SQL con MySQL 5

Gerardo A. Otero Rodríguez

publicatuslibros.com

SQL con MySQL 5

Gerardo A. Otero Rodríguez

Publicatuslibros.com es una iniciativa de:

Íttakus, sociedad para la información, S.L.
C/ Sierra Mágina,10
23009 Jaén-España
Tel.: +34 902 500 421
www.ittakus.com

SQL con MySQL 5

Prólogo

Mi corta pero intensa experiencia en el mundo de la docencia me ha llevado a impartir, en varias ocasiones, módulos de Ciclos Formativos de Informática relacionados directamente o de algún modo con las Bases de Datos. Ya durante mi etapa como analista programador en diferentes consultoras me había acercado con cierta profundidad al lenguaje SQL y al manejo de las Bases de Datos.

En el mundo de la informática, antes o después, uno se da cuenta de que es necesario especializarse y cuanto antes lo haga mejor. Debido a este acercamiento en parte obligado por mi trabajo y en parte por ser una de las ramas de esta profesión que más me apasionan, han hecho que, en cierto modo, me haya acabado especializando en Base de Datos.

Uno acaba consultando algunos de las innumerables referencias bibliográficas que existen sobre este tema y acaba acogiendo como manuales de referencia y, en muchos casos, libros de cabecera, a algunos de ellos. Este es el caso destacable de "*SQL Para usuarios y programadores*" que tomé como punto de partida para la construcción de este libro, sin olvidarse de todas la bibliografía que he ido consultando a lo largo de mi experiencia profesional.

¿Por qué MySQL? Durante toda mi anterior carrera profesional me había dedicado práctica y exclusivamente a entornos ORACLE pero, una vez en el mundo de la docencia, me di cuenta de que era un entorno demasiado poco amigable para utilizar con alumnos no iniciados. Eso por no decir que se trata de un producto de pago y con entornos gratuitos bastante limitados. MySQL sin embargo es gratuíto, está profusamente extendido y permite la utilización de diferentes entornos – también gratuitos -, amigables y bastante potentes, teniendo en cuenta hacia quien va dirigido. También opino que la palabra GRATUITO/A debería ser imprescindible en la docencia, más aún en la docencia pública.

¿Por qué este libro? Pese a la existencia del anterior libro mencionado y de otros muchos que podían adaptarse en mayor o menor medida a mis necesidades, hay ciertas cosas que uno siempre tiene que acabar modificando, entornos o aplicaciones que se van quedando obsoletas, estructura que no encaja del todo en tu concepto de enseñar la asignatura, funciones en desuso, terminología, diversidad de lenguajes, redacción, conceptos nuevos,... Todos estos motivos y su adaptación a MySQL propiciarion la construcción de este libro.

Quiero dejar claro que no trato de apropiarme de ningún tipo de conocimiento que en él expongo. Todo es fruto de la documentación consultada y del trabajo y experiencia desarrollados

durante todos estos años. Simplemente he tratado de construir una herramienta para poder utilizar tanto yo - y quien lo desee, por ello recurro a la licencia creative commons – como mis alumnos en el desarrollo de mis clases o para ser utilizado a modo de pequeño manual de consulta. Y creo haberlo conseguido.

Instalación

Previamente voy a proceder a explicar la instalación del software necesario para poder utilizar SQL con MySQL. Para ello necesitaremos instalar, como mínimo, un servidor de Base de Datos MySQL, aunque también instalaremos un cliente para poder realizar las consultas de una manera más sencilla, sin tener que utilizar la consola del servidor. Es cierto que todas las consultas que vamos a necesitar se podrían ejecutar directamente en la consola pero, como ya digo, por motivos de comodidad, rapidez y flexibilidad se hace imprescindible el uso de un cliente. El cliente elegido será phpMyAdmin por diversas razones:

1. Es software libre.
2. Es sencillo de instalar e utilizar, simplemente basta con abrir un navegador y acceder a la dirección del servidor.
3. Es multiplataforma: lo podemos instalar/usar en Linux, Windows, Mac OS,... de la misma forma.
4. Cumple con creces todas las funcionalidades que necesitamos para implementar o probar cada uno de los capítulos de este libro.

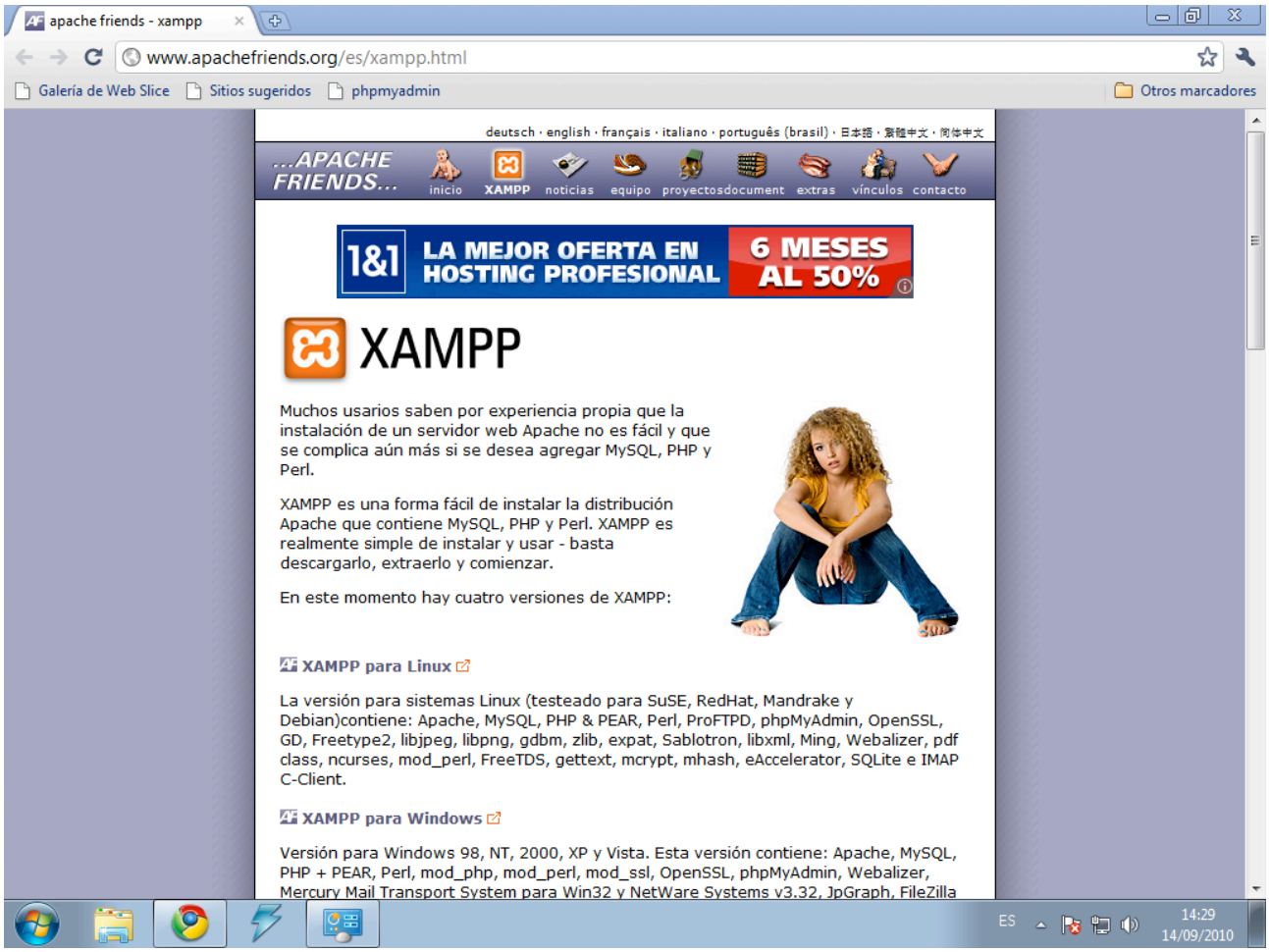
Para instalar phpMyAdmin necesitaríamos instalar primero PHP, ya que está programado en ese lenguaje. También necesitaríamos instalar un servidor de aplicaciones, como Apache, para poder instalar PHP después como módulo. A modo de resumen, los pasos a seguir serían entonces los siguientes:

1. Instalción del servidor de Bases de Datos MySQL.
2. Instalación del servidor web Apache.
3. Instalación de PHP.
4. Instalación del cliente phpMyAdmin.

