

TEMA - III:

DISEÑO FÍSICO RELACIONAL

BD - (1º DAM)

ÍNDICE

1. INTRODUCCIÓN	4
1.1. FUNCIONES DEL SQL	4
1.2. CARACTERÍSTICAS DEL SQL	4
1.3. HISTORIA DEL SQL	4
1.4. FORMAS DE USAR EL SQL	5
1.5. ELEMENTOS DE SQL	5
1.5.1. IDENTIFICADORES	5
1.5.2. PALABRAS RESERVADAS	5
1.6. MySQL	5
1.6.1. EVOLUCIÓN DE LAS VERSIONES DE MySQL	5
1.6.2. TUTORIAL BÁSICO	6
1.6.2.1. Conectarse al servidor mediante el programa cliente en modo consola <i>mysql</i>	6
1.6.2.2. Algunas consultas básicas de interés para el administrador	7
1.6.2.3. Añadir password al usuario root	8
1.6.2.4. Crear, usar y acceder a bases de datos en MySQL	9
1.6.2.5. El comando SHOW y el comando DESC	11
1.6.2.6. Editar las sentencias desde el cliente en modo consola <i>mysql</i>	12
1.6.2.7. Lanzar <i>scripts</i> desde el cliente en modo consola <i>mysql</i>	13
1.6.2.8. Verificar el estado, arrancar y detener el proceso servidor	14
2. TIPOS DE DATOS DE MySQL	16
2.1. Tipos numéricos:	16
2.1. Tipos fecha:	17
2.2. Tipos cadena:	17
3. Lenguaje de Definición de Datos (LDD)	19
3.1. CREACIÓN de TABLAS	19
3.1.1. CONSIDERACIONES PREVIAS A LA CREACIÓN DE UNA TABLA	19
3.1.2. CREACIÓN DE TABLAS CON LA DEFINICIÓN DE RESTRICCIONES	20
3.1.2.1. RESTRICCIONES DEFINIDAS A NIVEL DE COLUMNA	20
3.1.2.2. EJEMPLOS de RESTRICCIONES a Nivel de COLUMNA	21
3.1.2.3. RESTRICCIONES DEFINIDAS A NIVEL DE TABLA	22
3.1.2.4. EJEMPLOS de RESTRICCIONES a Nivel de TABLA	22
3.1.3. INTEGRIDAD REFERENCIAL	23
3.1.4. FORMATO COMPLETO DE LA CREACIÓN DE TABLAS	27
3.1.5. CREACIÓN DE UNA TABLA A PARTIR DE OTRA	28
3.2. MODIFICACIÓN de TABLAS	29
3.2.1. EJEMPLOS DE MODIFICACIÓN DE TABLAS	30
3.3. BORRADO de TABLAS	31
3.3.1. EJEMPLOS DE BORRADO DE UNA TABLA	31
3.4. RENOMBRADO	31
4. PRÁCTICAS	32
4.1. Práctica - A	32
4.2. Práctica - B	34
4.3. Práctica - C	35
5. SOLUCIONES	37
5.1. Práctica - B	37

1. INTRODUCCIÓN

Antes de abordar esta unidad hay que tener en cuenta:

1º Se trata de una guía para que sirva de referencia o de consulta cuando se necesite a lo largo del curso. Posteriormente haremos ejemplos utilizando los elementos estudiados en este tema. Permitirá irse familiarizando con el vocabulario y la sintaxis de las sentencias que se explicarán posteriormente

2º En esta unidad se abordan cuestiones que, aunque están definidas por el estándar ANSI/ISO SQL, no están asumidas al 100% por todos los fabricantes. Por tanto, pueden existir ligeras diferencias de algunos productos con algunas de las especificaciones que aquí se exponen. Se ha escogido el gestor de bases de datos MySQL para realizar los ejemplos y los ejercicios. Aunque como este es un curso de SQL, no se profundizará en las particularidades de este SGBD hasta los temas de Administración (temas 10, 11 y 12 del curso) que si son específicos de MySQL.

1.1. FUNCIONES DEL SQL

- . Definir objetivos que van a contener los datos.
- . Añadir, editar y eliminar datos de esos objetos.
- . Controlar el acceso a los datos.
- . Compartir datos.
- . Garantizar la integridad de los datos.
- . Sirve para comunicarse con el SGBD.

1.2. CARACTERÍSTICAS DEL SQL

- . No tiene sentencias de Bucles, ni la condición if.
- . Tiene pocas sentencias.
- . No es muy estructurado.
- . Es un lenguaje de programación de Bases de Datos.
- . Es un lenguaje para la administración de la Base de Datos.
- . Funciona en estructuras cliente/servidor.
- . Puede trabajar en Bases de Datos Distribuidas.

1.3. HISTORIA DEL SQL

- CODD en los 70 dio ideas de las bases de datos relacionales en plan teórico, pero no se le olvidó que era necesario que existiese un lenguaje para trabajar con esos datos.
- El primer boceto se llamo SEQUEL (diseñado por IBM) (entre el 70 y el 75)
- En el 77 sale una segunda versión que se llama SEQUEL / 2 (una mejora del SEQUEL) por exigencias legales de aquellos tiempos, se tuvo que cambiar el nombre → Pasó a llamarse SQL
- El SQL corría sobre un S.O. de IBM (el SYSTEM R)
- Por estas fechas, ORACLE y otros fabricantes ya estaban trabajando sobre lo mismo (diseñar un lenguaje para bases de datos)
- El siguiente paso de IBM fue mejorar el SYSTEM R y cambiarle el nombre (SYSTEM R pasó a llamarse DB2)
- En el 87 se produce la estandarización.
- Se empieza a usar dentro de lenguajes de 3ª generación. (C, PASCAL,...)

1.4. FORMAS DE USAR EL SQL

- **Interactiva:** Se accede desde un terminal al SGBD por medio del SQL (y por tanto a la base de datos)
- **PL/SQL:** El SQL original + instrucciones usadas para crear bucles y poner condiciones (IF, WHILE,...). Con PL/SQL se crea un lenguaje de programación que permite hacer un programa íntegro en SQL.

Características:

Se pueden crear variables y constantes.

Dispone de mecanismos de control de errores.

Funcionamiento: Cada instrucción pasa a un analizador. Este analizador distingue si es una orden SQL o no. Si es una orden de SQL (SELECT, REVOKE,...) lo manda a un intérprete de SQL. Si es una extensión de SQL lo manda a un módulo de ejecución de instrucciones procedurales. Puede trabajar en herramientas CASE y en lenguajes de 4^a generación.

- Desde un programa hecho en lenguaje de 3^a generación (C, PASCAL, ...)

1.5. ELEMENTOS DE SQL

1.5.1. IDENTIFICADORES

Es la forma de dar **nombres a los objetos** de la base de datos y a la propia base de datos. Un objeto tendrá un nombre que lo identifique de forma **única** dentro de su base de datos.

El estándar define que pueden tener hasta **18 caracteres** empezando con un carácter **alfabético** y continuando con caracteres numéricos y/o alfabéticos.

En la práctica este estándar se ha ampliado y MySQL permite nombres de identificadores de **hasta 64 caracteres** sin espacios en blanco. Para los nombres de las bases de datos y de las tablas no están permitidos '/', '\' ni '.'.

1.5.2. PALABRAS RESERVADAS

Al igual que en el resto de los lenguajes de programación, existen palabras que tiene un significado especial para el gestor de la base de datos y no pueden ser utilizadas como identificadores.

Serán todas las palabras que irán apareciendo en los formatos de las instrucciones, más algunas que se verán en este curso.

1.6. MySQL.

1.6.1. EVOLUCIÓN DE LAS VERSIONES DE MySQL.

Feature	MySQL Series
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for MyISAM tables)
Stored procedures	5.0
Views	5.0
Cursors	5.0
Foreign keys	5.1 (implemented in 3.23 for InnoDB)
Triggers	5.0 and 5.1
Full outer join	5.1
Constraints	5.1

1.6.2. TUTORIAL BÁSICO

1.6.2.1. Conectarse al servidor mediante el programa cliente en modo consola mysql

El cliente básico de MySQL es el **programa mysql** (mysql.exe en Windows) que viene en la distribución y una vez instalado se encuentra en la carpeta /bin que genera la instalación

Para conectarse al servidor, usualmente necesitamos de un nombre de **usuario** (login) y de una **contraseña** (password), y si el servidor al que nos deseamos conectar está en una máquina diferente de la nuestra, también necesitamos indicar el nombre o la **dirección IP** de dicho servidor. Una vez que conocemos estos tres valores, podemos conectarnos de la siguiente manera:

```
shell> mysql -h NombreDelServidor -u NombreDeUsuario -p
```

Cuando ejecutamos este comando, se nos pedirá que proporcionemos también la contraseña para el nombre de usuario que estamos usando.

Si la conexión al servidor MySQL se pudo establecer de manera satisfactoria, recibiremos el mensaje de bienvenida y estaremos en el prompt de mysql:

```
shell>mysql -h 192.168.1.2 -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.Your MySQL connection
id is 5563 to server version: 4.1.8-nt-max-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Este prompt **mysql>** nos indica que hemos conectado y mysql está listo para recibir comandos.

Por defecto la instalación MySQL incorpora dos usuarios el usuario **Administrador ('root')** y el usuario invitado ('ODBC' y también '') ambos con privilegios de administración cuando se conectan de modo local al servidor. Por lo tanto es posible acceder con derechos de administrador local del propio equipo sin necesidad de introducir ningún tipo de opción y sin password.

```
shell> mysql
```

Después de que nos hemos conectado de manera satisfactoria, podemos desconectarnos en cualquier momento al escribir "quit" o "exit".

Esta forma de acceso como administrador sin password por defecto es útil y cómoda para esta fase de aprendizaje pero altamente insegura y desaconsejable en una instalación de producción convencional.

En la mayoría de los ejemplos siguientes se asume que estamos conectados al servidor, lo cual se indica con el prompt de mysql. (**mysql>**)

Ejemplos de establecimiento de conexión entre cliente y servidor:

Conectarse localmente con el usuario invitado. Normalmente se accede con privilegios de administrador

```
C:\MySQL\bin>mysql
```

Conectarse localmente con el usuario administrador root introduciendo la password micontraseña en la propia línea de comandos.

