1.- Introducción

En este capítulo vamos a analizar la segunda categoría de sentencias de SQL , lenguajes de definición de datos.

Mediante las sentencias que vamos a ver en este tema podemos realizar sobre una base de datos las siguientes operaciones :

- Crear, borrar y modificar tablas
- Crear y borrar bases de datos
- Crear, borrar y modificar los campos
- Crear, borrar y modificar claves : primarias, foráneas, únicas, ...
- Crear, borrar y modificar aserciones, checks y disparos

Los comandos de SQL característicos de esta parte son :

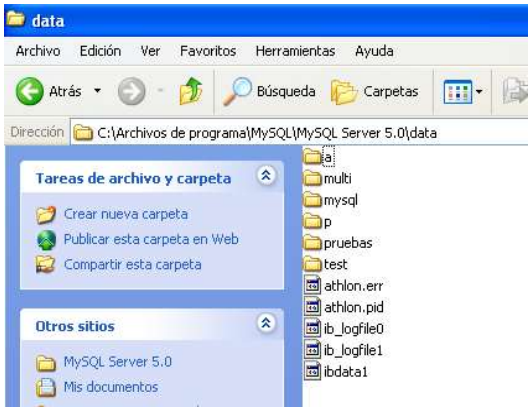
- ALTER
- CREATE
- DROP
- RENAME

Son las sentencias necesarias para definir cada uno de los elementos de una base de datos.

2.- Sentencia Create database

Mediante esta sentencia podemos realizar la creación de una base de datos (tambien llamado esquema o catalogo).

Cada conjunto de relaciones que componen un modelo completo forma una base de datos. Desde el punto de vista de SQL, una base de datos es sólo un conjunto de relaciones (o tablas), y para organizarlas o distinguirlas se accede a ellas mediante su nombre. A nivel de sistema operativo, cada base de datos se guarda en un directorio diferente.



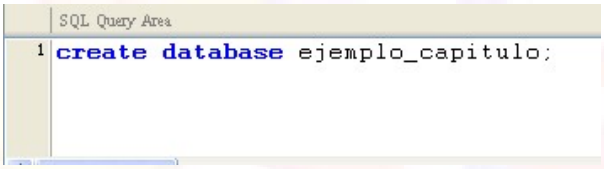
Debido a esto, crear una base de datos es una tarea muy simple. Claro que, en el momento de crearla, la base de datos estará vacía, es decir, no contendrá ninguna tabla.

Dado que no hay tablas en la base de datos cuando ésta se crea, la sentencia CREATE DATABASE sólo crea un directorio bajo el directorio de datos de MySQL.

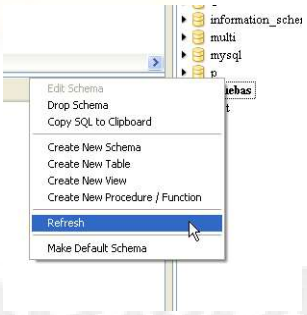
La sintaxis del comando seria :

CREATE DATABASE [IF NOT EXISTS] nombre;

Ejemplo



Con este comando hemos creado una base de datos, en el cliente para poder observarla en ciertas ocasiones deberás refrescar la lista de elementos.

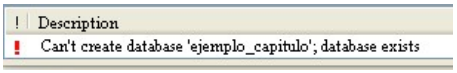


CREATE DATABASE crea una base de datos con el nombre dado.

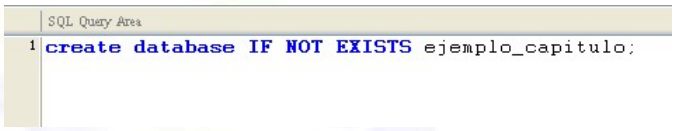
Se da un error si la base de datos existe, para evitarlo podemos utilizar IF NOT EXISTS.

Ejemplo

Si volvemos a ejecutar el comando del ejemplo anterior nos daria un error porque la base de datos ya existe.



Para evitar este error podemos utilizar el siguiente comando.



Una vez que habeis creado una base de datos normalmente se suele colocar como elemento por defecto para evitar tener que hacer referencia a ella en todos los comandos SQL. Para realizarlo podeis dar doble clic sobre la base de datos o utilizar el comando

USE base_datos

Ejemplo

Cuando tienes seleccionada una base de datos aparece en la barra de titulo del cliente.