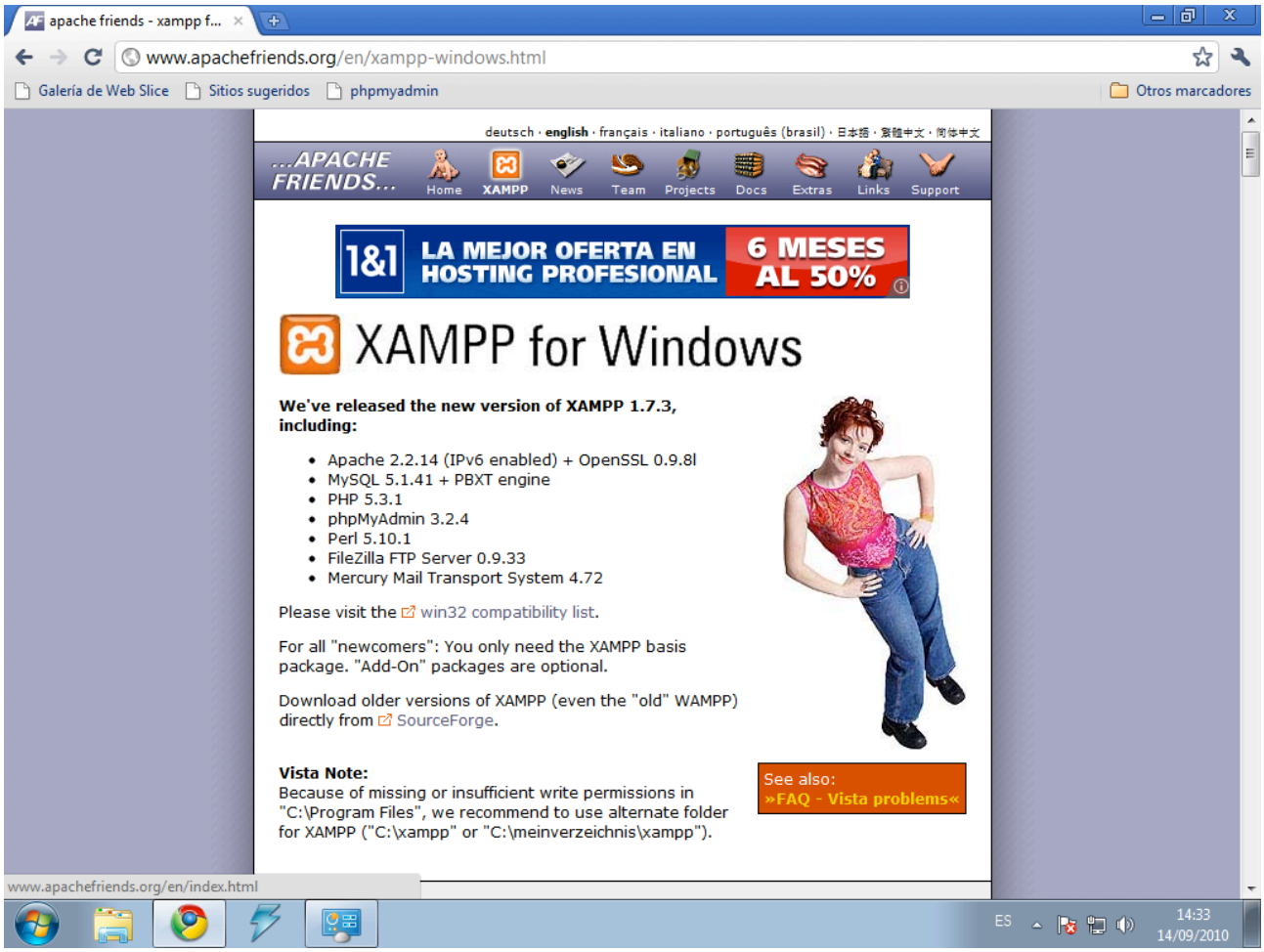
Todos estos pasos se pueden realizar de manera manual pero existen diferentes soluciones en el mercado que posibilitan su instalación de un sólo golpe. Una de ellas será XAMPP (acrónimo de Apache – MySQL – PHP – Perl) que, aparte de todo el software que necesitamos, trae aplicaciones como un servidor FTP (Filezilla Server). Otras soluciones podrían ser WAMP (Windows), LAMP (Linux), MAMP (Mac OS),... Así que lo primero será acudir a la página web del producto seleccionado y descargárnoslo.

La dirección web es la siguiente:

http://www.apachefriends.org/es/xampp.html

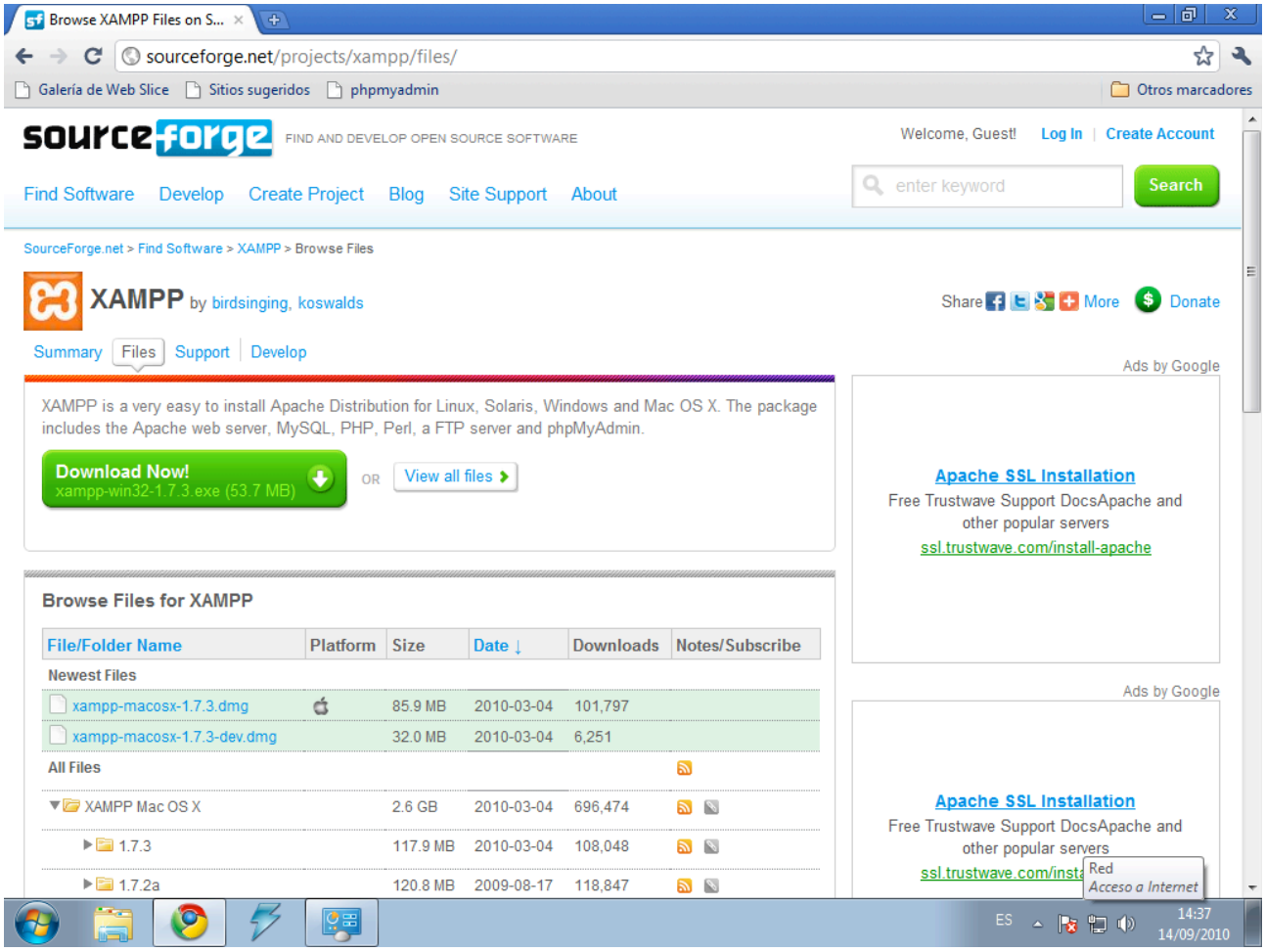


Elegimos la opción para Windows aunque, como se puede ver, existen diferentes opciones para diferentes Sistemas Operativos. Incluso lo hay portable, es decir, ejecutable desde una unidad externa o pendrive.