```
C:\MySQL\bin>mysql -uroot -pmicontraseña
```

Conectarse localmente con el usuario administrador root sin introducir la password en la propia línea de comandos. La password se introduce después de pulsar Enter

```
C:\MySQL410\bin>mysql -uroot -p
Enter password: *****
```

Conectarse a un servidor de otra maquina con el usuario invitado sin password. En este caso no se accede con privilegios de administrador. Lo normal (depende de la distribución) es que solo tenga acceso a la base de datos test que viene incluida para pruebas

```
C:\MySQL410\bin>mysql -h192.168.3.33
```

En el tema 11 de esta documentación se trata en profundidad el tema de los usuarios de MySQL y sus privilegios de acceso.

1.6.2.2. Algunas consultas básicas de interés para el administrador

En este momento debimos de haber podido conectarnos ya al servidor MySQL, aún cuando no hemos seleccionado alguna base de datos para trabajar. Lo que haremos a continuación es escribir algunos comandos para irnos familiarizando con el funcionamiento de mysql.

- Consulta para conocer la versión del servidor y la fecha actual:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 4.1.8-nt-max-log | 2005-01-08 |
+-----+-----+
1 row in set (0.33 sec)

mysql>
```

Este comando ilustra distintas cosas acerca de mysql:

- Un comando normalmente consiste de una sentencia SQL seguida por un punto y coma. Cuando emitimos un comando, mysql lo manda al servidor para que lo ejecute, nos muestra los resultados y regresa el prompt indicando que está listo para recibir más consultas.
- mysql muestra los resultados de la consulta como una tabla (filas y columnas). La primera fila contiene etiquetas para las columnas. Las filas siguientes muestran los resultados de la consulta. Normalmente las etiquetas de las columnas son los nombres de los campos de las tablas que estamos usando en alguna consulta. Si lo que estamos recuperando es el valor de una expresión (como en el ejemplo anterior) las etiquetas en las columnas son la expresión en sí.
- mysql muestra cuántas filas fueron regresadas y cuanto tiempo tardó en ejecutarse la consulta, lo cual puede darnos una idea de la eficiencia del servidor, aunque estos valores pueden ser un tanto imprecisos ya que no se muestra la hora del CPU, y porque pueden verse afectados por otros factores, tales como la carga del servidor y la velocidad de comunicación en una red. Las palabras clave pueden ser escritas usando mayúsculas y minúsculas.

Aquí está un ejemplo que muestra una consulta simple escrita en varias líneas.

-Consulta para conocer el nombre del usuario que ha efectuado la conexión:

```
mysql> SELECT
-> USER();
+-----+
| user()      |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

mysql>
```

En este ejemplo debe notarse como cambia el prompt (de mysql> a ->) cuando se escribe una **consulta en varias líneas**. Esta es la manera en cómo mysql indica que está esperando a que finalice la consulta. Sin embargo si deseamos no terminar de escribir la consulta, podemos hacerlo al escribir \c como se muestra en el siguiente ejemplo:

```
mysql> SELECT
-> USER(),
-> \c
mysql>
```

De nuevo, se nos regresa el comando el prompt mysql> que nos indica que mysql está listo para una nueva consulta.

En la siguiente tabla se muestran cada uno de los prompts que podemos obtener y una breve descripción de su significado para mysql:

<u>Prompt</u>	<u>Significado</u>
mysql>	Listo para una nueva consulta.
->	Esperando la línea siguiente de una consulta multi-línea.
'>	Esperando la siguiente línea para completar una cadena que comienza con una comilla sencilla (').
">	Esperando la siguiente línea para completar una cadena que comienza con una comilla doble (").

Los prompts '>' y ">" ocurren durante la escritura de cadenas. En mysql podemos escribir cadenas utilizando **comillas sencillas o comillas dobles** (por ejemplo, 'hola' y "hola"), y mysql nos permite escribir cadenas que ocupen múltiples líneas. De manera que cuando veamos el prompt '>' o ">", mysql nos indica que hemos empezado a escribir una cadena, pero no la hemos finalizado con la comilla correspondiente.

1.6.2.3. Añadir password al usuario root

MySQL por defecto viene con la cuenta de administrador sin password. Realmente MySQL incluye dos cuentas de root (administrador) una para conectarse localmente y otra para conectarse desde cualquier máquina externa al servidor. Eso lo podemos ver haciendo el siguiente select sobre la **tabla de usuarios** que se encuentra en la base de datos mysql (diccionario de datos del servidor).

```
mysql> select user,host,password from mysql.user;
+-----+-----+-----+
| user | host      | password |
+-----+-----+-----+
| root | localhost |          |
| root | %         |          |
|      | localhost |          |
|      | %         |          |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

En este select se ve claramente como las cuentas de root y de invitado están sin password, por lo tanto el sistema es totalmente inseguro, solo válido para pruebas.

Lo anterior nos indica que la primera vez que se inicialice el servidor la primera cosa que se debe hacer es **especificar un password para el usuario root** (esto es para el administrador). Esto se puede realizar de la siguiente manera:

```
shell> mysql -u root
mysql> SET PASSWORD FOR root=PASSWORD('new_password');
```

Con esta sentencia le hemos añadido un password a la cuenta de root que sirve para conectar desde cualquier máquina.

Si también le queremos añadir un password a la cuenta de root para conectar en modo localhost ejecutaremos la siguiente instrucción:

```
mysql> SET PASSWORD FOR root@localhost=PASSWORD('new_password');
```

Volvemos a repetir la select sobre la tabla usuarios para ver como ha cambiado después de ejecutar las sentencias anteriores.

```
mysql> select user,host,password from mysql.user;
+-----+-----+-----+
| user | host   | password          |
+-----+-----+-----+
| root | localhost | *8C6D142DE6424E095EC1E1E756974E907289CE6B |
| root | %        | *8C6D142DE6424E095EC1E1E756974E907289CE6B |
|      | localhost |                               |
|      | %        |                               |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

Nota1: **SET PASSWORD FOR** se puede usar para establecer nueva contraseña para cualquier usuario, no solo root.

Nota 2: La función **PASSWORD('new_password')** genera la password encriptada.

1.6.2.4. Crear, usar y acceder a bases de datos en MySQL

Ahora que conocemos como escribir y ejecutar sentencias, es tiempo de crear y acceder a una base de datos.

Esta sección muestra como ver las bases de datos a las que tenemos acceso en el servidor, crear una base de datos nueva , posicionar al usuario en una determinada base de datos y acceder a tablas de una base de datos cuando estamos situados en otra base de datos.

Primeramente usaremos la sentencia **SHOW DATABASES** para ver cuáles son las bases de datos existentes en el servidor al que estamos conectados:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
2 rows in set (0.00 sec)

mysql>
```

Es probable que la lista de bases de datos que veamos sea diferente en nuestro caso, pero seguramente las bases de datos "mysql" y "test " estarán entre ellas. En particular, la base de datos "**mysql**" es necesaria, ya que ésta tiene la información de los privilegios de los usuarios de MySQL. La base de datos "test " se crea durante la instalación de MySQL con el propósito de servir como área de trabajo para los usuarios que inician en el aprendizaje de MySQL.

Para acceder a la base de datos "test" se usa el comando **USE**:

```
mysql> USE test
Database changed
mysql>
```

Observar que USE, al igual que QUIT, no requieren el uso del punto y coma, aunque si se usa éste, no hay ningún problema. El comando USE es especial también de otra manera: debe ser usado en una sola línea.

```
mysql> USE prueba
ERROR 1049: Unknown database 'prueba'
mysql>
```

El mensaje anterior indica que la base de datos no ha sido creada, por lo tanto necesitamos crearla.

```
mysql> CREATE DATABASE prueba;
Query OK, 1 row affected (0.00 sec)

mysql> USE prueba
Database changed
mysql>
```

Ahora ya se pueden lanzar sentencias **CREATE TABLE** para crear tablas sobre la base de datos prueba que hemos creado y hemos accedido.

Al crear una base de datos no se selecciona ésta de manera automática; debemos hacerlo de manera explícita, por ello usamos el comando **USE** en el ejemplo anterior.

La base de datos se crea sólo una vez, pero nosotros debemos seleccionarla cada vez que iniciamos una sesión con mysql. Por ello es recomendable que se indique la base de datos sobre la que vamos a trabajar al momento de invocar al monitor de MySQL. Por ejemplo:

```
shell>mysql -h 192.168.1.2 -u root -p prueba
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 17 to server version: 4.1.8-nt-max-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer
mysql>
```

Observar que "prueba" no es la contraseña que se está proporcionando desde la línea de comandos, sino el nombre de la base de datos a la que deseamos acceder. Si deseamos proporcionar la contraseña en la línea de comandos después de la opción "-p", debemos de hacerlo sin dejar espacios (por ejemplo, -phola123, no como -p hola123). Sin embargo, escribir nuestra contraseña desde la línea de comandos no es recomendado, ya que es bastante **inseguro**.

Para ver en un momento dado en que base de datos nos encontramos situados podemos lanzar la siguiente consulta:

```
mysql> select database();
+-----+
| database() |
+-----+
| prueba     |
+-----+
1 row in set (0.00 sec)

mysql>
```

Para acceder a una tabla de una base de datos cuando nos encontramos situados en otra base de datos es necesario anteponer en la sentencia el **nombre de la base de datos seguido de un punto**. Por ejemplo accedo mediante el comando *USE* a la base de datos *test* y a continuación hago un select de la tabla *departamentos* de la base de datos prueba.

```
mysql> use test
Database changed
mysql> select * from prueba.departamentos;
+-----+-----+-----+
| DEP_NO | DNOMBRE      | LOCALIDAD   |
+-----+-----+-----+
|    10  | CONTABILIDAD  | BARCELONA  |
|    20  | INVESTIGACION | VALENCIA   |
|    30  | VENTAS        | MADRID     |
|    40  | PRODUCCION   | SEVILLA    |
+-----+-----+-----+
4 rows in set (0.10 sec)

mysql>
```

Esto se podrá hacer siempre que el usuario tenga privilegios suficientes como para acceder a ambas bases de datos (en este ejemplo a *prueba* y a *test*).

1.6.2.5. El comando SHOW y el comando DESC

Estos dos comandos son muy útiles cuando trabajamos con el cliente de MySQL en entorno de comandos puesto que no tenemos el soporte visual que proporcionan los clientes gráficos.

Para ver un listado con las tablas dentro de una determinada base de datos podemos lanzar la sentencia:

```
mysql> show tables;
+-----+
| Tables_in_prueba |
+-----+
| departamentos    |
| empleados        |
+-----+
2 rows in set (0.10 sec)
```

Para ver la descripción de una tabla en sus columnas y tipos asociados, además de poder ver los atributos que aceptan nulos y las opciones por defecto debemos usar el comando **DESC** o DESCRIBE que realmente es un sinónimo de la sentencia SHOW COLUMNS FROM.