3.- Comando drop database

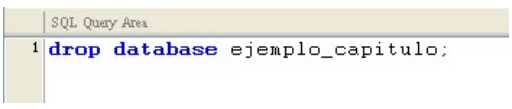
Con este comando borramos una base de datos completa, incluyendo todo lo que contenga.

La sintaxis del comando es :

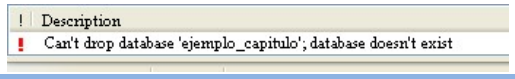
DROP {DATABASE | SCHEMA} [IF EXISTS] nombre_base_datos;

Si intentamos borrar una base de datos que no exista se produce un error, para evitarlo podeis utilizar la clausula IF EXISTS.

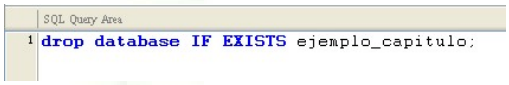
Ejemplo



Con este comando se borrara la base de datos ejemplo_capitulo. Si volvemos a ejecutar el comando se produce un error.



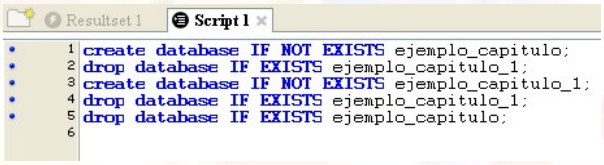
Para evitar este error podemos utilizar el siguiente comando.



Podemos utilizar la cláusula SCHEMA en vez de DATABASE a partir de la versión 5 de MySQL.

Ejemplo

Explicar que realiza el siguiente script.



4.- Comando Create Table

CREATE TABLE crea una tabla con un nombre dado en la base de datos actual. Se da un error si no existe base de datos actual o si la tabla ya existe.

La sintaxis del comando es bastante compleja, vamos a estudiarla diviendola en bloques. Tenemos la siguiente sintaxis:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nombre_tabla

[(definición_create,...)]

[opciones_tabla] [sentencia_select]

En MySQL 3.22 o posterior, el nombre de la tabla puede ser especificado como nombre_base_datos.nombre_tabla. Esto funciona independientemente de si existe una base de datos actual.

En MySQL 3.23 y posterior, puedes utilizar la palabra clave TEMPORARY cuando creas una tabla. Una tabla temporal se borrará automáticamente si se pierde una conexión y el nombre es por conexión. Esto significa que dos conexiones diferentes pueden utilizar el mismo nombre de tabla temporal sin entrar en conflicto con la otra o con una tabla existente del mismo nombre. (la tabla existente estará oculta hasta que la tabla temporal se borre). En MySQL 4.0.2. el usuario debe disponer del privilegio CREATE TEMPORARY TABLES para poder crear tablas temporales.

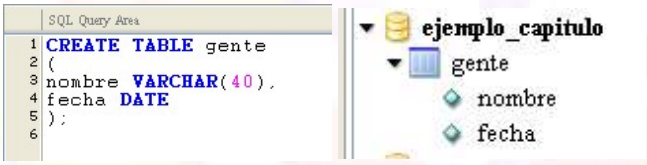
En MySQL 3.23 o superior, puedes utilizar las palabras clave IF NOT EXISTS para que no tenga lugar un error en caso que la tabla exista. Nota que no hay verificación de que las estructuras de las tablas sean idénticas.

Cada tabla table_name se representa con algunos archivos en el directorio de la base de datos.

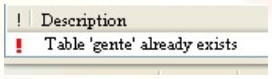
En su forma más simple, la sentencia CREATE TABLE creará una tabla con las columnas que indiquemos. Deberemos indicar el nombre de la tabla y los nombres y tipos de las columnas:

Ejemplo

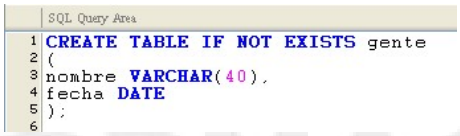
Crearemos, como ejemplo, una tabla que nos permitirá almacenar nombres de personas y sus fechas de nacimiento.