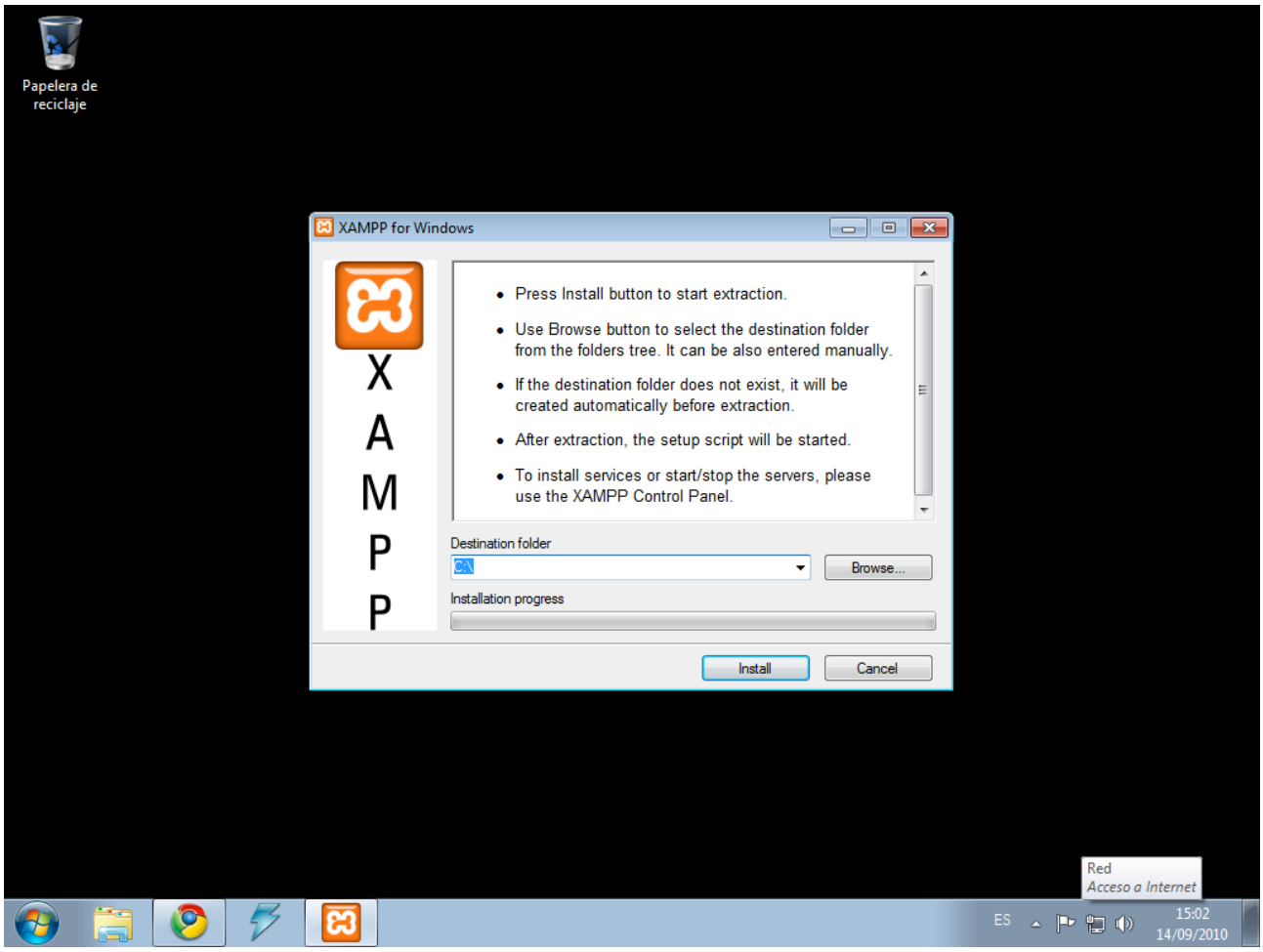


Nos indica las versiones de todo el software que vamos a instalar. Nos fijamos en las versiones de MySQL (5.1) y de phpMyAdmin (3.2).

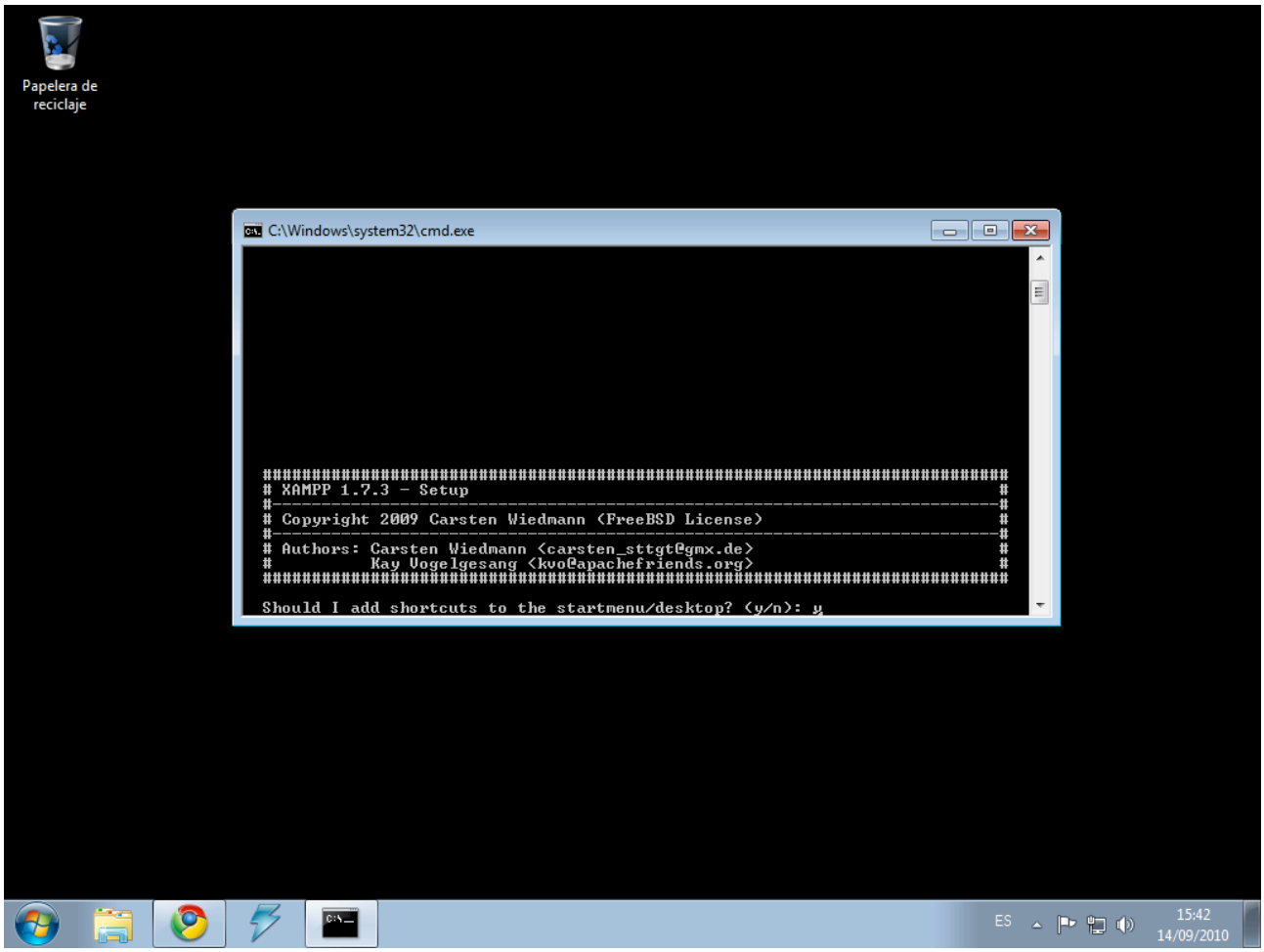
Descargamos el fichero ejecutable .EXE desde Sourceforge. En este caso es la versión 1.7.3 de XAMPP.



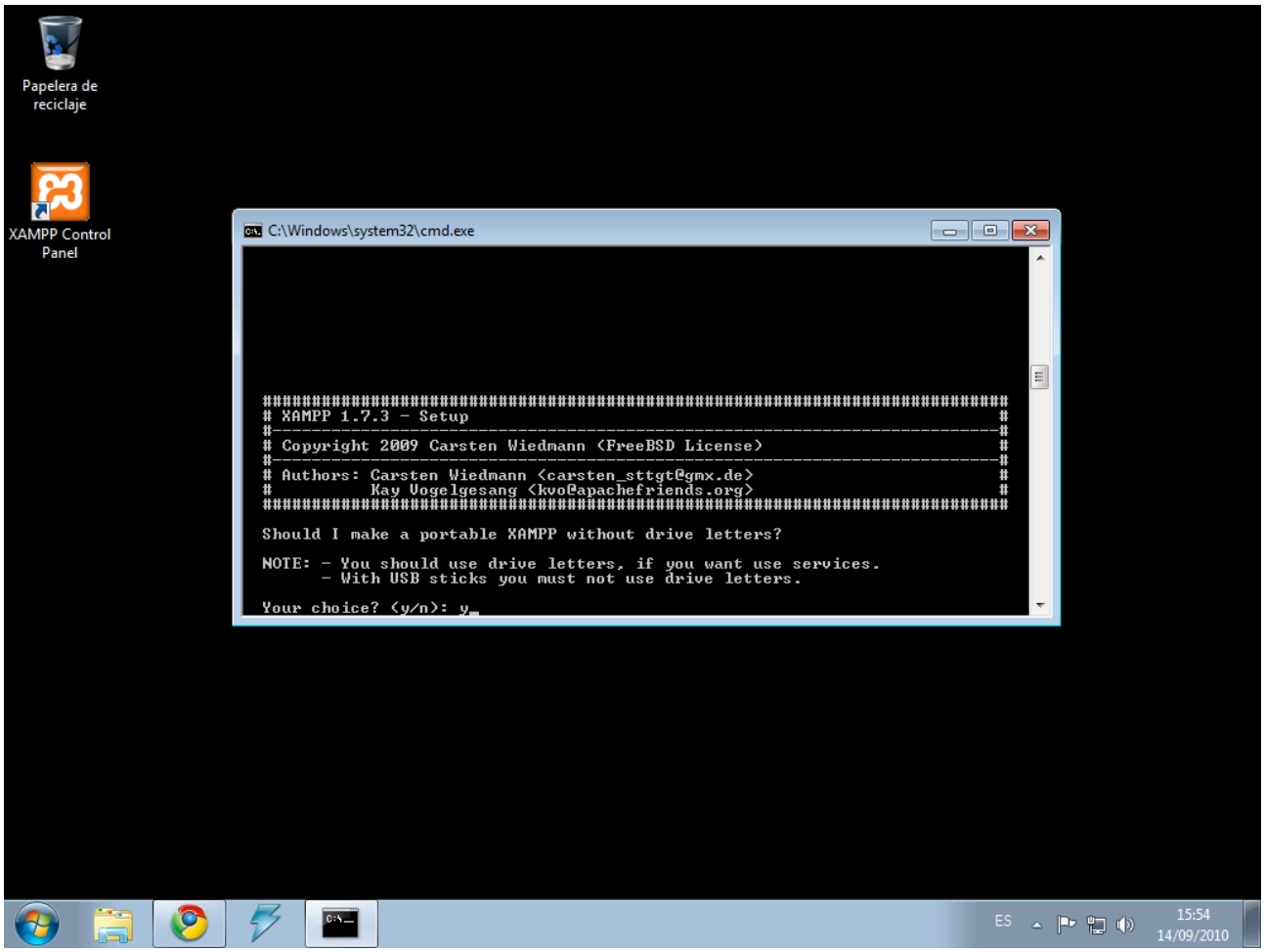
Una vez descargado el fichero en nuestro PC, ejecutamos el fichero de instalación. Lo podemos instalar directamente en el raíz. Es lo más cómodo y lo más recomendable si luego queremos acceder a los diferentes productos instalados.