```
mysql> desc departamentos;
+-----+-----+-----+-----+-----+-----+
| Field   | Type       | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| DEP_NO  | int(2)    | YES  | PRI  | 0       |       |
| DNOMBRE | varchar(14)| YES  |       | NULL    |       |
| LOCALIDAD | varchar(10)| YES  |       | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.05 sec)
```

Es muy interesante también poder ver la sentencia de creación de tabla asociada a una determinada tabla de la base de datos. Es una información que puede complementar la proporcionada por el comando DESC.

```
mysql> show create table departamentos \G
***** 1. row *****
Table: departamentos
Create Table: CREATE TABLE `departamentos`
(`DEP_NO` int(2) NOT NULL default '0',
`DNOMBRE` varchar(14) default NULL,
`LOCALIDAD` varchar(10) default NULL,
PRIMARY KEY (`DEP_NO`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

La opción \G permite visualizar la salida de la consulta en una sola fila.

1.6.2.6. Editar las sentencias desde el cliente en modo consola mysql

El programa cliente mysql no es muy amigable a la hora de editar sentencias largas de SQL antes de lanzarlas. Es por lo tanto conveniente y aconsejable editar dichas sentencias desde un sencillo **editor ASCII** (*Notepad o Wordpad en entorno Windows*) y una vez editadas lanzar las sentencias mediante el usual procedimiento de "copiar y pegar" al cliente mysql.

Con el siguiente ejemplo se muestra el proceso desde Windows (en Linux es muy similar).

Paso 1. Se abre una ventana de comandos y se inicia una sesión con el cliente mysql sobre la base de datos test.

```
C:\MySQL5\bin>mysql test
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 247 to server version: 5.0.2-alpha-nt-max-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Paso 2. Se abre en otra ventana el editor y se edita la sentencia SQL que se va a lanzar al cliente.

```
Archivo Edición Formato Ver Ayuda
SELECT DNOMBRE FROM DEPARTAMENTOS
WHERE DEP_NO > 10
ORDER BY DEP_NO DESC;
```

Paso 3. Se marca con el ratón el párrafo con la sentencia para su copia mediante el comando Menú→Edición→Copiar o bien pulsando Ctrl+C

```
Archivo Edición Formato Ver Ayuda
SELECT DNOMBRE FROM DEPARTAMENTOS
```

Paso 4. Se pega la copia anterior pulsando con el botón derecho del ratón sobre cualquier punto de la ventana de comandos y seleccionando la opción Pegar.

```
C:\MySQL5\bin>mysql test
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 247 to server version: 5.0.2-alpha-nt-max-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT DNOMBRE FROM DEPARTAMENTOS
-> WHERE DEP_NO > 10
-> ORDER BY DEP_NO DESC;
```

Paso 5. Se pulsa Enter para lanzar la sentencia.

```
C:\MySQL5\bin>mysql test
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 247 to server version: 5.0.2-alpha-nt-max-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT DNOMBRE FROM DEPARTAMENTOS
-> WHERE DEP_NO > 10
-> ORDER BY DEP_NO DESC;
+-----+
| DNOMBRE   |
+-----+
| PRODUCCION|
| VENTAS    |
| INVESTIGACION|
+-----+
3 rows in set (0.00 sec)
```

Este proceso, aunque se haya explicado en cinco pasos, una vez que el usuario se ha acostumbrado a la mecánica, es la manera más cómoda de trabajar cuando estamos en entorno de comandos y no disponemos de un cliente gráfico.

1.6.2.7. Lanzar scripts desde el cliente en modo consola mysql.

Normalmente hasta ahora hemos usado *mysql* de manera interactiva para ejecutar algunas consultas y ver los resultados. Sin embargo, es posible usar *mysql* en modo batch. Para hacer esto tenemos que poner los comandos que deseamos ejecutar dentro de un archivo, y entonces decirle a *mysql* que lea los comandos de dicho archivo:

```
shell> mysql < archivo-batch.sql
```

Si se usa *mysql* de esta manera, se está creando un pequeño script, y posteriormente se está ejecutando dicho script. La extensión **.sql** no es obligatoria pero si aconsejable para reconocer los scripts de sql. Al ejecutar las sentencias y comandos que se encuentran en el script, es posible que suceda algún error. Si se desea que se continúen ejecutando las demás sentencias, a pesar de que haya ocurrido un error, se tiene que usar la opción **--force**

```
shell> mysql --force < archivo-batch.sql
```

Así mismo, es posible especificar los parámetros de conexión desde la línea de comandos. Por ejemplo:

```
shell> mysql -h 192.168.2.1 -uroot -p < archivo-batch.sql
```

Algunas cuantas razones para usar scripts en lugar de trabajar en modo interactivo:

- Si se ejecutan un cierto número de consultas frecuentemente (cada día, o cada semana), al hacer un script nos evitamos tener que volver a teclear cada una de las consultas.
- Se pueden generar nuevas consultas que sean similares a las existentes al copiar y editar estos scripts.
- Al escribir consultas de varias líneas, los scripts ayudan bastante para que no se tengan que escribir todas las líneas nuevamente si se comete algún error.
- Se puede guardar la salida en un archivo para revisarla posteriormente.

```
shell> mysql < archivo-batch.sql > salida-del-script.txt
```

- Se pueden distribuir los scripts a otras personas para que puedan ejecutar también nuestros comandos y sentencias.
- En algunas situaciones no se permite el uso interactivo de mysql. (En el caso de las tareas programadas). En este caso, es indispensable usar el modo batch.

Cabe mencionar que el formato de la salida es diferente (más conciso) cuando se ejecuta mysql en modo batch, que cuando se usa de manera interactiva.

También se puede lanzar el script una vez conectados a mysql mediante el comando source (o el método abreviado \.)

```
mysql> source archivo-batch.sql
o el equivalente abreviado mysql> \. archivo-batch.sql
```

1.6.2.8. Verificar el estado, arrancar y detener el proceso servidor.

En la mayoría de las instalaciones de MySQL, en sus distintas versiones, se selecciona por defecto la opción de **arrancar automáticamente** el proceso servidor cuando se arranca el host sobre el que está instalado. Pero es fundamental para un administrador saber verificar, en un momento dado, el estado del servidor (en ejecución o detenido) y conocer perfectamente cual es el mecanismo de arranque y/o parada en el caso que sea necesario efectuarlo.

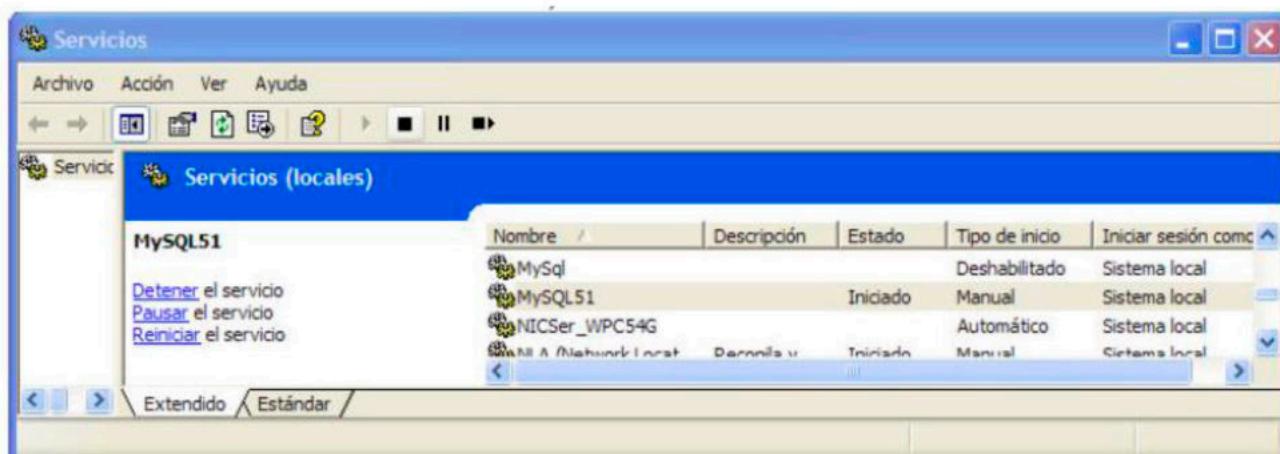
Nota: El proceso (*demonio*) servidor en MySQL se denomina *mysqld* o una de sus variantes *mysqld-opt, mysqld-nt, mysqld-max, mysqld-max-nt*.

Verificar el estado del servidor en Windows.

Para acceder a la ventana de servicios de Windows rápidamente

Menú Inicio → Ejecutar → services.msc

En la siguiente imagen se ve la ventana de servicios donde aparece el servicio asociado al servidor MySQL51 que está Iniciado y tiene un tipo de inicio Manual (el administrador debe iniciar el servicio)



Verificar el estado del servidor en Linux.

En Linux abrimos una terminal y pulsamos el comando

```
ps -A | grep mysqld
```

Si MySQL esta ejecutándose deberá responder con algo parecido a:

```
2752 ? 00:00:00:00 mysqld
```

Arrancar el proceso servidor en Windows.

Si tenemos el proceso servidor asociado a un servicio de Windows (lo más habitual) pulsamos botón derecho del ratón sobre el servicio que queremos iniciar y seleccionamos *Iniciar*.

En caso contrario lo mejor es iniciar el proceso desde una consola de comandos, es muy importante que en el proceso de arranque se indique mediante la opción *defaults-file* donde se encuentra el fichero de configuración. El comando será algo parecido a lo siguiente

```
C:\MySQL410\bin\mysqld-max-nt --defaults-file="C:\MySQL410\my.ini"
```

Arrancar el proceso servidor en Linux.

En Linux lo más habitual es tener automatizado el arranque del servidor de MySQL cuando se inicia el equipo. En el caso de que el servicio este detenido tendremos que arrancar el proceso de forma manual.

En Linux hay muchas formas de arrancar el servidor MySQL. Una de las formas más simples y seguras es arrancar el servidor a través del script de arranque *mysqld_safe* que suele encontrarse en la carpeta */usr/bin* en la mayoría de las distribuciones. Por lo tanto abrimos una terminal y pulsamos el comando */usr/bin/mysqld_safe &*

Detener el proceso servidor en Windows.

Si tenemos el proceso servidor asociado a un servicio de Windows (lo más habitual) pulsamos botón derecho del ratón sobre el servicio que queremos iniciar y seleccionamos *Detener*.

En caso contrario lo mejor es usar el programa *mysqladmin* desde la consola de comandos de Windows y añadirle la orden *shutdown* Ejemplo con usuario administrador de MySQL sin password

```
C:\> mysqladmin shutdown
```

Ejemplo con usuario administrador de MySQL con password