Si volvemos a ejecutar el mismo comando te dará un error.



Para evitarlo.



La sintaxis de definición_create seria la siguiente:

definición_columnas

| [CONSTRAINT [símbolo]] PRIMARY KEY (index_nombre_col,...)

| KEY [nombre_index] (nombre_col_index,...)

| INDEX [nombre_index] (nombre_col_index,...)

| [CONSTRAINT [símbolo]] UNIQUE [INDEX]

[nombre_index] [tipo_index] (nombre_col_index,...)

| [FULLTEXT|SPATIAL] [INDEX] [nombre_index] (nombre_col_index,...)

| [CONSTRAINT [símbolo]] FOREIGN KEY

[nombre_index] (nombre_col_index,...) [definición_referencia]

| CHECK (expr)

La sintaxis de definición_columnas:

nombre_col tipo [NOT NULL | NULL] [DEFAULT valor_por_defecto]

[AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']

[definición_referencia]

La sintaxis de tipo:

TINYINT[(longitud)] [UNSIGNED] [ZEROFILL]

| SMALLINT[(longitud)] [UNSIGNED] [ZEROFILL]

| MEDIUMINT[(longitud)] [UNSIGNED] [ZEROFILL]

| INT[(longitud)] [UNSIGNED] [ZEROFILL]

| INTEGER[(longitud)] [UNSIGNED] [ZEROFILL]

| BIGINT[(longitud)] [UNSIGNED] [ZEROFILL]

| REAL[(longitud,decimales)] [UNSIGNED] [ZEROFILL]

| DOUBLE[(longitud,decimales)] [UNSIGNED] [ZEROFILL]

| FLOAT[(longitud,decimales)] [UNSIGNED] [ZEROFILL]

| DECIMAL(longitud,decimales) [UNSIGNED] [ZEROFILL]

| NUMERIC(longitud,decimales) [UNSIGNED] [ZEROFILL]

| DATE

| TIME

| TIMESTAMP

| DATETIME

| CHAR(longitud) [BINARY | ASCII | UNICODE]

| VARCHAR(longitud) [BINARY]

| TINYBLOB

| BLOB

| MEDIUMBLOB

| LONGBLOB

| TINYTEXT

| TEXT

| MEDIUMTEXT

| LONGTEXT

| ENUM(valor1,valor2,valor3,...)

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

| tipo_spatial

La sintaxis de nombre_col_index:

nombre_col [(longitud)] [ASC | DESC]

La sintaxis de definición_referencia:

REFERENCES nombre_tbl [(nombre_col_index,...)]

[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

[ON DELETE opción_referencia]

[ON UPDATE opción_referencia]

La sintaxis de opción_referencia:

RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

La sintaxis de opciones_tabla:

opción_tabla [opción_tabla] ...

Las posibles opciones de tabla:

{ENGINE|TYPE} = {BDB|HEAP|ISAM|InnoDB|MERGE|MRG_MYISAM|MYISAM }

| AUTO_INCREMENT = valor

| AVG_ROW_LENGTH = valor

| CHECKSUM = {0 | 1}

| COMMENT = 'cadena'

| MAX_ROWS = valor

| MIN_ROWS = valor

| PACK_KEYS = {0 | 1 | DEFAULT}

| PASSWORD = 'cadena'

| DELAY_KEY_WRITE = {0 | 1}

| ROW_FORMAT = { DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNTANT|COMPACT}

| RAID_TYPE = { 1 | STRIPED | RAID0 }

RAID_CHUNKS=valor

RAID_CHUNKSIZE=valor

| UNION = (nombre_tabla,[nombre_tabla...])

| INSERT_METHOD = { NO | FIRST | LAST }

| DATA DIRECTORY = 'camino de directorio absoluto'

| INDEX DIRECTORY = 'camino de directorio absoluto'

| [DEFAULT] CHARACTER SET nombre_conjunto_caracteres [COLLATE nombre_cotejo]

La sintaxis de sentencia_select:

[IGNORE | REPLACE] [AS] SELECT ... (Alguna sentencia select legal)

Como podéis observar el numero de opciones es bastante elevado, por ello vamos a analizar el comando por bloques.