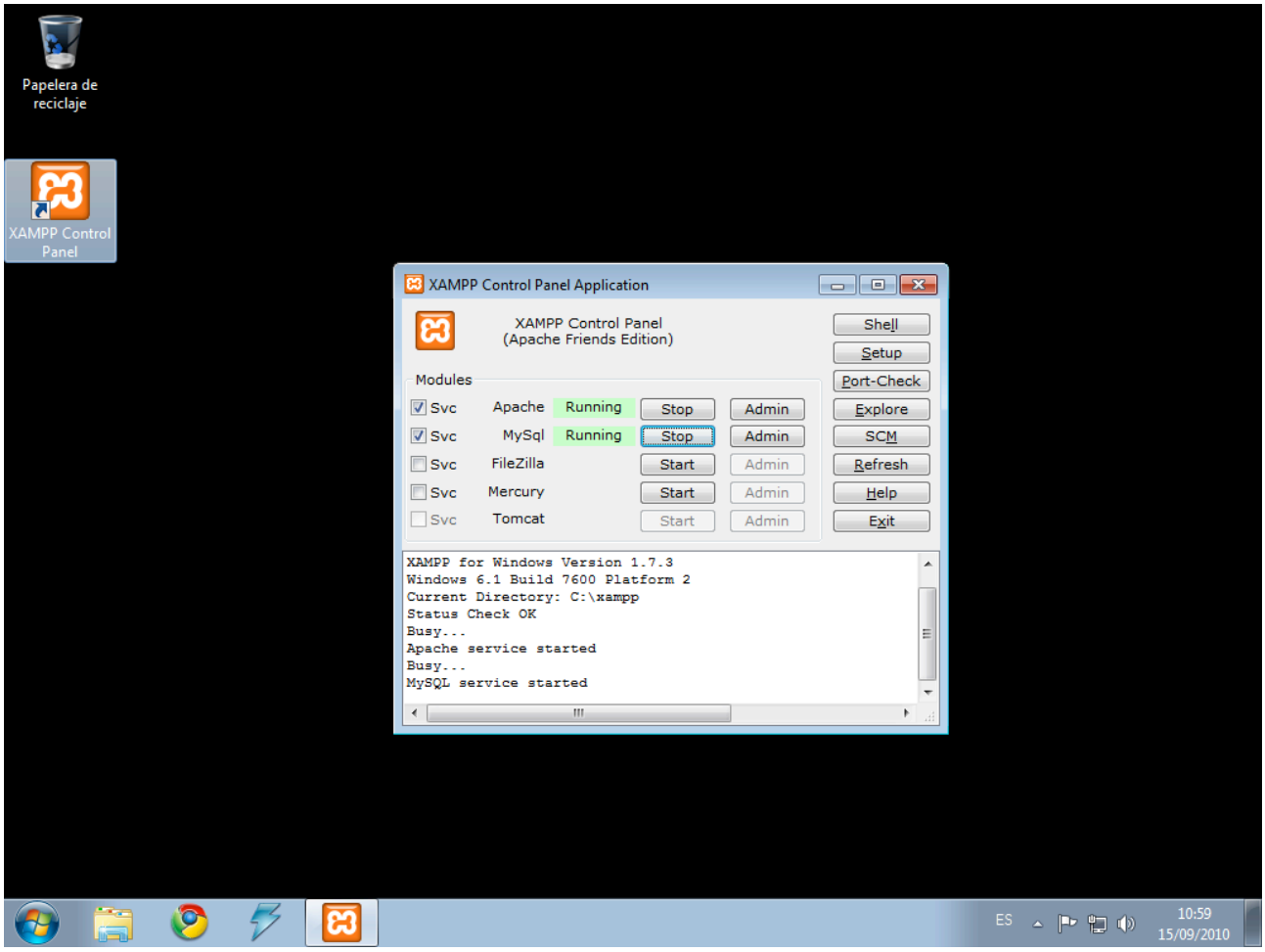
Una vez extraiga todos los ficheros correspondientes al programa pasamos a configurar alguna de las opciones del programa. Por ejemplo, la creación de accesos directos en el escritorio.



Y algo importante como el la asociación directa con letras de unidades. En el caso de que queramos utilizar XAMPP como servicio deberemos indicar que SI. Si queremos utilizar XAMPP desde una unidad externa deberemos indicar que NO. Nosotros vamos a indicarle que “SI” ya que resulta mucho más cómodo que el programa se inicie como servicio y no tener que estar ejecutando el panel de control y levantando los servidores cada vez que lo necesitemos.

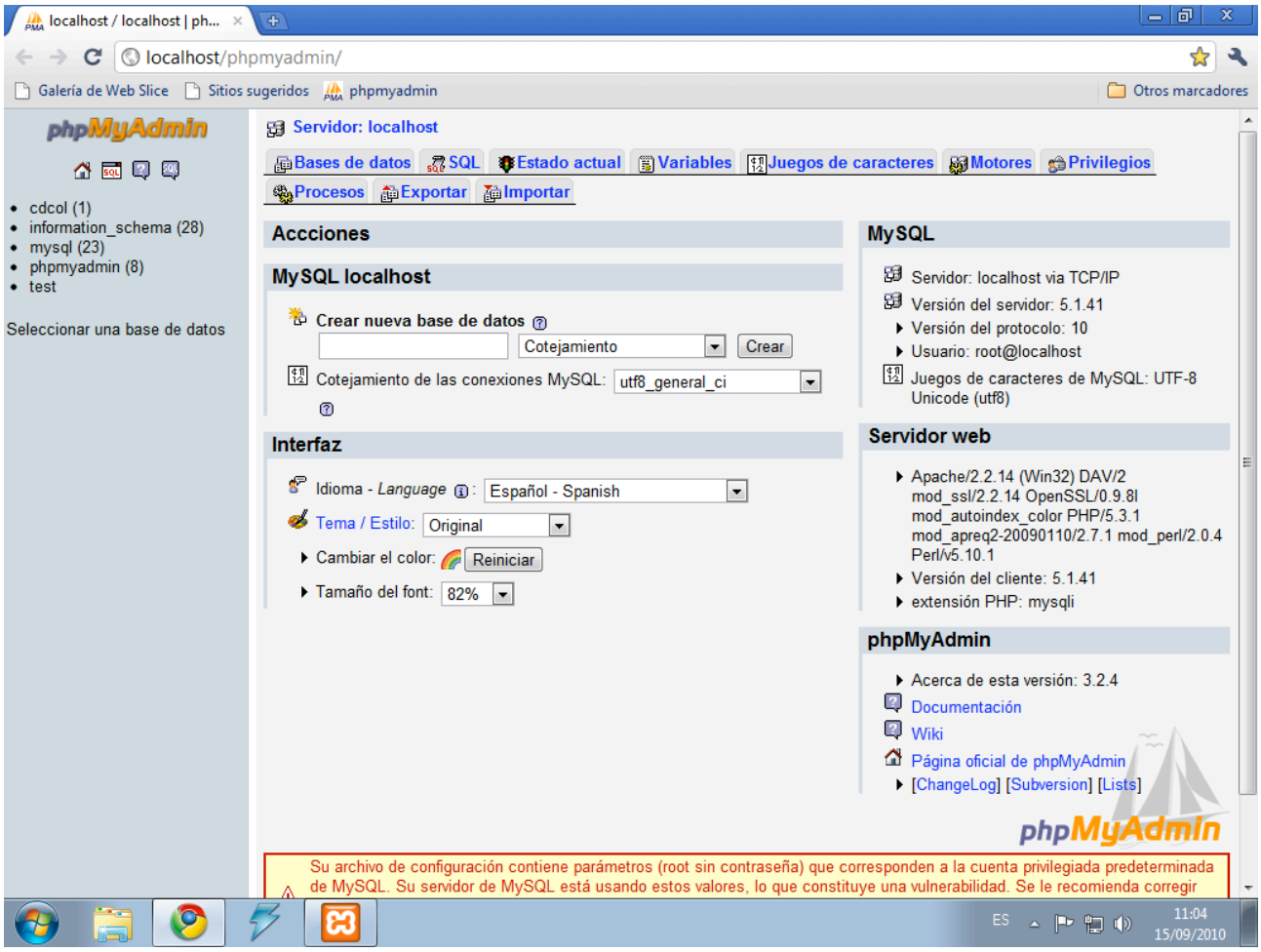


Una vez arrancada la consola de administración de XAMPP comprobamos que tenga los servicios de Apache y de MySQL levantados. Si no es así los levantamos y activamos la casilla svc (services). Pulsamos en el botón Start en cada uno de ellos si es que no están levantados. Filezilla, Mercury y Tomcat los dejamos porque no son necesarios.



El objetivo es obtener tanto en Apache como en MySQL un “Running”.

Una vez instalado y levantados los servidores, para acceder al cliente phpMyAdmin no tenemos más que pulsar en el botón “Admin” de MySQL y nos llevará a la página correspondiente.



Desde este momento ya podemos utilizar el cliente phpMyAdmin para utilizar las consultas de cada uno de los capítulos de este libro.