```
C:\>mysqladmin -uroot -ppassword shutdown
```

Detener el proceso servidor en Linux.

En Linux lo más cómodo es usar el programa *mysqladmin* exactamente igual como se ha comentado en el apartado anterior para detener el proceso servidor en Windows

2. TIPOS DE DATOS DE MySQL

Las columnas de la base de datos almacenan valores que tendrán diferentes valores en cada fila. Estos datos se definen indicando su nombre (*NombreColumna*) y el tipo de datos que almacenarán. La forma de almacenarlos no es la misma para todos, por lo tanto una parte importante de la definición de un dato es la especificación de su tipo. Al indicar el tipo de datos se suele indicar también el tamaño.

La cantidad de tipos de datos posibles es muy extensa, quedando la enumeración completa fuera de los objetivos de este curso. A continuación se indican algunos de los tipos de datos más utilizados suficientes para este curso (para mayor detalle consultar el manual).

2.1. Tipos numéricos:

Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

TinyInt: es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

Bit ó Bool: un número entero que puede ser 0 ó 1

SmallInt: número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

MediumInt: número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a 16777215.

Integer, Int: número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

BigInt: número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

Float: número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

xReal, Double: número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

Decimal, Dec, Numeric: Número en coma flotante desempaquetado. El número se almacena como una cadena.

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

2.1. Tipos fecha:

A la hora de almacenar fechas, hay que tener en cuenta que MySql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

Date: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

Datetime: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

TimeStamp: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

Time: almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

Year: almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Tamaño	Formato
14	AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDiaHoraMinutoSegundo aammmddhhmmss
8	AñoMesDia aaaammdd
6	AñoMesDia aammdd
4	AñoMes aamm
2	Año aa

2.2. Tipos cadena:

Char(n): almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

VarChar(n): almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object)

La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.

Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

TinyText y TinyBlob: Columna con una longitud máxima de 255 caracteres.

Blob y Text: un texto con un máximo de 65535 caracteres.

MediumBlob y MediumText: un texto con un máximo de 16.777.215 caracteres.

LongBlob y LongText: un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

Enum: campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos.

Set: un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

Diferencia de almacenamiento entre los tipos Char y VarChar

Valor	CHAR(4)	Almacenamiento	VARCHAR(4)	Almacenamiento
"	"	4 bytes	"	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

3. Lenguaje de Definición de Datos (LDD).

3.1. CREACIÓN de TABLAS

3.1.1. CONSIDERACIONES PREVIAS A LA CREACIÓN DE UNA TABLA

Antes de escribir la sentencia para crear una tabla debemos pensar una serie de datos y requisitos que va a ser necesario definir en la creación. Es importante pensar en la tabla que se quiere crear tomando las decisiones necesarias antes de escribir el formato de la creación.

Definición de la tabla

Por una parte unas consideraciones básicas como:

- El nombre de la tabla.
- El nombre de cada columna.
- El tipo de dato almacenado en cada columna.
- El tamaño de cada columna.

Restricciones en las tablas

Por otra parte, información sobre lo que se pueden almacenar las filas de la tabla. Esta información serán las restricciones que almacenamos en las tablas. Son una parte muy importante del modelo relacional pues nos permiten relacionar las tablas entre sí y poner restricciones a los valores de que pueden tomar los atributos (columnas) de estas tablas.

Restricciones, llamadas **CONSTRAINTS**, son condiciones que imponemos en el momento de crear una tabla para que los datos se ajusten a una serie de características predefinidas que mantengan su integridad.

Se conocen con su nombre en inglés y se refieren a los siguientes conceptos:

- a) **NOT NULL**. Exige la existencia de valor en la columna que lleva la restricción.
- b) **DEFAULT**. Proporciona un valor por defecto cuando la columna correspondiente no se le da valor en la instrucción de inserción. Este valor por defecto debe ser una constante. No se permiten funciones ni expresiones.
- c) **PRIMARY KEY**. Indica una o varias columnas como dato o datos que identifican únicamente cada fila de la tabla. Sólo existe una por tabla y en ninguna fila puede tener valor **NULL**, por definición. Es obligatoria su existencia en el modelo relacional.
- d) **FOREIGN KEY**. Indica que una determinada columna de una tabla, va a servir para referenciar a otra tabla en la que está definida la misma columna (columna o clave referenciada). El valor de la clave ajena deberá coincidir con uno de los valores de esta clave referenciada o ser **NULL**. No existe límite en el número de claves ajenas que pueda tener una tabla. Como caso particular, una clave ajena puede referenciar a la misma tabla en la que está. Para poder crear una tabla con clave ajena deberá estar previamente creada la tabla maestra en la que la misma columna es clave primaria.
- e) **UNIQUE**. Indica que esta columna o grupo de columnas debe tener un valor único. También admite valores nulos. Al hacer una nueva inserción se comprobará que el valor es único o **NULL**. Algunos sistemas gestores de bases de datos relacionales generan automáticamente índices para estas columnas
- f) **CHECK**. Comprueba si el valor insertado en esa columna cumple una determinada condición.

El nombre de la tabla puede tener de 1 a 30 caracteres, ha de ser única y no ser palabra reservada, el primer carácter ha de ser una letra.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
  ( col1  TipoDato  ,
    col2  TipoDato  ,
    .....
  ) ;
```

La cláusula **IF NOT EXISTS** previene el posible error generado si existiese una tabla con ese nombre.

3.1.2. CREACIÓN DE TABLAS CON LA DEFINICIÓN DE RESTRICCIONES

Los valores por defecto pueden ser: constantes, funciones SQL o las variables USER o SYSDATE. Las restricciones pueden definirse de dos formas, que llamaremos

1. **Restriccion1** Definición de la restricción a nivel de columna
2. **Restriccion2** Definición de la restricción a nivel de tabla

Estas restricciones, llamadas CONSTRAINTS se pueden almacenar con o sin nombre. Si no se lo damos nosotros lo hará el sistema siguiendo una numeración correlativa, que es poco representativa.

Es conveniente darle un nombre, para después podernos referirnos a ellas si las queremos borrar o modificar. Estos nombres que les damos a las CONSTRAINTS deben ser significativos para hacer más fácil las referencias. Por ejemplo:

- *pk_NombreTabla* para PRIMARY KEY
- *fk_NombreTabla1_NombreTabla2* FOREIGN KEY donde NombreTabla1 es la tabla donde se crea y NombreTabla2 es la tabla a la que referencia.
- *uq_NombreTabla_NombreColumna* para UNIQUE

3.1.2.1. RESTRICCIONES DEFINIDAS A NIVEL DE COLUMNA

Definimos cada restricción al mismo tiempo que definimos la columna correspondiente.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
( NombreColumna TipoDatos [Restriccion1]
[ , NombreColumna TipoDatos [Restriccion1] ..... ] );
```

donde Restriccion1..... es la definición de la restricción a nivel de columna

Las restricciones solo se pueden definir de esta forma si afectan a una sola columna, la que estamos definiendo es ese momento.

Definición de los diferentes tipos de CONSTRAINTS a nivel de tabla (Restriccion1)

- a) **CLAVE PRIMARIA:** PRIMARY KEY
- b) **POSIBILIDAD DE NULO:** NULL | NOT NULL
- c) **VALOR POR DEFECTO:** DEFAULT ValorDefecto
- d) **UNICIDAD:** UNIQUE
- e) **COMPROBACION DE VALORES:** CHECK (Expresion)

Nota: Esta cláusula de SQL estándar, en MySQL en la versión 5 está permitida pero no implementada

- f) **CLAVE AJENA:** REFERENCES NombreTabla [(NombreColumna)]

Notación: el nombre de tabla referenciada es el nombre de la tabla a la que se va a acceder con la clave ajena. Si la columna que forma la clave referenciada en dicha tabla no tiene el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro del paréntesis. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.

- g) **AUTO INCREMENTO:** AUTO_INCREMENT

Aunque no es propiamente una restricción, pero como parte de la definición de una columnas podemos indicar que sea AUTO_INCREMENT. De esta forma el sistema gestor irá poniendo valores en esta columna incrementándolos de 1 en 1 respecto al anterior y empezando por 1. Esta definición sólo se puede aplicar sobre columnas definidas como y enteras y que sean claves.

3.1.2.2. EJEMPLOS de RESTRICCIONES a Nivel de COLUMNA

a) **PRIMARY KEY.** El numero de socio en la tabla socios

```
mysql> CREATE TABLE socios_1a
-> (socio_no INT(4) PRIMARY KEY,
-> apellidos VARCHAR(14),
-> telefono CHAR(9),
-> fecha_alta DATE,
-> direccion VARCHAR(20),
-> codigo_postal INT(5));
```

b) **NOT NULL.** La columna teléfono es obligatoria en la tabla socios, nunca irá sin información.

```
mysql> CREATE TABLE socios_1b
-> (socio_no INT(4) PRIMARY KEY,
-> apellidos VARCHAR(14),
-> telefono CHAR(9) NOT NULL,
-> fecha_alta DATE,
-> direccion VARCHAR(20),
-> codigo_postal INT(5));
```

c) **DEFAULT.** En ausencia de valor el campo fecha _ alta tomará el valor de 1 de enero de 2000.

```
mysql> CREATE TABLE socios_1c
-> (socio_no INT(4) PRIMARY KEY,
-> apellidos VARCHAR(14),
-> telefono CHAR(9) NOT NULL,
-> fecha_alta DATE DEFAULT '2000-01-01',
-> direccion VARCHAR(20),
-> codigo_postal INT(5));
```

d) **UNIQUE.** La columna apellido será única en la tabla socios.

```
mysql> CREATE TABLE socios_1d
-> (socio_no INT(4) PRIMARY KEY,
-> apellidos VARCHAR(14) UNIQUE,
-> telefono CHAR(9) NOT NULL,
-> fecha_alta DATE DEFAULT '2000-01-01',
-> direccion VARCHAR(20),
-> codigo_postal INT(5));
```

e) **CHECK.** Se comprobará que la columna *cdigo_postal* corresponde a Madrid (entre 28000 y 28999)

```
mysql> CREATE TABLE socios_1e
-> (socio_no INT(4) PRIMARY KEY,
-> apellidos VARCHAR(14) UNIQUE,
-> telefono CHAR(9) NOT NULL,
-> fecha_alta DATE DEFAULT '2000-01-01',
-> direccion VARCHAR(20),
-> codigo_postal INT(5)
CHECK (codigo_postal BETWEEN 28000 AND 28999));
```

f) **FOREIGN KEY.** El campo socio_num de la tabla *prestamos* tendrá que tener valores existentes en el campo *num_socio* de la tabla *socios* o valor nulo.

