al *optimizador MySQL* a organizar las tablas en el mismo orden en el que han sido especificados los campos en la cláusula FORM.

Sirve para mejorar -en casos muy concretos- la velocidad de gestión de tablas de gran tamaño.

[SQL_BIG_RESULT]

Es una *cláusula* **opcional** que se usa para indicar al *optimizador* que el resultado va a tener **una gran cantidad de registros**.

En ese caso, MySQL utilizará *tablas temporales* cuando sea necesario para optimizar la velocidad de gestión de la información.

Esta *cláusula* también puede ser utilizada dentro de GROUP BY.

[SQL_BUFFER_RESULT]

Es **opcional** y su finalidad es la de **forzar** a MySQL a tratar el resultado en un *fichero temporal*.

Ese tratamiento ayuda a MySQL a *liberar recursos más rápidamente* y es de gran utilidad (siempre desde el punto de vista de la rapidez) cuando es necesario un largo proceso de cálculo antes de enviar los resultados al *cliente*.

[HIGH_PRIORITY]

Esta cláusula, **opcional** da prioridad al comando SELECT sobre otros comandos que simultáneamente pudieran estar intentando acceder a la tabla para *escribir* en ella (añadir o modificar registros).

Si esta opción está activa, los intentos de escritura que pudieran producirse de forma simultánea deberían esperar al final de este proceso para ejecutarse.

campo1, campo2, ...

Tienen carácter **obligatorio** y

Consultar los registros de una tabla

Las consultas de los datos y registros contenidos en una tabla ofrecen un amplísimo abanico de posibilidades a partir de las opciones que tienes descritas al margen. Veamos algunas de las posibilidades.

La consulta más simple

Si utilizamos la sentencia

SELECT * FROM *tabla*

obtendremos información sobre
todos los campos
(*) y la salida estará en el mismo orden en el que fueron añadidos los datos. Si visualizas este ejemplo, verás que aparecen ordenados por el valor
autonumérico
del campo
Contador
lo cual, como ves, resulta coherente con la afirmación anterior.

Ver código fuente	Ejecutar la consulta

Consultando sólo *algunos campos*

Ahora utilizaremos la sentencia

SELECT *campo1*,*campo2*, ... FROM *tabla*

y tendremos como resultado una lista completa, por el mismo orden que la anterior, pero sólo
mostrando
los campos indicados.

Ver código fuente	Ejecutar la consulta

¡Cuidado!

En los comentarios contenidos en estos ejemplos puedes ver la forma en la que **mysql_fetch_row** y **mysql_fetch_array** tratan los índices escalares de los resultados que producen los SELECT de MySQL.
Los valores de los índices se asignan a los contenidos de los campos por el mismo orden en el que estos se escriben en la sentencia SELECT. El *campo1* (primero que se escribe) será recogido por el elemento de índice **cero** del array, el *campo2* será recogido con índice **uno** y así sucesivamente

Consultando sólo *algunos campos* y limitando la salida a *n* registros

Ahora utilizaremos la sentencia

SELECT *campo1*,*campo2*, ... FROM *tabla* LIMIT (n, m)

y tendremos como resultado una lista que contendrá
m
registros a partir del
n+1
, por el mismo orden que la anterior, y
mostrando
los campos indicados.

Ver código fuente	Ejecutar la consulta

Consultando sólo *algunos campos* y ordenando la salida

Utilizaremos la sentencia MySQL de esta forma

SELECT *campo1*,*campo2*, ... FROM *tabla* ORDER BY *campo_n* [ASC|DESC], *campo_m* [ASC|DESC]

y tendremos como resultado una lista ordenada por el primero de los campos indicados en
ORDER BY
, y en caso de
coincidencia
de valores en ese campo, utilizaríamos el criterio de ordenación señalado en segundo lugar.

Ver código fuente	Ejecutar la consulta

Consulta *seleccionando registros*

señalan los campos de la tabla que deben incluirse en la consulta.

La función SELECT sólo devolverá información de aquellos campos que estén enumerados aquí.

Si se desea que la consulta **incluya a todos campos** bastará con incluir en esta posición un *****, que es el carácter *comodín* que indica a MySQL que se desea incluir todos los campos en la consulta.

Los **campos numéricos** tienen la opción de llevar asociadas **funciones** MySQL que devuelven información *estadística*.