4.1.- Creación de campos simples

Vamos a ver como podemos utilizar este comando para generar columnas en una determinada tabla.

Simplemente debemos colocar el nombre de los campos y a continuación colocaremos el tipo de datos que van a almacenar.

Ejemplo

```
CREATE TABLE tabla1
(
   campo1 varchar(40),
   campo2 int,
   campo3 date
);
```

En el nombre de los campos debeis colocar las comillas `campo2` o en el nombre de las tablas, siempre que existan caracteres especiales como por ejemplo los espacios en blanco.

```
CREATE TABLE IF NOT EXISTS tabla1
(
  `campo1` varchar(40),
  `campo2` int,
  `campo3` date
);
```

4.2.- Definiciones de restricciones

Las restricciones nos van a permitir limitar los valores que puedes utilizar en las columnas. Esto ya lo vimos en la parte de diseño de bases de datos. Vamos a ver como se pueden aplicar en mysql

4.2.1.- Definir un valor predeterminado

Para ello utilizamos la opción DEFAULT. La palabra clave default nos permite dar un valor a un campo cuando carece de un valor de asignación al crearse la fila de la tabla.

Ejemplo

Definimos la tabla de la siguiente forma:

```
CREATE TABLE IF NOT EXISTS tabla1
(
  `campo1` varchar(40) default NULL,
  `campo2` int default 15,
  `campo3` date default '2003/12/1'
);
```

Insertamos una serie de registros con la siguiente instrucción:

```
SQL Query Area
insert tabla1 values();
```

El resultado es el siguiente:

```
SQL Query Area
SELECT * FROM tabla1 t
```

campo1	campo2	campo3
	15	2003-12-01
	15	2003-12-01
	15	2003-12-01
	15	2003-12-01
	15	2003-12-01
	15	2003-12-01

En caso de que algun campo no tenga valor por defecto se coloca el NULL.

Ejemplo

```
CREATE TABLE IF NOT EXISTS tabla1
(
  `campo1` varchar(40) default NULL,
  `campo2` int default 15,
  `campo3` date default '2003/12/1',
  campo4 char(1)
);
```

Insertamos una serie de registros en blanco y obtenemos el siguiente resultado.

```
SELECT * FROM tabla1;
```

campo1	campo2	campo3	campo4
NULL	15	2003-12-01	NULL
NULL	15	2003-12-01	NULL
NULL	15	2003-12-01	NULL
NULL	15	2003-12-01	NULL

4.2.2.- Exigir que un campo no quede vació

Al definir cada columna podemos decidir si podrá o no contener valores nulos.

Debemos recordar que, como vimos en los capítulos de modelado, aquellas columnas que son o forman parte de una clave primaria no pueden contener valores nulos.

Veremos que, si definimos una columna como clave primaria, automáticamente se impide que pueda contener valores nulos, pero este no es el único caso en que puede ser interesante impedir la asignación de valores nulos para una columna.

La opción por defecto es que se permitan valores nulos, NULL, y para que no se permitan, se usa NOT NULL.

Ejemplo

Creamos la siguiente tabla:

```
CREATE TABLE IF NOT EXISTS tabla1
(
  `campo1` varchar(40) default NULL,
  `campo2` int default 15,
  `campo3` date default '2003/12/1',
  campo4 char(1) NOT NULL
);
```

Si intentamos añadir registros con valores por defecto:

```
SQL Query Area
1 insert tabla1 values();
```

Esto nos produce un error:

!	Description
!	Field 'campo4' doesn't have a default value

4.2.3.- Colocar un campo de tipo autonumerico

En MySQL tenemos la posibilidad de crear una columna autonumerica.

Esta columna sólo puede ser de tipo entero y debemos colocar la palabra clave AUTO_INCREMENT.

Ejemplo

```
1  CREATE TABLE  `niños`
2  (
3    `id` int(10) NOT NULL auto_increment,
4    `nombre` varchar(45) default NULL,
5    `edad` int(10) default NULL,
6    `nacimiento` date default NULL,
7    `padre` varchar(10) default NULL,
8    `madre` varchar(10) default NULL
9  );
10
```

La definición de la tabla anterior nos produce un error debido a que un columna autonumerica se debe colocar como clave principal.