```
mysql> CREATE TABLE prestamos
-> (num_prestamo INT(2) PRIMARY KEY,
-> socio_no INT(4) REFERENCES socios_1e(socio_no));
```

g) **AUTO INCREMENTO.** Crearemos una tabla con una columna AUTO_INCREMENT.

```
mysql> CREATE TABLE inventario
-> (num INT(2) AUTO_INCREMENT PRIMARY KEY,
-> descripcion VARCHAR(15));
```

3.1.2.3. RESTRICCIONES DEFINIDAS A NIVEL DE TABLA

En este caso definimos todas las restricciones al final de la sentencia, una vez terminada la definición de las columnas.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
( NombreColumna TipoDatos [ , NombreColumna TipoDatos..... ]
[Restriccion2 [ , Restriccion2]..... ] );
```

donde **Restriccion2.....** es la definición de la restricción a nivel de tabla

Las restricciones siempre se pueden definir de esta forma tanto si afectan a una sola columna como a varias columnas y puede darse un nombre a cada una de las restricciones.

Definición de los diferentes tipos de CONSTRAINTS a nivel de tabla (Restriccion2)

a) **PRIMARY KEY**

```
[CONSTRAINT [NombreConst]] PRIMARY KEY (NombreCol [,NombreCol...])
```

b) **UNIQUE**

```
[CONSTRAINT [NombreConst]] UNIQUE (NombreCol [,NombreCol...])
```

c) **CHECK**

```
[CONSTRAINT [NombreConst]] CHECK (Expresion)
```

d) **FOREIGN KEY**

```
[CONSTRAINT [NombreConst]] FOREIGN KEY (NombreCol [,NombreCol...]) REFERENCES
NombreTabla ([NombreCol [,NombreCol.....]])
```

Notación: los nombres de columna o columnas que siguen a la cláusula FOREIGN KEY es aquella o aquellas que están formando la clave ajena. Si hay más de una se separan por comas. El nombre de tabla referenciada es el nombre de la tabla a la que se va a acceder con la clave ajena. Si la columna o columnas que forman la clave referenciada en dicha tabla no tienen el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro de paréntesis. Si son más de una columna se separan por comas. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.

3.1.2.4. EJEMPLOS de RESTRICCIONES a Nivel de TABLA

Veremos un ejemplo de cada caso donde se van definiendo la columna con la correspondiente restricción. Algunos ejemplos con nombre de constraint y otros sin él. En un ejemplo, igual que el apartado anterior, de una biblioteca queremos guardar los datos de los socios en una tabla socios y los préstamos que se realizan en una tabla prestamos

a) **PRIMARY KEY.** El numero de socio será la clave primaria en la *tabla socios*. Será obligatoriamente no nulo y único.

```
mysql> CREATE TABLE socios_2a
-> (socio_no INT(4),
-> apellidos VARCHAR(14),
-> telefono CHAR(9),
-> fecha_alta DATE,
-> direccion VARCHAR(20),
-> codigo_postal INT(5),
-> CONSTRAINT PK2_DEPARTAMENTOS PRIMARY KEY (socio_no);
```

- b) **UNIQUE**. El campo apellido es único. Tendrá valores diferentes en cada fila o el valor nulo

```
mysql> CREATE TABLE socios_2b
-> (socio_no INT(4),
-> apellidos VARCHAR(14),
-> telefono CHAR(9),
-> fecha_alta DATE,
-> direccion VARCHAR(20),
-> codigo_postal INT(5),
-> CONSTRAINT PK_SOCIOS PRIMARY KEY(socio_no),
-> CONSTRAINT UQ_UNIQUE UNIQUE (apellidos));
```

- c) **CHECK**. La columna codigo_postal no admitirá como válidas aquellas filas en las que el código postal no tenga valores entre 28.000 y 28.999 (correspondientes a Madrid)

```
mysql> CREATE TABLE socios_2c
-> (socio_no INT(4),-> apellidos VARCHAR(14),
-> telefono CHAR(9),
-> fecha_alta DATE,
-> direccion VARCHAR(20),
-> codigo_postal INT(5),
-> CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY(socio_no),
-> CONSTRAINT UQ_UNIQUE UNIQUE(apellidos),
-> CONSTRAINT CK_CODIGO
-> CHECK (codigo_postal BETWEEN 28000 AND 28999) );
```

- d) **FOREIGN KEY**. El número de socio en una tabla *prestamos* será clave ajena referenciando a la columna correspondiente de la tabla *socios*.

```
mysql> CREATE TABLE prestamos_2
-> (num_prestamo INT(2),
-> socio_no INT(4) ,
-> CONSTRAINT PK_PRESTAMOS PRIMARY KEY(num_prestamo),
-> CONSTRAINT FK_SOCIO_PRESTAMOS FOREIGN KEY (socio_no)
REFERENCES socios_2c(socio_no) );
```

3.1.3. INTEGRIDAD REFERENCIAL

La definición de claves ajenas nos permiten mantener la integridad referencial en una base de datos relacional. Hemos dicho que la columna o columnas definidas como clave ajena deben tomar valores que se correspondan con un valor existente de la clave referenciada. La pregunta es: ¿qué sucede si queremos **borrar** o **modificar** un valor de la clave primaria refenciada? Pues que el sistema debe impedirnos realizar estas acciones pues dejaría de existir la integridad referencial.

Por ejemplo si tenemos

```
CREATE TABLE departamentos
(dep_no INT(4) CONSTRAINT pk_departamentos PRIMARY KEY.....);

CREATE TABLE empleados
(....dep_no INT(4) CONSTRAINT fk_empleados_departamentos
 REFERENCES departamentos(dep_no).... );
```

En este caso empleados.dep_no solo puede tomar valores que existan en departamentos.dep_no pero ¿que sucede si queremos borrar o modificar un valor de departamentos.dep_no.? El sistema no nos lo permitirá si existen filas con ese valor en la tabla de la clave ajena.

Sin embargo, en ocasiones, será necesario hacer estas operaciones. Para mantener la integridad de los datos, al borrar (**DELETE**), modificar (**UPDATE**)una fila de la tabla referenciada, el sistema no nos permitirá llevarlo a cabo si existe una fila con el valor referenciado. También se conoce como **RESTRICT**. Es la opción por defecto, pero existen las siguientes opciones en la definición de la clave ajena:

- **CASCADE.** El borrado o modificación de una fila de la tabla referenciada lleva consigo el borrado o modificación en cascada de las filas de la tabla que contiene la clave ajena. Es la más utilizada.
- **SET NULL.** El borrado o modificación de una fila de la tabla referenciada lleva consigo poner a NULL los valores de las claves ajenas en las filas de la tabla que referencia.
- **SET DEFAULT.** El borrado o modificación de una fila de la tabla referenciada lleva consigo poner un valor por defecto en las claves ajenas de la tabla que referencia.
- **NO ACTION.** El borrado o modificación de una fila de la tabla referenciada solo se produce si no existe ese valor en la tabla que contiene la clave ajena. Tiene el mismo efecto que RESTRICT.

Formato de la definición de clave ajena con opciones de referencia

```
REFERENCES NombreTabla [ ( NombreColumna [ , NombreColumna ] ) ]
[ON DELETE {CASCADE| SET NULL|NO ACTION | SET DEFAULT| RESTRICT}]
[ON UPDATE {CASCADE| SET NULL|NO ACTION | SET DEFAULT| RESTRICT}]
```

por ejemplo

```
CREATE TABLE empleados
(
    ...
    dep_no NUMBER(4) [DEFAULT valor],
    CONSTRAINT FK_EMPLEADOS_DEPARTAMENTOS REFERENCES departamentos(dep_no)
        ON DELETE SET NULL ON UPDATE CASCADE,
    ...
);
```

Esto quiere decir que el sistema pondrá nulos en `dep_no` de la tabla `empleados` si se borra el valor correspondiente en `departamentos` y modificará el valor de `dep_no` en la tabla `empleados` con el nuevo valor si se modifica la columna `dep_no` en la tabla `departamentos`.

En el tema siguiente veremos con más detalle los borrados y modificaciones en los casos en los que existan estas cláusulas en las definiciones de las tablas, realizando algún ejemplo.

Restricciones en MySQL FOREIGN KEY (El motor de almacenamiento InnoDB)

Las definiciones de claves foráneas están sujetas a las siguientes condiciones:

- Ambas tablas deben ser `InnoDB` y no deben ser tablas temporales.
- Si se proporciona un `CONSTRAINT simbolo`, éste debe ser único en la base de datos. Si no se suministra, `InnoDB` crea el nombre automáticamente.

`InnoDB` rechaza cualquier operación `INSERT` o `UPDATE` que intente crear un valor de clave foránea en una tabla hija sin un valor de clave candidata coincidente en la tabla padre. La acción que `InnoDB` lleva a cabo para cualquier operación `UPDATE` o `DELETE` que intente actualizar o borrar un valor de clave candidata en la tabla padre que tenga filas coincidentes en la tabla hija depende de la *accion referencial* especificada utilizando las subcláusulas `ON UPDATE` y `ON DELETE` en la cláusula `FOREIGN KEY`. Cuando el usuario intenta borrar o actualizar una fila de una tabla padre, `InnoDB` soporta cinco acciones respecto a la acción a tomar:

- **CASCADE:** Borra o actualiza el registro en la tabla padre y automáticamente borra o actualiza los registros coincidentes en la tabla hija. Tanto `ON DELETE CASCADE` como `ON UPDATE CASCADE` están disponibles en MySQL 5.0. Entre dos tablas, no se deberían definir varias cláusulas `ON UPDATE CASCADE` que actúen en la misma columna en la tabla padre o hija.
- **SET NULL:** Borra o actualiza el registro en la tabla padre y establece en `NULL` la o las columnas de clave foránea en la tabla hija. Esto solamente es válido si las columnas de clave foránea no han sido definidas como `NOT NULL`.

- **NO ACTION**: Significa *ninguna acción* en el sentido de que un intento de borrar o actualizar un valor de clave primaria no será permitido si en la tabla referenciada hay una valor de clave foránea relacionado.
- **RESTRICT**: Rechaza la operación de eliminación o actualización en la tabla padre. **NO ACTION** y **RESTRICT** son similares en tanto omiten la cláusula **ON DELETE** u **ON UPDATE**.
- **SET DEFAULT**: Esta acción es reconocida por el procesador de sentencias (parser), pero **InnoDB rechaza** definiciones de tablas que contengan **ON DELETE SET DEFAULT** u **ON UPDATE SET DEFAULT**.

InnoDB soporta las mismas opciones cuando se actualiza la clave candidata en la tabla padre. **Con CASCADE**, las columnas de clave foránea en la tabla hija son establecidas a los nuevos valores de la clave candidata en la tabla padre. Del mismo modo, las actualizaciones se producen en cascada si las columnas actualizadas en la tabla hija hacen referencia a claves foráneas en otra tabla.