Algunas de esas funciones son las siguientes:

- **MAX(***campo..***)**
  Devuelve el valor **máximo** de ese campo en todos los registros de la tabla, salvo que tenga la opción GROUP BY, en cuyo caso devolverá el máximo de cada grupo, o cuando tenga activada la opción WHERE, en cuyo caso la función sólo será aplicada a los registros que resulten de tal *filtrado*.
- **MIN(***campo..***)**
  Idéntica a la anterior en cuanto a criterios de selección, esta función devuelve el *mínimo*.
- **AVG(***campo..***)**
  Devuelve el valor *promedio* de todos los registros numéricos seleccionados con los mismos criterios del caso anterior.
- **SUM(***campo..***)**
  Devuelve la **suma** de los valores del **campo** y sigue idénticos criterios de selección de campos que en los casos anteriores.
- **STDDEV(***campo..***)**
  Devuelve la estimación de la *desviación típica* de la población.
- **COUNT(***campo..***)**
  Cuenta los valores *no nulos* del campo indicado.

En el caso de aplicar estas funciones, el **resultado de la consulta** contiene una sola línea, salvo que active la opción GROUP BY, en cuyo caso devolverá **tantas líneas** como **grupos resulten**.

FROM *tabla*

Esta *expresión* -que aunque no tiene carácter obligatorio podría tomarse como tal– indica a MySQL el **nombre de la tabla** en el que debe efectuarse la consulta.

WHERE *definicion*

Esta *instrucción* tiene carácter opcional y su utilidad es la de *filtrar* la consulta estableciendo los criterios de selección de los **registros** que debe devolver.

Si se omite WHERE, la consulta devolverá **todos** los registros de la tabla.

En la parte derecha tienes información sobre la manera de **definir** los criterios de selección de esta opción.

GROUP BY *definicion*

Tiene carácter **opcional** y su finalidad es la de presentar los resultados de la consulta

Utilizaremos la sentencia MySQL de esta forma

SELECT *campo1*, ... FROM *tabla* WHERE *condición*

que nos devolverá la lista de registros que **cumplen la condición indicada**

. Aquí tienes un ejemplo muy sencillo.

Ver código fuente	Ejecutar la consulta

La claúsula **WHERE** permite un variado abanico de **condiciones**, que trataremos de resumir aquí. Algunos de ellas son los siguientes:

Operador	Tipo de campo	Sintaxis	Descripción	Código fuente	Ver ejemplo
=	**Numérico**	WHERE *campo*=num	Selecciona los registros que contienen en el *campo* un valor **igual** a **num**	Ver	Probar
=	**Cadena**	WHERE *campo*="cadena"	Selecciona los registros que contienen en el *campo* una cadena **idéntica** a *cadena* (*)	Ver	Probar
<	**Numérico**	WHERE *campo*<num	Selecciona los registros que contienen en el *campo* un valor **menor** a **num**	Ver	Probar
<	**Cadena**	WHERE *campo*<"cadena"	Selecciona los registros que contienen en el *campo* una cadena **cuyos *n* primeros caracteres** son **menores** que los de la *cadena*, siendo *n* el número de caracteres que contiene *cadena*. (**)	Ver	Probar
< =	**Numérico**	WHERE *campo*<=num	Selecciona los registros que contienen en el *campo* un valor **menor** O **igual** a **num**	Ver	Probar
< =	**Cadena**	WHERE *campo*<="cadena"	Selecciona los registros que contienen en el *campo* una cadena **cuyos *n* primeros caracteres** son **menores** que los de la *cadena*, siendo *n* el número de caracteres que contiene *cadena* y añade respecto al caso anterior la opción de que en caso de que **ambos valores fueran iguales** también los presentaría (**)	Ver	Probar
>	**Numérico**	WHERE *campo*>num	Selecciona los registros que contienen en el *campo* un valor **mayor** a **num**	Ver	Probar
>	**Cadena**	WHERE *campo*>"cadena"	Selecciona los registros que contienen en el *campo* una cadena **cuyos *n* primeros caracteres** son **mayores** que los de la *cadena*, siendo *n* el número de caracteres que contiene *cadena*. (**)	Ver	Probar
> =	**Numérico**	WHERE *campo*>=num	Selecciona los registros que contienen en el *campo* un valor **mayor** o **igual** a **num**	Ver	Probar
> =	**Cadena**	WHERE *campo*>="cadena"	Selecciona los registros que contienen en el *campo* una cadena **cuyos *n* primeros caracteres** son **mayores** que los de la *cadena*, siendo *n* el número de caracteres que contiene *cadena* y añade respecto al caso anterior la opción de que en caso de que **ambos valores fueran iguales** también los presentaría (**)	Ver	Probar
IN	**Numérico o Cadena**	WHERE *campo*IN (valor1, valor2..)	Selecciona los registros que contienen en el *campo* valores que coinciden con alguno de los especificados dentro del paréntesis. Cuando se trata de valores *no numéricoz* han de ir *entre comillas*	Ver	Probar

agrupados según el criterio establecido en su **definición**.