Simplemente recordar que para ejecutar las sentencias hay que dirigirse a la pestaña SQL para introducir el código de las mismas.

Capítulo 1 - SQL

1.1 Qué es SQL

SQL (Structured Query Language - Lenguaje Estructurado de Consultas) es un lenguaje para acceder a información almacenada en Bases de Datos relacionales.

SQL fue la evolución de SEQUEL, que surgió como solución al concepto formulado por el Sr. Codd de Bases de Datos Relacionales en 1970.

Permite expresar operaciones aritméticas, combinatorias y lógicas (entre otras) con los datos almacenados en Bases de Datos relacionales.

El ANSI (American National Standards Institute - Instituto Americano de Normas) ha publicado unas especificaciones para este lenguaje, pero esto no quiere decir que los productos disponibles las sigan estrictamente. Hay diferencias entre ellos y se recomienda leer los manuales de referencia de los mismos. Se van a seguir las especificaciones definidas por la casa IBM en sus normas SAA.

1.2 Como se usa SQL

Las peticiones sobre los datos se expresan en SQL mediante sentencias. Estas sentencias pueden:

- Escribirse directamente sobre el SGBD.
- Pueden escribirse embebidas en programas. Dentro de éstas hay dos técnicas:
    - SQL estático: las sentencias SQL no pueden cambiar en tiempo de ejecución. Más sencillo y eficiente.
    - SQL dinámico: una sentencia puede ser modificada por el propio programa durante su ejecución. Más flexible.

1.3 Para qué sirve SQL

- Consultar y actualizar datos almacenados.
- Definir y destruir objetos de la Base de Datos (generalmente un DBA).
- Conceder y denegar autorizaciones para usar esos objetos (generalmente un DBA).

1.4 Elementos componentes de una sentencia SQL

1. Palabras predefinidas (SELECT, FROM, WHERE, INTO,...)
2. Nombres de tablas y columnas.
3. Constantes (numéricas y alfanuméricas).
4. Signos delimitadores.

Por ejemplo:

```
SELECT nomce FROM tcentr WHERE numce = 10
```

- Palabras predefinidas: `SELECT`, `FROM`, `WHERE`.
- Nombres de tablas y columnas: nomce, tcentr, numce.
- Constantes: 10.
- Signos delimitadores: =.

1.5 Tipos de sentencias SQL

1. Sentencias DML (Data Manipulation Language)
    - `SELECT`.
    - `INSERT`.
    - `UPDATE`.
    - `DELETE`.
2. 2. Sentencias DDL (Data Definition Language)
    - `CREATE`.
    - `DROP`.
3. Sentencias DCL (Data Control Language)
    - `GRANT`.
    - `REVOKE`.

1.6 Tipos de Datos

Hay 3 grandes tipos de datos:

1. Numéricos (enteros, decimales, en coma flotante)
2. Alfanuméricos.
3. De tiempo.

Ya que vamos a utilizar MySQL 5.1 es recomendable visitar la web de referencia sobre sus tipos de datos.

1.7 Valores Nulos

Son valores especiales que se puede permitir asignar a cualquier tipo de dato. Puede interpretarse como desconocido. A la hora de definir la columna de la tabla que va a aceptar este tipo de valores habrá que indicar NULL. Para excluir este valor del dominio se especifican las palabras NOT NULL.

Capítulo 2 - DDL

El ***Data Definition Language*** (DDL) es un subconjunto de SQL que nos permite destruir y definir nuevos objetos en el SGBD, como por ejemplo: tablas, vistas, índices, etc.

La definición de un objeto se hace con la sentencia CREATE y su ejecución almacenará en el catálogo del SGBD la definición y el nombre del objeto.

La destrucción de un objeto, previamente existente, se realizará con la sentencia DROP. Cuando se ejecuta el SGBD borrará del catálogo su definición.

Considero oportuno empezar por DDL y no por DML porque creo que es más interesante aprender a hacer consultas sobre tablas ya creadas sabiendo cómo se crean.

2.1. Cómo crear Bases de Datos en MySQL con phpMyAdmin

Al igual que podemos crear tablas desde la opción de SQL del phpMyAdmin, Diego Torres nos dice que también se pueden crear las Bases de Datos que contengan estas tablas. La sintaxis es muy sencilla:

```
CREATE DATABASE [IF NOT EXISTS] nombre_de_la_BD;
```

El `IF NOT EXISTS` es para que no intente crear BD con nombres que ya existen actualmente.

2.2. Cómo crear tablas en MySQL con phpMyAdmin

Para crear tablas en phpMyAdmin sin utilizar el interface, es decir, usando sólo SQL hay que hacer lo siguiente:

1. Crear la Base de Datos de contendrá la tabla. Ya que sin Base de Datos poco se puede hacer.
2. Una vez creada la BD nos situamos en ella y pulsamos en la pestaña SQL.
3. Nos aparece una ventana en la que ya podemos introducir el código SQL que deseemos. La sintaxis es la siguiente:

```
CREATE TABLE Nombre_de_la_BD.Nombre_de_la_tabla (
campo1 TIPO_DE_DATO [(longitud)] NOT NULL/NULL [DEFAULT ...],
campo2 TIPO_DE_DATO [(longitud)] NOT NULL/NULL [DEFAULT ...],
campo3 TIPO_DE_DATO [(longitud)] NOT NULL/NULL [DEFAULT ...],
...
PRIMARY KEY (campo1, campo2,...) [, UNIQUE KEY
Nombre_de_la_restriccion1 (campo1) , UNIQUE KEY
```

```
Nombre_de_la_restriccion2 (campo2) ,...] )
```

4. Para acabar pulsamos en Continuar y listo, tabla creada.

Recordar que todo lo que vaya entre corchetes indica "opcional" ya que, por ejemplo, los tipos de dato `DATE`, `INT`, `DOUBLE` no llevan longitud. OJO: queda confirmado que no hay que utilizar ningún tipo de apóstrofe para el nombre de los campos ni de las restricciones.

Para declarar los campos auto-numéricos de Access tendremos que usar la palabra clave `AUTO_INCREMENT`, después de `NOT NULL`/`NULL`.

2.3. Cómo borrar una tabla en MySQL con phpMyAdmin

Borrar una tabla de la BD debe de ser una de las operaciones más sencillas de realizar en una Base de Datos. De las más sencillas y de las más **PELIGROSAS**. Sobra decir que debemos asegurarnos bien de que lo que vamos a borrar es lo que queremos borrar. La sentencia es la siguiente:

```
DROP TABLE [IF EXISTS] Nombre_de_la_BD.Nombre_de_la_tabla;
```

Sencillo, ¿verdad? Muchas veces querremos usar esta sentencia en combinación con la de creación de tablas, porque nos hemos equivocado en la estructura y queremos volver a crearla. No hay problema, en la pestaña de SQL podemos poner todas las sentencias de SQL que queramos que se ejecuten en proceso batch, eso sí, separadas siempre por ;

2.4. Cómo crear claves ajenas en MySQL con phpMyAdmin

¿Cómo implementar las FOREIGN KEYS o las restricciones de integridad referencial en SQL? Primero comentar que hemos de partir del grafo relacional, donde tendremos perfectamente definidas las restricciones de integridad pertinentes. Luego resulta muy sencillo: tan sólo tenemos que añadir unas opciones en nuestra sentencia `CREATE TABLE`. Por ejemplo, tenemos:

```
CREATE TABLE Mi_BD.Mi_tabla (
     cod_mi_tabla int NOT NULL,
     campo2 int NOT NULL,
     cod_tabla2 int,
     PRIMARY KEY (cod_mi_tabla))

CREATE TABLE Mi_BD.Mi_tabla2 (
     cod_mi_tabla2 int NOT NULL,
```

```
descripcion VARCHAR (50) NOT NULL,
PRIMARY KEY (cod_mi_tabla2))
```

Simplemente tendremos que añadir en la primera consulta lo siguiente:

```
CREATE TABLE Mi_BD.Mi_tabla (
     cod_mi_tabla int NOT NULL,
     campo2 int NOT NULL,
     cod_tabla2 int,
     PRIMARY KEY (cod_mi_tabla),
     FOREIGN KEY (cod_tabla2) REFERENCES Mi_tabla2
(cod_mi_tabla2))
```

Como se puede apreciar es muy sencillo, tan sólo hay que indicar la tabla que contiene la PK y el campo de la tabla en la que estamos. Sobra decir que Mi_tabla2 tiene que existir antes de crear Mi_tabla.

Obviamente no sólo se puede hacer con la sentencia `CREATE TABLE`, con `ALTER TABLE` podríamos hacer lo mismo, de la siguiente manera:

```
ALTER TABLE Mi_BD.Mi_tabla
ADD CONSTRAINT fk_Tabla2 FOREIGN KEY (cod_tabla2) REFERENCES
Mi_tabla2 (cod_mi_tabla2)
```

Si encuentro una manera de hacerlo "más gráfica" con phpMyAdmin 3.1.x editaré el artículo, pero hasta el momento es lo que hay. De todas maneras, con SQL siempre acertaremos, ya que los cambios de versiones en el interface provocan cambios en la manera de hacer las cosas.

Capítulo 3 – Consultas sencillas

SELECT es una palabra predefinida que se utiliza en SQL para extraer datos de una o varias tablas. Se utiliza para realizar consultas. El formato de esta sentencia sería

```
SELECT col1[,col2,col3,...]/*
FROM tabla
[WHERE predicado]
[ORDER BY columna]
```

La `tabla` se corresponde con la tabla de donde queremos extraer los datos.