Las columnas involucradas en la clave foránea y en la clave referenciada deben tener similares tipos de datos internos dentro de **InnoDB**, de modo que puedan compararse sin una conversión de tipo. La **longitud y la condición de con o sin signo de los tipos enteros deben ser iguales**. La longitud de los tipos cadena no necesita ser la misma. Si se especifica una acción **SET NULL**, hay que asegurarse de que **las columnas en la tabla hija no se han declarado como NOT NULL**.

Si MySQL informa que ocurrió **un error número 1005** en una sentencia **CREATE TABLE** y la cadena con el mensaje de error se refiere al errno (número de error) 150, significa que la creación de una tabla falló debido a una restricción de clave foránea formulada incorrectamente. Del mismo modo, si un **ALTER TABLE** falla y hace referencia al número de **error 150**, significa que se ha formulado incorrectamente una restricción de clave extranjera cuando se alteró la tabla. Se puede emplear **SHOW INNODB STATUS** para mostrar una explicación detallada del último error de clave foránea sufrido por **InnoDB** en el servidor.

Una desviación del estándar SQL: Si en la tabla padre hay varios registros que contengan el mismo valor de clave referenciada, entonces **InnoDB** se comporta en las verificaciones de claves extranjeras como si los demás registros con el mismo valor de clave no existiesen. Por ejemplo, si se ha definido una restricción del tipo **RESTRICT**, y hay un registro hijo con varias filas padre, **InnoDB** no permite la eliminación de ninguna de éstas.

Una desviación del estándar SQL: Si **ON UPDATE CASCADE** u **ON UPDATE SET NULL** vuelven a modificar la *misma tabla* que se está actualizando en cascada, el comportamiento es como en **RESTRICT**. Esto significa que en una tabla no se pueden ejecutar operaciones **ON UPDATE CASCADE** u **ON UPDATE SET NULL** que hagan referencia a ella misma. De ese modo se previenen bucles infinitos resultantes de la actualización en cascada. En cambio, una operación **ON DELETE SET NULL**, puede hacer referencia a la misma tabla donde se encuentra, al igual que **ON DELETE CASCADE**.

Note: Actualmente, los disparadores no son activados por acciones de claves foráneas en cascada.

Un ejemplo sencillo que relaciona tablas **padre** e **hijo** a través de una clave foránea de una sola columna:

```
CREATE TABLE parent
(d INT NOT NULL,
 PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE child
(id INT,
 parent_id INT,
 FOREIGN KEY (parent_id) REFERENCES parent(id) ON DELETE CASCADE
) ENGINE=INNODB;
```

Aquí, un ejemplo más complejo, en el cual una tabla **product_order** tiene claves foráneas hacia otras dos tablas. Una de las claves foráneas hace referencia a un índice de dos columnas en la tabla **product**. La otra hace referencia a un índice de una sola columna en la tabla **customer**:

```

CREATE TABLE product
(category INT NOT NULL,
 id INT NOT NULL,
 price DECIMAL,
 PRIMARY KEY(category, id)) ENGINE=INNODB;

CREATE TABLE customer
(id INT NOT NULL,
 PRIMARY KEY (id)) ENGINE=INNODB;

CREATE TABLE product_order
(no INT NOT NULL AUTO_INCREMENT,
 product_category INT NOT NULL,
 product_id INT NOT NULL,
 customer_id INT NOT NULL,
 PRIMARY KEY(no),
 FOREIGN KEY (product_category, product_id) REFERENCES
 product(category, id) ON UPDATE CASCADE ON DELETE RESTRICT,
 FOREIGN KEY (customer_id) REFERENCES customer(id))
ENGINE=INNODB;

```

InnoDB permite agregar una nueva restricción de clave foránea a una tabla:

```

ALTER TABLE yourtablename
ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

```

InnoDB también soporta el uso de `ALTER TABLE` para borrar claves foráneas:

```
ALTER TABLE nombre_tabla DROP FOREIGN KEY símbolo_clave_foránea;
```

Si la cláusula `FOREIGN KEY` incluye un **nombre de constraint** cuando se crea la clave foránea, se puede utilizar ese nombre **para eliminarla**. En otro caso, el valor `símbolo_clave_foránea` es generado internamente por **InnoDB** cuando se crea la clave foránea.

InnoDB devuelve las definiciones de claves foráneas de una tabla como parte de la salida de la sentencia `SHOW CREATE TABLE`:

```
SHOW CREATE TABLE tbl_name;
```

A partir de esta versión, si `mysqldump` también produce definiciones correctas de las tablas en el fichero generado, sin omitir las claves foráneas.

Se pueden mostrar las restricciones de claves foráneas de una tabla de este modo:

```
SHOW TABLE STATUS FROM nombre_bd LIKE 'nombre_tabla';
```

Las restricciones de claves foráneas se listan en la columna `Comment` de la salida producida.

3.1.4. FORMATO COMPLETO DE LA CREACIÓN DE TABLAS

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
( NombreColumna TipoDatos [Restriccion1 ]
[ , NombreColumna TipoDatos [Restriccion1..... ]
[Restriccion2 [ , Restriccion2....] );
```

Notación: los diferentes formatos de definición de restricciones, restricción1 y restricción2, están entre corchetes porque son opcionales, pudiéndose elegir entre ambos **sólo si la restricción afecta a una sola columna**.

Vamos a crear la tabla socios y prestamos con el formato completo. Algunas CONSTRAINTS las creamos a nivel de columna y otras de tabla:

```
CREATE TABLE socios
(socio_no INT(4),
apellidos VARCHAR(14),
telefono CHAR(9) NOT NULL,
fecha_alta DATE DEFAULT '2000-01-01',
CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY(socio_no),
CONSTRAINT UQ_UNIQUE UNIQUE(apellidos));

CREATE TABLE prestamos
(num_prestamo INT(2) PRIMARY KEY,
socio_no INT(4) ,
CONSTRAINT FK_SOCIO_PRESTAMOS FOREIGN KEY (socio_no)
REFERENCES socios(socio_no));
```

Mas Ejemplos:

```
CREATE TABLE Tabla1
(COL1 CHAR(5) PRIMARY KEY,
COL2 INTEGER NOT NULL,
COL3 INTEGER UNIQUE,
COL4 INTEGER DEFAULT 100 ) ENGINE=InnoDB;

CREATE TABLE Tabla1CP
(COL1 CHAR(5),
COL2 INTEGER NOT NULL,
COL3 INTEGER UNIQUE,
PRIMARY KEY (COL1) ) ENGINE=InnoDB;

CREATE TABLE Tabla1CF
(COL1 CHAR(5),
COL2 INTEGER NOT NULL,
COL3 INTEGER UNIQUE,
PRIMARY KEY (COL1),
FOREIGN KEY (COL2) REFERENCES Tabla1CP (COL1) ) ENGINE=InnoDB;

CREATE TABLE Tabla1CF2
(COL1 CHAR(5),
COL2 INTEGER NOT NULL,
COL3 INTEGER UNIQUE,
PRIMARY KEY (COL1,COL2),
CONSTRAINT ClaveF1 FOREIGN KEY (COL1) REFERENCES Tabla1 (COL1),
CONSTRAINT ClaveF2 FOREIGN KEY (COL2) REFERENCES Tabla1CP (COL1)
) ENGINE=InnoDB;

CREATE TABLE AutoIncre
(COL1 INTEGER AUTO_INCREMENT PRIMARY KEY,
COL2 INTEGER ) ENGINE=InnoDB;
```

3.1.5. CREACIÓN DE UNA TABLA A PARTIR DE OTRA

**CREATE TABLE NombreTabla [(Col1, Col2, ... Coln)] AS
SELECT ...**

*Vamos a crear la tabla ProMadrid formada por el código (S#) y el nombre (NOMBS) de aquellos proveedores que sean de MADRID.

*CREATE TABLE ProMadrid
AS SELECT S#,NOMBS
FROM Proveedores
WHERE CIUDAD = 'MADRID'*

Hay que tener en cuenta que cuando se lleva a cabo esta operación, la tabla ProMadrid se crea con los datos seleccionados de la tabla origen (Proveedores), pero **no con las claves de esta**, por los que si deseamos que tenga claves tendremos que añadirlas, tarea que veremos más adelante.

Comprobamos que se ha llevado a cabo la operación de forma correcta :

*SELECT *
FROM ProMadrid*

No es necesario especificar tipos ni tamaño de las columnas, ya que vienen determinados por los tipos y tamaños de las recuperadas en la consulta

Ahora queremos asignar un nombre distinto a las columnas de EJEMPLO_AS:

*CREATE TABLE Ejemplo (COL1, COL2, COL3, COL4) AS
SELECT * FROM EJEMPLO2;*

Habrá que vigilar qué restricciones se crean en la tabla destino y cuáles no.

3.2. MODIFICACIÓN de TABLAS

Una vez que hemos creado una tabla, a menudo se presenta la necesidad de tener que modificarla. La sentencia SQL que realiza esta función es **ALTER TABLE**.

La especificación de la modificación es parecida a la de la sentencia **CREATE** pero varía según el objeto SQL del que se trate.

```
ALTER TABLE NombreTabla  
EspecificacionModificacion [ , EspecificacionModificacion..... ]
```

Donde **NombreTabla**..... nombre de la tabla se desea modificar.

EspecificacionModificacion... las modificaciones que se quieren realizarse sobre la tabla

Las modificaciones que se pueden realizar sobre una tabla son:

- Añadir una nueva columna
- Añadir un nueva restricción
- Borrar una columna
- Borrar una restricción
- Modificar una columna sin cambiar su nombre
- Modificar una columna y cambiar su nombre
- Renombrar la tabla

a) AÑADIR UNA NUEVA COLUMNA:

```
ADD [COLUMN] NombreColumna TipoDatos [ Restriccion1 ]
```

Puede añadirse una nueva columna y todas las restricciones, salvo NOT NULL. La razón es que esta nueva columna tendrá los valores NULL al ser creada.

b) AÑADIR UNA CONSTRAINT:

```
ADD [CONSTRAINT [NombreConstraint] ]  
      PRIMARY KEY (NombreColumna [, NombreColumna... ] )  
ADD [CONSTRAINT [NombreConstraint] ]  
      FOREIGN KEY (NombreColumna [, NombreColumna... ] )  
          REFERENCES NombreTabla[(NombreColumna[, NombreColumna... ] )]  
ADD [CONSTRAINT [NombreConstraint] ]  
      UNIQUE (NombreColumna [, NombreColumna... ] )
```

c) BORRAR UNA COLUMNA: **DROP COLUMN** NombreColumna

d) BORRAR UNA CONSTRAINT:

Clave Primaria: **DROP PRIMARY KEY**
Clave Foranea: **DROP FOREIGN KEY NombreConstraint**
 DROP INDEX NombreConstraint
UNIQUE: **DROP INDEX NombreConstraint**

e) MODIFICAR UNA COLUMNA SIN CAMBIAR SU NOMBRE:

```
MODIFY [COLUMN] NombreColumna TipoDatos [Restriccion1]
```

f) MODIFICAR LA DEFINICION DE UNA COLUMNA Y SU NOMBRE

```
CHANGE [COLUMN] NombreColumnaAntiguo NombreColumnaNuevo  
           TipoDatos [Restriccion1]
```

f) RENOMBRAR LA TABLA: **RENAME** [TO] NombreTablaNuevo

3.2.1. EJEMPLOS DE MODIFICACIÓN DE TABLAS

a) Ejemplo de añadir una columna

Añadir la columna para la dirección de correo electrónico, `direccion_correo_2c`, a la tabla de `socios_2c` con un tipo de dato alfanumérico de 20 caracteres.