Resulta de gran utilidad cuando se pretende obtener **valores estadísticos** de los registros que **cumplen determinadas condiciones** (las condiciones del *agrupamiento*).

ORDER BY *definicion*

También tiene carácter **opcional** y su utilidad es la de presentar la información de la consulta **ordenada** por los contenidos de **uno** o **varios** campos.

Siempre tiene como opción complementaria de que **en cada campo utilizado para la ordenación** puede establecerse uno de estos criterios **ASC** (ascendente, es el valor por defecto) o **DESC**.

Si no se establece ningún orden, los resultados de la consulta aparecerán en el mismo orden en el que fueron añadidos los registros.

LIMIT *m, n*

Esta cláusula es **opcional** y permite establecer **cuántos** y **cuáles** registros han de presentarse en la salida de la consulta.

Por ejemplo: *LIMIT 4, 8* indicaría a MySQL que la consulta debería *mostrar* **OCHO** registros contados a partir del **quinto** (sí, el quinto porque LIMIT considera el primer registro como CERO).

El criterio límite se aplica sobre los resultados de la salida, es decir, sobre los resultados **seleccionados**, **ordenados** y **filtrados** siguiendo los criterios establecidos por las cláusulas anteriores.

Si se escribe como un solo parámetro (LIMIT k), MySQL lo interpretará como que **k** es el segundo de ellos y que el primero es CERO, es decir:
LIMIT 0, k

Recuento de resultados

PHP dispone de dos funciones que permiten conocer el número de registros de la tabla afectados por una sentencia MySQL.

mysql_num_rows (*$c* **)**

Esta función devuelve un valor numérico que recoge el número de registros que cumplen las condiciones establecidas en una **consulta**. Sólo es válido para sentencia tipo SELECT

mysql_affected_rows(*$c* **)**

En este caso la función devuelve también el número de registros afectados, pero sólo en el caso de que la sentencia MySQL haya producido modifi- caciones en los contenidos de la tabla. Es decir, sólo recoge resultados de sentencias que: **añaden**, **modifican** o **borran** registros.

Manejo de fechas en las consultas

MySQL dispone de algunas cláusulas de gestión de fechas que pueden tener una gran utilidad a la hora de gestionar consultas. Son las siguientes:

DATE_FORMAT(*campo,formato)*

BETWEEN	**Numérico o Cadena**	WHERE *campo* BETWEEN valor1 AND valor2	Selecciona los registros en los que los valores contenidos en el *campo* seleccionado están comprendidos en el intervalo *valor1* (mínimo) – *valor2* (máximo) incluyendo en la selección ambos extremos. Cuando los contenidos de los campos son *cadenas* sigue los mismos criterios que se indican para los demás operadores de comparación	Ver	Probar
IS NULL	**Cadena**	WHERE *campo* IS NULL	Selecciona los registros en los que los valores contenidos en el *campo* seleccionado son NULOS	Ver	Probar
IS NOT NULL	**Cadena**	WHERE *campo* IS NOT NULL	Selecciona los registros en los que los valores contenidos en el *campo* seleccionado son NO NULOS	Ver	Probar

(*) Cuando se trata de cadenas de caracteres, el concepto **menor que** significa **anterior** en la ordenación de los caracteres según su código ASCII y **mayor que** significa **posterior** en esa misma ordenación.
(**) La discriminación de Mayúsculas/Minúsculas dependerá del tipo de campo.
Recuerda que los tipo **BLOB** hacen esa discriminación, mientras que los de tipo **TEXT** son insensibles a Mayúsculas/Minúsculas.