El `predicado` es una condición que simplemente puede ser verdadera (TRUE) o no (FALSE). Podría ser, por ejemplo, una comparación entre valores.

ORDER BY sirve para ordenar la lista de resultados. Puede ser de menor a mayor (`ASC` - por defecto) o de mayor a menor (`DESC` - habría que indicarlo explícitamente). Si no se utiliza la cláusula `ORDER BY` el resultado podría salir de cualquier orden. Si hay valores nulos podrían presentarse al final de los demás. Aunque ya digo "podrían", dependerá del SGBD que estemos usando. La columna especificada en el `ORDER BY` debe de ser una de las especificadas en la cláusula `SELECT`. En vez de usar el nombre de una columna se puede usar también un número. Por ejemplo: `ORDER BY 2` Haría referencia a la segunda columna especificada en la cláusula `SELECT`. Si no hay dos columnas probablemente nos de un error.

Si queremos eliminar filas duplicadas usaremos la cláusula **DISTINCT** dentro de la cláusula `SELECT` de la siguiente manera:

```
SELECT DISTINCT columna
...
```

OJO: Aquí si que ya no podemos/debemos usar el asterisco.

Capítulo 4 - Expresiones

En una consulta se pueden realizar operaciones con los datos, por ejemplo: se pueden sumar datos, multiplicarlos por un cierto factor, concatenarlos (cuando se traten de cadenas),... Estos cálculos se especifican mediante **expresiones**.

Una **expresión** no es más que una combinación de operadores, operandos y paréntesis. En algún caso, como en la concatenación, el operador se tratará de una función SQL (`CONCAT`).

Se pueden emplear en la cláusula `SELECT` y en la cláusula `WHERE` (aunque aún no viéramos esta cláusula).

Con valores numéricos sólo se podrán realizar operaciones aritméticas con los siguientes operadores:

- + (sumar)
- - (restar y cambiar de signo)
- * (multiplicar)
- / (dividir)
- DIV (división entera)
- % (resto)

Con los operadores alfanuméricos no hay operadores definidos, pero una de las operaciones clásicas es la de concatenar. Se realiza con la función `CONCAT` de la siguiente manera:

```
CONCAT ('cadena1', ',cadena2', 'cadena3',...);
```

Se pueden concatenar constantes con constantes, constantes con variables y variables con variables:

```
CONCAT ('cadena1', campo1,...);
CONCAT (campo1, campo2,...);
```

Capítulo 5 - Predicados

Un predicado contiene las condiciones o restricciones que vamos a aplicar a nuestras sentencias. Se especifican en la cláusula `WHERE` y siempre dan como resultado Verdadero (True) o Falso (False). Por establecer una clasificación podríamos dividirlos en dos grandes subtipos: simples y compuestos. **Simples** son aquellos en los que simplemente se comparan dos valores mientras que los **Compuestos** serán el resto. Veamos:

- **Simples**: comparación entre dos valores. Podemos utilizar los operadores de comparación ya vistos (=, >, <) o las combinaciones entre ellos. Si alguno de ellos es nulo el predicado podría tomar un valor desconocido. El segundo elemento de la comparación podría ser el resultado de otra sentencia `SELECT` (sentencia subordinada) pero siempre debe ir entre paréntesis y producir como resultado un único valor. Vamos a ver diferentes tipos de predicados que podemos construir:
    - **NULL**: podemos preguntar si un valor es nulo o no.
        - `WHERE nombre_columna IS [NOT] NULL`
    - **Predicados cuantificados**: podemos anteceder a sentencias `SELECT` subordinadas diferentes cuantificadores. Disponemos de los siguientes cuantificadores en MySQL:
        - **ANY**: verdadero si se cumple para alguno de los valores
        - **SOME**: es un alias (AS) de ANY.
        - **ALL**: verdadero si se cumple para todos los valores.
            - `WHERE campo >= ALL (SUBSELECT)`
    - **BETWEEN**: nos permitirá comprobar si un valor se encuentra entre otros dos (ambos inclusive). Podemos usarla negada también. Excuso decir que este predicado se puede construir a partir de otros combinado usos de `AND` y `OR` (pero ya serían predicados compuestos).
        - `WHERE campo [NOT] BETWEEN valor_1 AND valor_2`
    - **LIKE**: nos permite comparar valores de manera similar a '=' aunque nos permite introducir salvedades. Por ejemplo:
        - Si tenemos cadenas y comparamos usando % le estaremos indicando que ahí puede ir cualquier carácter de cualquier longitud (incluso cero).
        - Si tenemos cadenas y comparamos usando _ le estaremos indicando que ahí puede ir SOLO UN carácter aunque podría ser cualquiera.
        - OJO, debemos tener cuidado de saber perfectamente lo que queremos hacer. El uso de estos predicados requiere un coste mucho

mayor que usar =.

- También se puede usar con el modificador `NOT`:
  - `WHERE campo [NOT] LIKE valor`

- **IN**: sirve para preguntar si el resultado de la expresión está incluido en la lista de valores que especificamos a continuación. OJO con los nulos.
  - `WHERE campo IN (subselect)` - Si se fija uno es equivalente a `WHERE campo = ANY (subselect)`
  - En vez de una subselect podemos usar una lista de constantes separadas por comas:
    - `WHERE campo [NOT] IN (valor1, valor2, valor3,...)`
  - Lo mismo, podemos utilizar el modificador NOT:
    - `WHERE campo [NOT] IN (subselect)`

- **EXISTS**: podremos conocer si el resultado de ejecutar una subselect devuelve una o más filas o, por el contrario, ninguna.
  - `WHERE [NOT] EXISTS (subselect)`

- **Compuestos**: simplemente son combinaciones de predicados simples utilizando los operadores lógicos `AND`, `OR` y `NOT`.

Capítulo 6 – Funciones escalares en MySQL

Una función escalar es aquella en la que su resultado se obtiene a partir de un valor único. Algunos ejemplos de funciones escalares serían:

- **LENGTH**: para obtener la longitud de un valor cualquiera. Tiene su sentido si se aplica a tipos de datos alfanuméricos. La sintaxis será:
    - `LENGTH (columna)`
- **SUBSTR**: para obtener subcadenas. Se puede usar de igual manera que SUBSTRING. La sintaxis será:
    - `SUBSTR (cadena, posicion_de_inicio)`

  De manera opcional se puede indicar la longitud de la subcadena resultante, por ejemplo:
    - `SUBSTR (cadena, posicion_de_inicio, longitud)`

Lo de siempre, OJO con los nulos.

Capítulo 7 – Fecha y Hora

7.1 Tipos de fecha y hora en MySQL

Los tipos de datos soportados por MySQL 5 para reflejar información de fechas y horas son `DATETIME`, `DATE`, `TIMESTAMP`, `TIME`, y `YEAR`:

- El tipo **DATE** se usa cuando necesita sólo un valor de fecha, sin una parte de hora. MySQL recibe y muestra los valores `DATE` en formato ISO 'YYYY-MM-DD'.
- El tipo **DATETIME** se usa cuando necesita valores que contienen información de fecha y hora. MySQL recibe y muestra los valores `DATETIME` en formato 'YYYY-MM-DD HH:MM:SS'.
- El tipo **TIME** se utiliza cuando se necesitan almacenar valores con información de horas. MySQL devuelve y muestra los valores `TIME` en formato 'HH:MM:SS' (o formato 'HHH:MM:SS' para valores de hora grandes).
- El tipo **YEAR** es un tipo de un byte usado para representar años. MySQL devuelve y muestra los valores `YEAR` en formato YYYY. El rango es de 1901 a 2155.
- El tipo **TIMESTAMP** es muy similar al tipo `DATETIME` a partir de MySQL 5.

7.2 Funciones de Fecha y Hora en MySQL

Como ya iremos sabiendo a estas alturas hay un montón de funciones auxiliares que MySQL nos proporciona y que nos pueden ayudar a la hora de construir nuestras sentencias. Con el caso de las fechas y de las horas no va a ser diferente. En el siguiente enlace tenemos un listado de todas las funciones de fecha y hora existentes en MySQL 5.1. No las vamos a ver todas en detalle pero si algunas de las más importantes o más útiles:

Para saber fechas o momentos actuales

- **CURDATE()/CURRENT_DATE()**: Devuelve la fecha actual en formato 'YYYY-MM-DD' o YYYYMMDD.
- **CURTIME()/CURRENT_TIME()**: Devuelve la hora actual en formato 'HH:MM:SS' or HHMMSS.uuuuuu.
- **NOW()/CURRENT_TIMESTAMP()**: Devuelve el momento actual en formato 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu.

Para conocer elementos de una fecha o tiempo

- **YEAR(fecha)**: Devuelve el año de la fecha que le pasemos. Entre 1000 y 9999.

- **MONTH(fecha)**: Devuelve el mes de la la fecha que le pasemos. Entre 0 y 12.
- **DAY(fecha)/DAYOFMONTH(fecha)**: Devuelve el día del mes de la la fecha que le pasemos. Entre 0 y 31. También podremos preguntar por:
    - El día de la semana `DAYOFWEEK(fecha)`. Un número entre 1 y 7.
    - El día del año `DAYOFYEAR(fecha)`. Un número entre 1 y 366.
- **HOUR(tiempo)**: Devuelve la hora del tiempo que le pasemos. Entre 0 y 23.
- **MINUTE(tiempo)**: Devuelve los minutos del tiempo que le pasemos. Es decir, si hacemos esto `SELECT MINUTE('2008-02-03 10:05:03')` nos devolverá un 5. Toma valores entre 0 y 59.
- **SECOND(tiempo)**: Devuelve los segundos del tiempo que le pasemos. Es decir, si hacemos esto `SELECT SECOND('2008-02-03 10:05:03')` nos devolverá un 3. Toma valores entre 0 y 59.