```
mysql> ALTER TABLE socios_2c  
->     ADD (direccion_correo_e varchar(9));
```

b) Ejemplo de añadir una restriccion

Añadir en la tabla `socios_2cla` restricción UNIQUE para la columna telefono.

```
mysql> ALTER TABLE socios_2c  
->     ADD CONSTRAINT uq_socios_telefono UNIQUE(telefono);
```

c) Ejemplo de borrar una columna

Borrar la columna `fecha_altade` la tabla `socios_a`

```
mysql> ALTER TABLE socios_1a  
->     DROP COLUMN fecha_alta;
```

d) Ejemplos de borrado de restricciones

Borrar la clave primaria de la tabla `socios_1a`

```
mysql> ALTER TABLE socios_1a  
->     DROP PRIMARY KEY;
```

Borrar en la tabla `socios_2cla` restricción UNIQUE para la columna telefono.

```
mysql> ALTER TABLE socios_2c  
->     DROP INDEX uq_socios_telefono;
```

e) Ejemplos de modificación de una columna

Modificar la tabla `socios_1a` para poner NOT NULL en la columna apellidos.

```
mysql> ALTER TABLE socios_1a  
->     MODIFY apellidos VARCHAR(14) NOT NULL;
```

f) Ejemplo de cambio de definición de una columna

Modificar la tabla `socios_1a` cambiándole el nombre a la columna apellidos por `apellidos_a` e incrementar de 14 a 20 caracteres el tamaño.

```
mysql> ALTER TABLE socios_1a  
->     CHANGE apellidos apellidos_a VARCHAR(20);
```

g) Renombrar una tabla

Cambiar el nombre de la tabla `socios_1a` por `socios1a`

```
mysql> ALTER TABLE socios_1a  
->     RENAME TO socios1a;  
  
mysql> SELECT * FROM socios_1a;  
ERROR 1146 (42S02): Table 'test.socios_1a' doesn't exist
```

3.3. BORRADO de TABLAS

Para borrar una tabla y su contenido de la base de datos se utiliza la sentencia `DROP TABLE`.

```
DROP TABLE [ IF EXISTS ] NombreTabla
[CASCADE | RESTRICT] ;
```

La cláusula `IF EXISTS` previene los errores que puedan producirse si no existe la tabla que queremos borrar.

Nota: las cláusulas `CASCADE` y `RESTRICT` para borrado en cascada y borrado de las restricciones están permitidas pero no implementadas en esta versión.

3.3.1. EJEMPLOS DE BORRADO DE UNA TABLA

1- Borraremos las tablas `departamentos2` y `empleados2` enlazadas con una clave ajena.
Hay que tener cuidado con el orden:

```
mysql> DROP TABLE departamentos2;
ERROR 1217 (23000): Cannot delete or update a parent row: a foreign key constraint fails
```

Debemos borrar primero `empleados2`:

```
mysql> DROP TABLE empleados2;
Query OK, 0 rows affected (0.10 sec)
mysql> DROP TABLE departamentos2;
Query OK, 0 rows affected (0.06 sec)
```

2 - Borraremos una tabla `empleados7`, que no existe, con la cláusula `IF EXISTS` y vemos que no nos da error.

```
mysql> DROP TABLE IF EXISTS empleados7;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

3.4. RENOMBRADO

Permite cambiar de nombre una tabla, asignándole un nombre nuevo y el nombre antiguo desaparece.

```
RENAME TABLE NombreTablaAntiguo TO NombreTablaNuevo
[, NombreTablaAntiguo TO NombreTablaNuevo .....]
```

Notación: se pueden renombrar varias tablas en una sentencia por eso van entre corchetes las siguientes tablas a renombrar.

donde `NombreTablaAntiguo` es el nombre antiguo de la tabla, que debe existir
`NombreTablaNuevo.....` es el nombre nuevo de la tabla, que no debe existir.

Permite renombrar en la misma instrucción una o varias tablas.

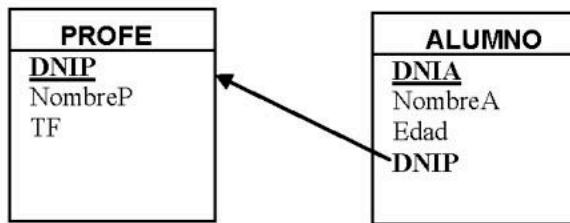
Ejemplo

Renombrar la tabla `socios` para que su nuevo nombre sea `socios2`

```
mysql> RENAME TABLE socios TO socios2;
mysql> select * from socios;
ERROR 1146 (42S02): Table 'test.socios' doesn't exist
```

4. PRÁCTICAS

4.1. A partir del siguiente diseño:



PROFE	ALUMNO
DNIP NombreP: Texto de 20 caracteres. – (Obligatorio) TF: Texto de 9 caracteres – (ÚNICO)	DNIA NombreA: Texto de 20 caracteres. – (por defecto ‘Jose’) Edad: Numérico – (Mayor de 18)

Practica-1: Crear las tablas SIN RESTRICCIONES

Practica-2: Crear las tablas CON RESTRICCIONES a nivel de COLUMNA.

Practica-3: Crear las tablas CON RESTRICCIONES a nivel de TABLA.

Practica-4: Crear las tablas CON RESTRICCIONES e INTEGRIDAD REFERENCIAL a nivel de TABLA.

Practica-5: Crear las tablas SIN RESTRICCIONES y AÑADIRSELAS DESPUÉS.

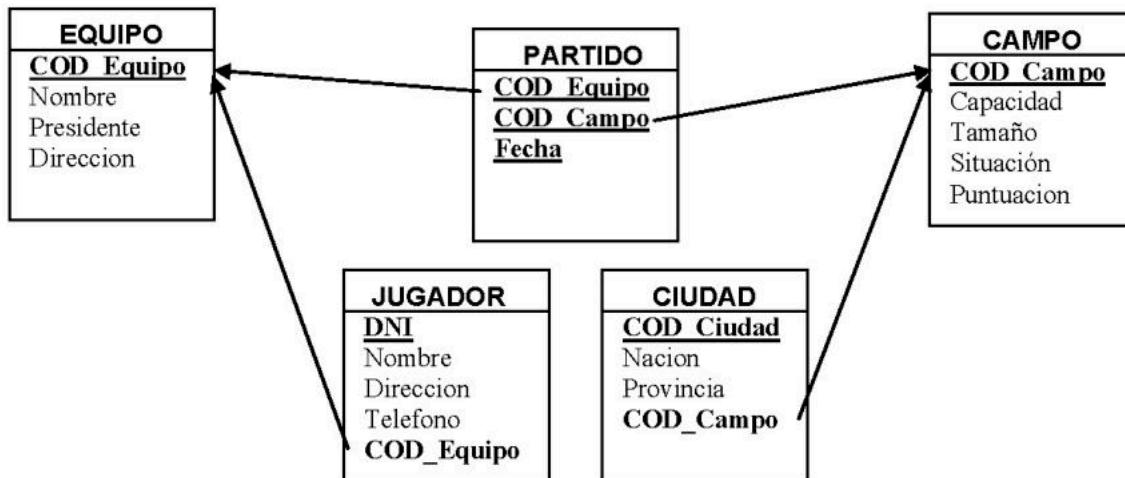
Practica-6:

- Añadir un atributo.
- Cambiar el tipo de un atributo.
- Cambiar el nombre de un atributo.
- La Clave Foránea tiene que ser con UPDATE CASCADE.
- La clave de alumno en lugar de DNIA debe ser NombreA.

Practica-7: Quitar todas las restricciones.

4.2. Práctica - A

1) Implementa en MySQL la BD Fútbol, definida por el siguiente esquema:

**ESPECIFICACIONES**

EQUIPO <u>COD_Equipo</u> : Nombre: Texto de 6 caracteres. Presidente: Texto de 10 caracteres Direccion: Texto de 10 caracteres	PARTIDO <u>COD_Equipo</u> : <u>COD_Campo</u> : <u>Fecha</u> :
CAMPO <u>COD_Campo</u> : Capacidad: Numérico Tamaño: Numérico y por defecto 20.000. Situación: Texto de 10 caracteres. Opcional. Puntuación: Numérico.	JUGADOR <u>DNI</u> : Nombre: Texto de 10 caracteres Direccion: Texto de 10 caracteres. Opcional Telefono: Texto de 10 caracteres. <u>COD_Equipo</u> :
CIUDAD <u>COD_Ciudad</u> : Nacion: Texto de 10 caracteres Provincia: Texto de 10 caracteres <u>COD_Campo</u> : <ul style="list-style-type: none"> • Casi todos los campos están situados en España. 	<ul style="list-style-type: none"> • Las tablas deben quedar relacionadas para un mejor manejo de las mismas. • Todo campo que no esté indicado como opcional se considera obligatorio.

4.3. Práctica - B

Crear la BD Facturas con la siguiente DESCRIPCION DE LAS TABLAS:

Tabla **COMPRADORES**

CIF_comprador	alfanumérico de 11 caracteres.
Nombre_social	alfanumérico de 30 caracteres.
Domicilio_social	alfanumérico de 30 caracteres.
Localidad	alfanumérico de 30 caracteres.
C_postal	alfanumérico de 5 caracteres.
Teléfono	alfanumérico de 9 caracteres.

Tabla **ARTICULOS**

Referencia_articulo	alfanumérico de 12 caracteres.
Descripción_articulo	alfanumérico de 30 caracteres.
Precio_unidad	numérico de 6 posiciones, con dos decimales
IVA	numérico de 2 posiciones.
Existencias_actuales	numérico de 5 posiciones.