Cuando se trata de comparar **cadenas** MySQL dispone de una potente instrucción (**LIKE**) que permite establecer los criterios de selección a **toda** o **parte** de la cadena. Su sintaxis contempla distintas posibilidades utilizando **dos comodines>**: **%** (que se comporta de forma similar al (*) en las búsquedas de Windows) y _ (de comportamiento similar a (?) en Windows). Aquí tienes algunas de sus posibilidades:

Sintaxis	Descripción	Código fuente	Ver ejemplo
WHERE *campo* LIKE '%*cadena*%'	Selecciona todos los registros que contengan la **cadena** en el **campo** indicado sea cual fuere su posición	Ver	Probar
WHERE *campo* LIKE '*cadena*%'	Selecciona todos los registros en los que el **campo** indicado que contengan la **cadena *exactamente*** al **principio del campo**	Ver	Probar
WHERE *campo* LIKE '%*cadena*'	Selecciona todos los registros en los que el **campo** indicado que contengan la **cadena *exactamente*** al **final del campo**	Ver	Probar
WHERE *campo* LIKE '_*cadena*%'	Selecciona todos los registros en los que el primer caracter del **campo** puede ser cualquiera pero los siguientes han de ser **exactamente** los indicados en **cadena** pudiendo ir seguidos de ***cualesquiera otros caracteres***	Ver	Probar

El comodín (_) puede ir tanto al principio como al final y puede repetirse tantas veces como sea necesario. Seria correcto **LIKE '___es%'** y también **LIKE 'a___es%'** así como: **LIKE '%a___es'**.

Como ves, *un montón* de posibilidades.

Aun tiene más opciones **WHERE** ya que acepta múltiples condiciones vinculadas por los operadores lógicos **AND**, **OR**, **NOT** o sus *sintaxis equivalentes*: **&&**, **||** y **!**.

El comportamiento de estos operadores es idéntico al descrito para sus homónimos de PHP. ¿Los recuerdas?... Aquí los tienes... por si acaso.

Un ejemplo de sintaxis puede ser:

WHERE (*campo1=valor* **AND** *campo2 LIKE '_cadena%***)**

Utilizando *funciones sobre campos*

La sintaxis

SELECT MAX(*campo1*), MIN (*campo2*), ... FROM *tabla*

nos devolvería UNA SOLA FILA cuyos valores serían los resultados de la aplicación de las funciones
a todos los registros
del campo indicado.

Aquí tienes un ejemplo que determina todos los valores de esos estadísticos aplicados al campo
Contador
de nuestra famosa tabla
demo4
.

Aquí está el ejemplo

Ver código fuente	Ejecutar la consulta

Las diferentes opciones de formato las tienes en la tabla de la derecha. Es importante tener en cuenta que la sintaxis correcta es **%Y** (sin espacio) ya que si hubiera un espacio **% Y** interpretaría la letra Y como un texto a incluir.

CURDATE()

Dentro de **DATE_FORMAT** se puede incluir -en vez del nombre del campo- una cadena en la que se indique una fecha en formato *YYYY-MM-DD hh:mm:ss*. Puedes verlo en los ejemplos. De igual modo es posible sustituir el nombre del campo -o la cadena- por la función **CURDATE()** que recoge la **fecha actual del sistema** (únicamente día, mes y año). A efectos de horas, minutos y segundos **CURDATE()** va a tomar el *mediodía* de la fecha actual.

CURTIME()

Se comporta de forma similar a CURDATE().

Devuelve la hora actual del sistema que alberga el servidor MySQL en formato **hh:mm:ss**

CURRENT_TIMESTAMP()

Se comporta de forma similar a CURDATE().

Devuelve la fecha y hora actual del sistema en formato **YYYY-MM-DD hh:mm:ss**

NOW()

Es un *alias* de **CURRENT_TIMESTAMP()**.

mysql_result($resultado,num, campo)

Esta función PHP permite obtener un solo campo de uno solo de los registros obtenidos como resultado de una consulta MySQL.

El parámetro *$resultado* es la variable que recoge en resultado obtenido de la ejecución de *mysql_query* de forma idéntica a como lo hacíamos en otras consultas.

El valor *num* es un número entero que indica el número de fila de la que queremos extraer el valor contenido en uno de sus campos.

El valor *campo* indica el *número del campo* que tratamos de extraer. Este número (la primera posición siempre es **cero**) indica el número de orden del campo tal como está especificado en la sentencia SELECT. Si en esta sentencia se incluyera * (extraer todos los campos) consideraría el orden en el que está creada la estructura de la tabla que los contiene.