Para transformar fechas en cadenas o cadenas en fechas

- **STR_TO_DATE(cadena,formato)**: A partir de la cadena que le pasamos y la cadena con el formato que queremos nos devuelve un valor DATETIME si la cadena de formato contiene parte de fecha y hora, o un valor DATE o TIME si la cadena contiene sólo parte de fecha o hora. Por ejemplo:
    - `SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');` nos devolvería '2004-04-31'
- **DATE_FORMAT(fecha,formato)**: Formatea la fecha que le pasemos según el formato que queramos . Por ejemplo:
    - `SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');` nos devolvería 'Saturday October 1997'
- **GET_FORMAT(tipo de dato fecha, tipo de formato)**: Devuelve una cadena con el tipo de formato que tiene la fecha. Es muy útil en combinación con las funciones DATE_FORMAT Y STR_TO_DATE. Por ejemplo:
    - `SELECT GET_FORMAT(DATE,'USA'));` nos devolvería '%m.%d.%Y'
    - `SELECT GET_FORMAT(DATETIME,'USA'));` nos devolvería '%Y-%m-%d %H.%i.%s'
- **MAKEDATE(año, día del año)**: Devuelve la fecha correspondiente al día del año del año que le pasamos. Si le pasamos un valor <= 0 el resultado será NULL. Por ejemplo:
    - `SELECT MAKEDATE(2011,31);` nos devolvería "2011-01-31"

Para operar con fechas

- **DATEDIFF(fecha mayor,fecha menor)**: A partir de las dos fechas que le pasamos como parámetros nos devuelve la diferencia en días entre las dos. Por ejemplo:

    - `SELECT DATEDIFF(CURRENT_DATE(),fecna);`
    - OJO: la fecha mayor debe de ser la primera fecha que le pasamos como parámetro, sino nos devolverá negativos.
- **FROM_DAYS(días)**: Se puede usar en combinación con la anterior, para obtener una fecha en formato yyy-mm-dd a partir del número de días que le estamos pasando. Por ejemplo:
    - `SELECT FROM_DAYS(`23577`);` nos devolvería '0064-07-20'
    - Juan Paz detectó que esta función sólo devuelve 0000-00-00 si el número de días es inferior a 365.
- **DATE_ADD(fecha, intervalo)**: Permite incrementar a una fecha un intervalo dado en días, semanas, meses, años,... Por ejemplo:
    - `SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);` nos devolvería '2008-02-02'
    - `SELECT DATE_ADD('2008-01-02', INTERVAL 6 MONTH);` nos devolvería '2008-07-02'
- **DATE_ADD(fecha, intervalo)**: Permite incrementar a una fecha un intervalo dado en días, semanas, meses, años,... Por ejemplo:
    - `SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);` nos devolvería '2008-02-02'
    - `SELECT DATE_ADD('2008-01-02', INTERVAL 6 MONTH);` nos devolvería '2008-07-02'
- **DATE_ADD(fecha, intervalo)**: Permite incrementar a una fecha un intervalo dado en días, semanas, meses, años,... Por ejemplo:
    - `SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);` nos devolvería '2008-02-02'
    - `SELECT DATE_ADD('2008-01-02', INTERVAL 6 MONTH);` nos devolvería '2008-07-02'
- **DATE_SUB(fecha, intervalo)**: Permite decrementar a una fecha un intervalo dado en días, semanas, meses, años,... Por ejemplo:
    - `SELECT DATE_SUB('2008-02-02', INTERVAL 31 DAY);` nos devolvería '2008-01-02'

7.3 Diferentes formatos para las fechas en MySQL 5.1

Especificador – Descripción

%a - Abbreviated weekday name (Sun..Sat)

%b - Abbreviated month name (Jan..Dec)

%c - Month, numeric (0..12)

%D - Day of the month with English suffix (0th, 1st, 2nd, 3rd, …)

%d - Day of the month, numeric (00..31)

%e - Day of the month, numeric (0..31)

%f - Microseconds (000000..999999)

%H - Hour (00..23)

%h - Hour (01..12)

%I - Hour (01..12)

%i - Minutes, numeric (00..59)

%j - Day of year (001..366)

%k - Hour (0..23)

%l - Hour (1..12)

%M - Month name (January..December)

%m - Month, numeric (00..12)

%p - AM or PM

%r - Time, 12-hour (hh:mm:ss followed by AM or PM)

%S - Seconds (00..59)

%s - Seconds (00..59)

%T - Time, 24-hour (hh:mm:ss)

%U - Week (00..53), where Sunday is the first day of the week

%u - Week (00..53), where Monday is the first day of the week

%V - Week (01..53), where Sunday is the first day of the week; used with %X

%v - Week (01..53), where Monday is the first day of the week; used with %x

%W - Weekday name (Sunday..Saturday)

%w - Day of the week (0=Sunday..6=Saturday)

%X - Year for the week where Sunday is the first day of the week, numeric, four digits;

used with %V

%x - Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v

%Y - Year, numeric, four digits

%y - Year, numeric (two digits)

%% - A literal “%” character

%x - x, for any “x” not listed above

Capítulo 8 – Funciones de columnas

Permiten obtener un sólo valor como resultado de aplicar una determinada operación a los valores contenidos en una columna (suma de todos ellos, la media,...)

Funciones:

- **AVG**: calcula la media de los valores. OJO con los nulos, no incluirá para calcular aquellas tuplas.

  `SELECT AVG(salar) FROM temple;` nos devolvería 302.941176470588, que es la media de los salarios de los empleados. OJO con los nulos, no incluirá para calcular el resultado aquellas tuplas que tengan valores a nulo, es decir, dará lo mismo ejecutar:

  ```
  SELECT AVG(comis) FROM temple;
  ```

  que

  ```
  SELECT AVG(comis) FROM temple WHERE comis IS NOT NULL;
  ```

- **MAX**: halla el valor máximo.

  `SELECT MAX(salar) FROM temple;` nos devolvería 720, que es el máximo salario de los empleados.

- **MIN**: halla el valor mínimo.

  `SELECT MIN(salar) FROM temple;` nos devolvería 100, que es el mínimo salario de los empleados.

- **SUM**: calcula la suma.

  `SELECT SUM(salar) FROM temple;` nos devolvería 10300, que es la suma de todos los salarios de todos los empleados.

- **COUNT**: cuenta el número de ocurrencias. También se puede utilizar `COUNT(*)`.

  `SELECT COUNT(salar) FROM temple;` nos devolvería 34, que es el número de empleados que tenemos.

Recordamos que estas funciones de columnas se pueden utilizar tanto en la cláusula `SELECT` como en la cláusula `WHERE`, aunque si lo queremos usar en la cláusula `WHERE` deberemos utilizar - en la mayoría de los casos - una subselect.

También indicar que dentro de cada una de las funciones se pueden utilizar expresiones:

```
SUM (3*A)

SUM ((A+B)/2)

SUM (A+MONTH(CURDATE())
```

Hay que tener una serie de consideraciones a la hora de utilizar las funciones colectivas:

- Tener cuidado con los **NULOS** existentes entre la colección de valores. Hemos visto que en algunos casos (`AVG`), puede darnos resultados equivocados.
- Los argumentos para `AVG` y `SUM` deben de ser numéricos, mientras que para `MAX` y `MIN` pueden ser de cualquier tipo.

Capítulo 9 – Consultas con agrupamiento de filas

SQL permite agrupar los resultados por grupos de acuerdo con un determinado criterio. Las cláusulas existentes son:

- **GROUP BY**: cláusula opcional de la sentencia `SELECT` que permite agrupar filas. Se coloca siempre detrás del `WHERE`, en caso de que exista, o detrás del `FROM`, en caso de que el `WHERE` no exista. Su formato es:

  ```
  GROUP BY col1 [, col2, col3,...]
  ```

  Es importante el orden de las columnas a la hora de mostrar los resultados pero no influye a la hora de calcularlos, es decir, si hacemos:

  ```
  SELECT col1, col2, COUNT(*)
  ...
  GROUP BY col1, col2
  ```

  El resultado de las sumas será el mismo que si hacemos:

  ```
  SELECT col2, col1, COUNT(*)
  ...
  GROUP BY col2, col1
  ```

  Simplemente modificará la colocación de los valores.

  Los valores NULL también se incluyen en el mismo grupo a la hora de agrupar.

  OJO, si ponemos en la `SELECT` una columna que no fuera de agrupamiento habiendo utilizado `GROUP BY`, el resultado sería erróneo. No nos mostraría un mensaje de error pero sí que mostraría resultados indeterminados. por ejemplo:

  ```
  SELECT numde, AVG(salar), MIN(salar), MAX(salar)
  FROM temple
  GROUP BY numde;
  ```

  Sería correcto porque numde no hace referencia a una función de grupo pero aparece en el `GROUP BY` pero, sin embargo, daría error si ponemos esto:

  ```
  SELECT numde, salar, AVG(salar), MIN(salar), MAX(salar)
  FROM temple
  GROUP BY numde;
  ```

- **HAVING**: cláusula opcional al `GROUP` que funciona de manera muy similar al `WHERE`. Permite descartar **grupos de filas** que no cumplan con la condición/es indicada. Después de haber separado las filas en uno o varios grupos, se

descartarán aquellos que no satisfagan la condición. La condición deberá ser un predicado: simple o compuesto.

```
SELECT numhim, COUNT(*)
FROM temple
GROUP BY numhi
HAVING COUNT(*) > 1
```

En la cláusula `HAVING` deberán ir todas aquellas condiciones que tengan que ver con los valores después de haber sido agrupados. En otro caso deberán ir en el `WHERE`.