Tabla **FACTURAS**

Factura_no	numérico de 6 posiciones
Fecha_factura	tipo fecha
CIF_cliente	alfanumérico de 11 caracteres

Tabla **LINEAS_FACTURA**

Factura_no	numérico de 6 posiciones
Referencia_articulo	alfanumérico de 12 caracteres..
Unidades	numérico de 3 posiciones.

1. Creación de las tablas con las restricciones:

- a. Crear la tabla COMPRADORES con la columna *cif_comprador* como **clave primaria** con nombre **PK_COMPRADORES_CIF**, y la columna *nombre_social* **única** con nombre **UQ_COMPRADORES_NOMBRE_SOCIAL**. La columna *teléfono* debe ser **obligatoria**.
 - b. Crear la tabla ARTICULOS, con *referencia_articulo* como **PRIMARY KEY** con el nombre **PK_ARTICULOS**, la columna *IVA* con valores entre 5 y 25 inclusive y la columna *existencias_actuales* con valor por **defecto** 0.
 - c. Crear la tabla FACTURAS con la columna *factura_no* como clave primaria con el nombre **PK_FACTURAS**, y la columna *fecha_factura* tendrá como valor por defecto la fecha 1 de enero de 2005.
 - d. Crear la tabla LINEAS_FACTURAS con las columnas *factura_no* y *referencia_articulo* como PRIMARY KEY con nombre **PK_LINEAS_FACTURA**, la columna *factura_no* como FOREIGN KEY con nombre **FK_LINEAS_FACTURAS** referenciando la columna *factura_no* de la tabla facturas con borrado en cascada y la columna *referencia_articulo* como FOREIGN KEY con nombre **FK_LINEAS_ARTICULOS** referenciando la columna *referencia_articulo* de la tabla articulos
2. Añadir a la tabla FACTURAS la columna *cod_oficina* de tipo numérico de 4 posiciones, con el número de oficina.
 3. Añadir en la tabla FACTURAS la columna *cif_cliente* como FOREIGN KEY con nombre **FK_FACTURA_COMPRADORES** referenciando a la columna *cif_comprador* de la tabla compradores
 4. Cambiar en la tabla COMPRADORES el nombre de la columna *c_postal* por *codigo_postal*
 5. Añadir a la columna *cod_oficina* de la tabla FACTURAS la comprobación de estar entre 1 y 1000

4.4. Práctica - C

- 1) Crear las siguientes tablas de acuerdo con las restricciones que se mencionan, dando nombre a todas las restricciones creadas. Como guía para los nombres de las restricciones:

PK_Nombreatributo
 FK_Nombredelastablasquerelaciona
 CK1_Nombreatributo ... CK2_Nombreatributo

A. Tabla *FABRICANTES*

Nombre	Tipo	¿Nulo?
COD_FABRICANTE	INTEGER(3)	NOT NULL
NOMBRE	VARCHAR(15)	
PAIS	VARCHAR(15)	

- La clave primaria es COD_FABRICANTE

B. Tabla *ARTICULOS*

Nombre	Tipo	¿Nulo?
ARTICULO	VARCHAR(20)	NOT NULL
COD_FABRICANTE	INTEGER(3)	NOT NULL
PESO	INTEGER(3)	NOT NULL
CATEGORIA	VARCHAR(10)	NOT NULL
PRECIO_VENTA	INTEGER (4)	
PRECIO_COSTO	INTEGER (4)	
EXISTENCIAS	INTEGER (5)	

- La clave primaria está formada por las columnas: ARTICULO, COD_FABRICANTE, PESO y CATEGORIA
- COD_FABRICANTE es clave ajena que referencia a la tabla FABRICANTES
- PRECIO_VENTA, PRECIO_COSTO y PESO han de ser > 0
- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'

C. Tabla *TIENDAS*

Nombre	Tipo	¿Nulo?
NIF	VARCHAR(10)	NOT NULL
NOMBRE	VARCHAR(20)	
DIRECCION	VARCHAR(20)	
POBLACION	VARCHAR(20)	
PROVINCIA	VARCHAR(20)	
CODPOSTAL	INTEGER(5)	

- La clave primaria es NIF

D. Tabla *PEDIDOS*

Nombre	Tipo	¿Nulo?
NIF	VARCHAR(10)	NOT NULL
ARTICULO	VARCHAR(20)	NOT NULL
COD_FABRICANTE	INTEGER(3)	NOT NULL
PESO	INTEGER(3)	NOT NULL
CATEGORIA	VARCHAR(10)	NOT NULL
FECHA_PEDIDO	DATE	NOT NULL
UNIDADES_PEDIDAS	INTEGER(4)	

- La clave primaria está formada por las columnas: NIF, ARTICULO, COD_FABRICANTE, PESO, CATEGORIA y FECHA_PEDIDO
- UNIDADES_PEDIDAS ha de ser > 0

- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'
- Las columnas ARTICULO, COD_FABRICANTE, PESO Y CATEGORIA son clave ajena y referencian a la tabla ARTICULOS. Realizar un borrado en cascada.
- NIF es clave ajena y referencia a la tabla TIENDAS

E. Tabla *VENTAS*

Nombre	Tipo	¿Nulo?
NIF	VARCHAR(10)	NOT NULL
ARTICULO	VARCHAR(20)	NOT NULL
COD_FABRICANTE	INTEGER(3)	NOT NULL
PESO	INTEGER(3)	NOT NULL
CATEGORIA	VARCHAR(10)	NOT NULL
FECHA_VENTA	DATE	NOT NULL
UNIDADES_VENDIDAS	INTEGER(4)	

- La clave primaria está formada por las columnas: NIF, ARTICULO, COD_FABRICANTE, PESO, CATEGORIA y FECHA_VENTA
- UNIDADES_VENDIDAS ha de ser > 0
- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'
- Las columnas ARTICULO, COD_FABRICANTE, PESO Y CATEGORIA son clave ajena y referencian a la tabla ARTICULOS. Realizar un borrado en cascada.
- NIF es clave ajena y referencia a la tabla TIENDAS

2) ¿Cuál sería el orden correcto a la hora de crear un SCRIPT que genere todas estas tablas para que no de errores a la hora de ejecutarlo? Preparar dicho script y ejecutarlo para comprobar que no da ningún error.

Por otro lado, ¿qué instrucciones de borrado de tablas deberíamos poner en primer lugar en dicho SCRIPT y en qué orden para borrar cada una de las tablas por si acaso existen previamente en la base de datos (sin utilizar la cláusula CASCADE CONSTRAINTS)?

3) Modificar las columnas de las tablas PEDIDOS y VENTAS para que las UNIDADES_VENDIDAS y LAS UNIDADES_PEDIDAS puedan almacenar cantidades numéricas de 6 dígitos

4) Impedir que se den de alta tiendas en la provincia de 'TOLEDO'

5. SOLUCIONES

5.1. Práctica - B

```
# creamos base de datos
```

```
CREATE DATABASE IF NOT EXISTS Facturas;
USE Facturas;
```

```
# borramos las tablas si existen
```

```
DROP TABLE IF EXISTS lineas_factura;
DROP TABLE IF EXISTS facturas;
DROP TABLE IF EXISTS articulos;
DROP TABLE IF EXISTS compradores;
```

```
# 1.creaacion de las tablas con restricciones
```

```
CREATE TABLE compradores(
    CIF_comprador      VARCHAR(11),
    Nombre_social       VARCHAR(30),
    Domicilio_social   VARCHAR(30),
    Localidad          VARCHAR(30),
    C_postal            CHAR(5),
    Teléfono           CHAR(9) NOT NULL,
    CONSTRAINT PK_COMPRADORES_CIF PRIMARY KEY(CIF_comprador),
    CONSTRAINT UQ_COMPRADORES_NOMBRE_SOCIAL UNIQUE(Nombre_social));
```

```
CREATE TABLE articulos(
    Referencia_articulo  VARCHAR(12),
    Descripcion_articulo VARCHAR(30),
    Precio_unidad        FLOAT(6,2),
    Iva                  INT(2),
    Existencias_actuales INT(5) DEFAULT 0,
    CONSTRAINT PK_ARTICULOS PRIMARY KEY(Referencia_articulo),
    CHECK (Iva BETWEEN 5 AND 25));
```

```
CREATE TABLE facturas(
    Factura_no          INT(6),
    Fecha_factura       DATE DEFAULT '2005-1-1',
    CIF_cliente         VARCHAR(11),
    CONSTRAINT PK_FACTURAS PRIMARY KEY(Factura_no));
```

```
CREATE TABLE lineas_factura(
    Factura_no          INT(6),
    Referencia_articulo VARCHAR(12),
    Unidades            INT(3),
    CONSTRAINT PK_LINEAS_FACTURA PRIMARY KEY
        (Factura_no,Referencia_articulo),
    CONSTRAINT FK_LINEAS_FACTURAS FOREIGN KEY
        (Factura_no) REFERENCES facturas(Factura_no)
        ON DELETE CASCADE,
    CONSTRAINT FK_LINEAS_ARTICULOS FOREIGN KEY
        (Referencia_articulo) REFERENCES articulos(Referencia_articulo));
```

2. Añadir a la tabla FACTURAS la columna "cod_oficina"

```
ALTER TABLE facturas  
ADD COLUMN cod_oficina INT(4);
```

3. Añadir en la tabla FACTURAS la columna cif_cliente como FORIEGN KEY
Con nombre FK_FACTURA_COMPRADORES referenciando a la columna
cif_comprador de la tabla compradores

```
ALTER TABLE facturas  
ADD CONSTRAINT FK_FACTURA_COMPRADORES FOREIGN KEY (CIF_cliente)  
REFERENCES compradores(CIF_comprador);
```

4. Cambiar en la tabla COMPRADORES el nombre de la columna c_postal por
codigo_postal

```
ALTER TABLE compradores  
CHANGE COLUMN C_Postal Código_postal CHAR(5);
```

5. Añadir a la columna cod_oficina de la tabla FACTURAS la comprobación
de estar entre 1 y 1000. La siguiente sentencia no funciona. Aunque según
los apuntes debería funcionar

```
/*ALTER TABLE facturas  
MODIFY cod_oficina INT(4) CHECK(cod_oficina BETWEEN 1 AND 1000);*/
```

Ni en los apuntes ni en el manual de MySQL especifica que se pueda
añadir una restricción CHECK Pero la siguiente sentencia MySQL la
interpreta correctamente.

```
ALTER TABLE facturas  
ADD CONSTRAINT CHECK(cod_oficina BETWEEN 1 AND 1000);
```