Este es el código fuente de un ejemplo comentado y este un enlace de prueba del script.

Aplicando la opción GROUP BY

Tal como señalamos al margen, las funciones anteriores pueden aplicarse a **grupos** de registros seleccionados mediante un criterio **GROUP BY (***nombre del campo***)**

En este ejemplo obtendremos los mismos parámetros estadísticos que en el anterior, pero ahora agrupados por **sexo**, lo que significaría que obtendremos **dos filas** de resultados. Aquí tienes el ejemplo

Ver código fuente	Ejecutar la consulta

Como habrás podido observar, la opción SELECT tiene un sinfín de posibilidades.

Creación de tablas a partir de la consulta de otra tabla

Es frecuente -podría decirse que es lo habitual- **relacionar tablas** mediante campos con **idéntico contenido**.

Supongamos que **entre los individuos** de nuestra tabla **demo4** se pretende establecer un **proceso de selección** para elegir entre ellos un número determinado de *astronautas*, pongamos por caso.

Supongamos también, que la *selección* va a constar de **tres** pruebas que serán *juzgadas* y calificadas por *tres tribunales* distintos.

Una primera opción sería crear tres tablas -una para cada tribunal- e incluir en ellas todos los datos de cada uno de los individuos.

Esa opción es factible pero no es *ni la más cómoda*, ni tampoco es la más *rápida* ni la que *menos espacio de almacenamiento* necesita. No debemos olvidar que una tabla puede tener una enorme cantidad de registros.

Una opción alternativa sería crear **tres nuevas tablas** que sólo contuvieran **dos campos** cada una. Por ejemplo el campo **DNI** y el campo **Calificación**.

Como quiera que el campo **DNI** ha de contener los mismos valores en las **cuatro** tablas y además es un campo **único** podrían crearse las nuevas tablas y luego **copiar** en cada una de ellas **todos los DNI** de la tabla original.

Nos garantizaría que *no habría errores* en los DNI y además nos garantizaría que **se incluyeran todos** los aspirantes en esas nuevas tablas.

Aquí tienes el *código fuente* de un *script* que crea esas tres tablas (a las que hemos llamado **demodat1**, **demodat2** y **demodat3**.

Ver código fuente

Crear las tablas anteriores

Una consulta *conjunta* de varias tablas

MySQL permite realizar consultas simultáneas en registros situados en varias tablas.

Para ese menester se usa la siguiente sintaxis:

SELECT *tabla1.campo1*, *tabla2.campo2*, ... FROM *tabla1*, *tabla2*

en la que, como ves, modificamos
ligeramente
la sintaxis ya que
anteponemos el nombre de la tabla
al del
campo
correspondiente separando ambos nombres por
un punto
, con lo cual no hay posibilidad de error de identificación del campo
incluso cuando campos de distinta tabla tengan el mismo nombre
.

Otra
innovación
-respecto a los ejemplos anteriores- es que detrás de la
cláusula
FROM
escribimos los nombres de todas las tablas que está usando
SELECT
.

A partir de ahí se pueden establecer todo tipo de relaciones para las sentencias
WHERE
,
ORDER BY
y
GROUP BY
utilizando para ello
campos de cualquiera de las tablas

sin otra particularidad más que
poner cuidado al aludir a los campos
utilizando siempre la sintaxis
nombre_tabla.nombre_campo
.

A modo de ejemplo -hemos procurado comentarlo línea a línea- aquí tienes un *script*
PHP que hace una
consulta conjunta
de las tablas
demo4
,
demodat1
,
demodat2
y
demodat3
y nos presenta una tabla con los datos personales y las puntuaciones de las *tres pruebas*
así como las
suma de puntos
de las tres y, además, ordena los resultados -de mayor a menor- según la
suma de las tres puntuaciones
.