Si en la misma consulta aparecen una cláusula `WHERE` y una cláusula `HAVING`, primero se aplica el predicado de la cláusula `WHERE`. Las tuplas que satisfagan el predicado `WHERE` son colocadas en grupos por la cláusula `GROUP BY`. Después se aplica la cláusula `HAVING` a cada grupo, es decir, si algún grupo no cumple con el filtro de esa sentencia será eliminado (el grupo completo) del conjunto de resultados.

Capítulo 10 – Sentencias para modificar datos

La sentencia SELECT permite realizar consultas de los datos contenidos en las tablas pero SQL proporciona además unas sentencias para modificar los datos.

- **INSERT**: permite añadir una o más filas completas a una tabla. Se puede especificar de dos formatos diferentes:
  - Formato 1

    ```
    INSERT INTO tabla [(col1, col2,...)] VALUES (valor1, valor2,...)
    ```

    Si se especifican columnas no es necesario ponerlas todas ni tampoco ponerlas en el orden que fueron definidas. Sin embargo, si no se especifican, han de ponerse todas y en el orden que fueron creadas. Debe haber tantos valores como los especificados en la lista de columnas. También se pueden insertar varias filas de una sola vez, separando el conjunto de valores por comas.

    ```
    INSERT INTO tabla [(col1, col2,...)]
    VALUES (valor01, valor02,...),(valor11, valor12,...),(valor21, valor22,...),...
    ```

  - Formato 2

    ```
    INSERT INTO tabla [(col1, col2,...)] subselect
    ```

    Permite insertar ninguna, una o varias filas de una vez. El número de columnas del resultado de la subselect debe ser igual al de nombres en la lista de columnas. Los tipos de datos también han de ser equivalentes. Debe haber tantos valores como los especificados en la lista de columnas.
- **DELETE**: permite eliminar filas de una tabla. A diferencia de `DROP`, `DELETE` tan sólo elimina los datos y no la estructura. El formato es el siguiente:
  - `DELETE FROM tabla [WHERE predicado]`

    El predicado puede contener sentencias subordinadas (subselect), pero hay que tener cuidado con lo siguiente:

    ```
    DELETE FROM tabla WHERE valores IN (SELECT * FROM tabla);
    ```

    Estamos leyendo datos de una tabla y tratando de, al mismo tiempo, eliminarlos en la misma tabla. Eso nos va a dar un error. Hemos de procurar que sean tablas diferentes:

    ```
    DELETE FROM tabla1 WHERE valores IN (SELECT * FROM
    ```

```
tabla2);
```

Si se omite la cláusula `WHERE` se eliminarán todas las filas.

- **UPDATE**: permite modificar o actualizar varias filas de una tabla. Puede ser una, varias o todas las columnas de la tabla las que se vean afectadas por la modificación. El formato es el siguiente:
  - `UPDATE tabla SET columna1=valor1, [columna2=valor2,...] [WHERE predicado]`

    Al igual que en el caso del DELETE, el predicado puede contener sentencias subordinadas (subselect), pero hay que tener cuidado con lo siguiente:

    ```
    UPDATE tabla columna1=valor1 WHERE valores IN (SELECT * FROM tabla);
    ```

    Lo mismo, estamos leyendo datos de una tabla y tratando de, al mismo tiempo, modificarlos en la misma tabla. Hemos de procurar que sean tablas diferentes:

    ```
    UPDATE tabla1 columna1=valor1 WHERE valores IN (SELECT * FROM tabla2);
    ```

    Si se omite la cláusula WHERE se modificarán todas las filas.

Capítulo 11 – Consultas sobre varias tablas

Hasta ahora las sentencias SELECT que hemos visto sólo extraían datos de una sola tabla. Sin embargo, es posible manejar varias tablas en una sola sentencia. Y se puede hacer de diversas maneras:

- **Cláusula From**: se pueden especificar una lista de nombres de tablas separadas por comas. Por ejemplo:

  ```
  SELECT * FROM tcentr, tdepto;
  ```

  Nos devolvería un producto cartesiano, es decir, tantas filas como el resultado de multiplicar las filas de TCENTR por las filas de TDEPTO y tantas columnas como el resultado de sumar las columnas de TCENTR con las columnas de TDEPT. Se estarían formando todas las combinaciones posibles.

  Si nos damos cuenta surge un problema: los resultados duplicados. Al hacernos un producto cartesiano está realizando una combinación de cada uno de los elementos de una tabla por cada uno de los elementos de la otra tabla. Para eliminar los duplicados deberemos utilizar un JOIN (lo que en álgebra relacional se conoce como combinación), que no es más que añadir una restricción en el predicado usando la FK. Veamos:

  ```
  SELECT *
  FROM tcentr, tdepto
  WHERE numce = numce;
  ```

  Ahora pasamos de tener 18 filas (2x9) a tener sólo 9. Lo que acabamos de hacer es mostrar los datos correspondientes al centro para cada departamento, sin duplicados.

  Es normal que, si intentamos ejecutar la consulta anterior, nos dé un error. Estamos haciendo `numce = numce` y el compilador no sabe a qué tabla nos estamos refiriendo en cada caso. Hay que tener en cuenta lo que le llaman "**calificación de nombres**", es decir, si interviene más de una tabla en la misma sentencia SQL puede ocurrir que algún nombre de columna se repita en más de una de ellas. Por eso tenemos que **indicar a qué tabla pertenece**. Por ejemplo

  ```
  SELECT tdepto.numde, tcentr.numce, tdepto.numce
  FROM tcentr, tdepto;
  ```

  Aunque sería más sencillo si utilizamos **nombres locales**, o lo que también se conoce como **variables de tupla**. Por ejemplo:

  ```
  SELECT D.numde, C.numce, D.numce
  FROM tcentr C, tdepto D;
  ```

- **Consultas correlacionadas**: son consultas que no hacen referencia a columnas de tablas de su cláusula FROM. Por ejemplo:
    - *Obtener por orden alfabético los nombres de los departamentos cuyo presupuesto es inferior a la mitad de la suma de los salarios anuales de sus empleados.*

```
SELECT nomde
FROM tdepto
WHERE presu < (SELECT (SUM(salar*12)/1000)/2
               FROM temple WHERE numde = tdepto.numde);
```

Anexo – Utilidades

A continuación describo una serie de funciones o aplicaciones de utilidad para el seguimiento de los temas y para el desarrollo de ejercicios relacionados con los mismos.

- **IFNULL**(expresión,valor): función que evalúa la expresión y, si el resultado es un NULO, lo sustituye por el valor que le indiquemos. Por ejemplo:
  - `IFNULL(15/0,0)` -> Devolverá un cero.
- **LOCATE**(carácter o cadena a buscar,cadena): función que devuelve la ÚLTIMA posición del carácter o de la subcadena que queremos buscar en la cadena que le indicamos. Si no la encuentra devolverá cero. Por ejemplo:
  - `SELECT LOCATE('bar', 'foobarbar');` -> 7
- **POSITION**(carácter o cadena a buscar IN cadena): funciona exactamente igual que LOCATE, lo único que cambia es la sintaxis.
- **FORMAT**(valor): Formatea el valor numérico que le pasemos al número de cifras decimales que le indiquemos:
  - `FORMAT(132.2,0)` -> Devolverá 132.
  - OJO: si le pasamos un número negativo también nos devuelve el suelo, es decir, `FLOOR(-1.23)` -> Devolverá un -2, no un -1.
- **FLOOR**(valor): nos devuelve el "suelo" entero del valor que le pasemos, es decir:
  - `FLOOR(1.23)` -> Devolverá un 1.
  - OJO: si le pasamos un número negativo también nos devuelve el suelo, es decir, `FLOOR(-1.23)` -> Devolverá un -2, no un -1.
- **Alias de una columna**: a veces es necesario/interesante indicar el nombre que queremos que tomen las columnas, sobre todo si hacemos expresiones complejas. Lo único que hay que hacer es utilizar un alias. Por ejemplo:
  - `SELECT columna AS 'Mi columna'` -> Nos devolverá los resultados encabezados por una columna de título *Mi columna*. Funciona con **AS** y sin poner **AS**.

Bibliografía

- Documentación:
  - J. Benavides Abajo; J. M. Olaizola Bartolomé; E. Rivero Cornelio. *SQL: Para usuarios y programadores*. Tercera Edición. Madrid: Paraninfo, 1997. ISBN: 84-283-1821-2.
  - MySQL 5.1 Reference Manual [en línea]. Oracle. Disponible en web: <http://dev.mysql.com/doc/refman/5.1/en/>
  - MySQL 5.0 Reference Manual [en línea]. Oracle. Disponible en web: <http://dev.mysql.com/doc/refman/5.0/es/>
- Programas:
  - *DBDesigner 4*. Fabforce. Disponible en web:

    <http://www.fabforce.net/dbdesigner4/>
  - *MySQL Workbench 5*. MySQL Inc. Disponible en web:

    <http://wb.mysql.com/>
  - *phpMyAdmin 3*. SourceForge. Disponible en web:

    <http://www.phpmyadmin.net/home_page/index.php>
  - K. 'Oswald' Seidler. *XAMPP*. Apache Friends. Disponible en web:

    <http://www.apachefriends.org/es/xampp.html>