Ver código fuente	Ejecutar la consulta

Formatos de fechas en consultas MySQL

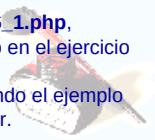
Los formatos soportados por la función DATE_FORMAT format son los siguientes:

Formato	Descripción	Sintaxis	Ver código	Ver ejemplo
%d	Día del mes en formato de dos dígitos	DATE_FORMAT(Nacimiento,'%d')	Ver	Probar
%e	Día del mes en formato de uno ó dos dígitos	DATE_FORMAT(Nacimiento,'%e')	Ver	Probar
%D	Número de día seguido del sufijo en inglés	DATE_FORMAT(Nacimiento,'%D')	Ver	Probar
%m	Número del mes en formato de dos dígitos	DATE_FORMAT(Nacimiento,'%m')	Ver	Probar
%c	Número del mes en formato de uno o dos dígitos	DATE_FORMAT(Nacimiento,'%c')	Ver	Probar
%M	Nombre del mes (en inglés)	DATE_FORMAT(Nacimiento,'%M')	Ver	Probar
%b	Nombre del mes abreviado (en inglés)	DATE_FORMAT(Nacimiento,'%b')	Ver	Probar
%y	Número del año en formato de dos dígitos	DATE_FORMAT(Nacimiento,'%y')	Ver	Probar
%Y	Número del año en formato de cuatro dígitos	DATE_FORMAT(Nacimiento,'%Y')	Ver	Probar
%w	Número de día de la semana 0=Domingo ... 6=Sábado	DATE_FORMAT(Nacimiento,'%w')	Ver	Probar
%W	Nombre del día de la semana (en inglés)	DATE_FORMAT(Nacimiento,'%W')	Ver	Probar
%W	Nombre abreviado del día de la semana (en inglés)	DATE_FORMAT(Nacimiento,'%W')	Ver	Probar
%j	Número de día del año en formato de 3 dígitos	DATE_FORMAT(Nacimiento,'%j')	Ver	Probar
%U	Número de semana del año considerando el DOMINGO como primer día de la semana (en formato de dos dígitos)	DATE_FORMAT(Nacimiento,'%U')	Ver	Probar
%u	Número de semana del año considerando el LUNES como primer día de la semana (en formato de dos dígitos)	DATE_FORMAT(Nacimiento,'%u')	Ver	Probar
La fecha para los ejemplos siguientes la extraemos de una variable del tipo: **$fecha="2005-10-12 14:23:42"** ya que la tabla no contiene campos de fecha que incluyan horas, minutos y segundos				
%H	Hora con dos dígitos (formato 0 a 24 horas)	DATE_FORMAT($fecha,'%H')	Ver	Probar
%k	Hora con uno ó dos dígitos (formato 0 a 24 horas)	DATE_FORMAT($fecha,'%k')		
%h	Hora con dos dígitos (formato 0 a 12 horas)	DATE_FORMAT($fecha,'%h')		
%I	Hora con uno ó dos dígitos (formato 0 a 12 horas)	DATE_FORMAT($fecha,'%I')		
%i	Minutos con dos dígitos	DATE_FORMAT($fecha,'%i')		
%s	Segundos con dos dígitos	DATE_FORMAT($fecha,'%s')		
%r	Hora completa (HH:mm:ss) en formato de 12 horas indicando AM ó PM	DATE_FORMAT($fecha,'%r')		
%T	Hora completa (HH:mm:ss) en formato de 24 horas	DATE_FORMAT($fecha,'%T')		
% texto	Incluye el texto que se indica detrás del %	DATE_FORMAT($fecha,'% texto')		

%p	Añade AM ó PM dependiendo de la **Hora**	DATE_FORMAT($fecha,'%p')
Se pueden combinar a voluntad varias opciones utilizando una sintaxis de este tipo: '% Hoy es: %d - %m - %Y % es %W % estamos en el mes de %M % y van transcurridos %j % dias de este año. Son las %r'		

Ejercicio nº 41

En esta actividad debes elaborar varios scripts –puedes llamarlos **ejercicio26_1.php**, etcétera– que permitan realizar consultas en la base de datos que has creado en el ejercicio nº 23.
Previamente, tendrías que añadirle datos, bien manualmente o bien modificando el ejemplo de generación de registros aleatorios que hemos incluido en la página anterior.



Ejercicio nº 42

Construye una nueva tabla –**tabla2**– con los mismos campos que tu **tabla1** pero añadiendo el carácter de clave principal al campo que recoge el DNI, con lo cual podrás impedir que puedan repetirse dos alumnos con el mismo DNI.
A partir de ella, crea tablas auxiliares (transfiriendo los datos de **tabla1**) –de calificaciones de materias, por ejemplo– que contengan dos campos: DNI y calificación.
Por último, tendrías que crear todo lo necesario para que el profesor de cada materia, pudiera insertar sus calificaciones y, además, crea un documento final que permita visualizar simultáneamente las calificaciones del alumno en todas la materias.