Line	Description	ErrorNr.
1	Incorrect table definition; there can be only one auto column and it must be defined as a key	1075

```
1  CREATE TABLE  `niños`
2  (
3    `id` int(10) NOT NULL auto_increment,
4    `nombre` varchar(45) default NULL,
5    `edad` int(10) default NULL,
6    `nacimiento` date default NULL,
7    `padre` varchar(10) default NULL,
8    `madre` varchar(10) default NULL,
9    PRIMARY KEY (id)
10 );
11
```

Este definición es correcta ya que hemos colocado el campo como clave principal.

Solamente puede haber una columna autonumerica.

Si al insertar una fila se omite el valor de la columna autoincrementada o si se inserta un valor nulo para esa columna, su valor se calcula automáticamente, tomando el valor más alto de esa columna y sumándole una unidad. Esto permite crear, de una forma sencilla, una columna con un valor único para cada fila de la tabla.

Generalmente, estas columnas se usan como claves primarias 'artificiales'. MySQL está optimizado para usar valores enteros como claves primarias, de modo que la combinación de clave primaria, que sea entera y autoincrementada es ideal para usarla como clave primaria artificial. Esto es igual que en su dia hemos realizado en Access.

4.2.4.- Claves

Dentro de mysql principalmente podemos utilizar los siguientes tipos de claves :

- Indexados con duplicados : INDEX
- Indexados sin duplicados : UNIQUE
- Clave principal : PRIMARY KEY
- Clave ajena : FOREIGN KEY.
- Resto : FULLTEXT y SPATIAL

Para crear una clave indexado con duplicados podemos utilizar las siguientes sintaxis que vemos en los ejemplos.

Ejemplo

Con este ejemplo podemos ver como se crea un índice llamado nombre en el campo nombre.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   `id` INTEGER UNSIGNED NOT NULL,
4   `nombre` VARCHAR(45),
5   `edad` INTEGER UNSIGNED,
6   INDEX (nombre)
7 );
```

Podemos colocar un nombre o alias delante del nombre del campo.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   `id` INTEGER UNSIGNED NOT NULL,
4   `nombre` VARCHAR(45),
5   `edad` INTEGER UNSIGNED,
6   INDEX clave1 (nombre)
7 );
```

También podemos utilizar la cláusula KEY sustituyendo a INDEX.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   `id` INTEGER UNSIGNED NOT NULL,
4   `nombre` VARCHAR(45),
5   `edad` INTEGER UNSIGNED,
6   KEY clave1 (nombre)
7 );
```

Tenemos otra opción que es utilizar solamente parte del campo como clave.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   `id` INTEGER UNSIGNED NOT NULL,
4   `nombre` VARCHAR(45),
5   `edad` INTEGER UNSIGNED,
6    INDEX (nombre(2))
7 );
```

Si queremos colocar mas de un campo como clave.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   `id` INTEGER UNSIGNED NOT NULL,
4   `nombre` VARCHAR(45),
5   `edad` INTEGER UNSIGNED,
6    INDEX (nombre,edad)
7 );
```

Podemos también colocar mas de una definición de claves.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   `id` INTEGER UNSIGNED NOT NULL,
4   `nombre` VARCHAR(45),
5   `edad` INTEGER UNSIGNED,
6    INDEX (nombre),
7    INDEX (edad)
8 );
```

Otro tipo de clave es la que nos permite evitar que existan duplicados. Para definir estas claves tenemos la palabra clave UNIQUE. Veamos su utilización en una serie de ejemplos.

Ejemplo

Para definir una clave indexado sin duplicados.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   UNIQUE (nombre)
7 );
```

Podemos colocar un nombre a la clave.

```
SQL Query Area
1 CREATE TABLE  alumno
2 (
3   id int(10) unsigned NOT NULL,
4   nombre varchar(45) default NULL,
5   edad int(10) unsigned default NULL,
6   clase int(10) unsigned default NULL,
7   UNIQUE NOMBRE (edad)
8 );
```

Podemos utilizar la cláusula UNIQUE KEY de forma similar a UNIQUE y realiza exactamente lo mismo.

```
SQL Query Area
1 CREATE TABLE  alumno
2 (
3   id int(10) unsigned NOT NULL,
4   nombre varchar(45) default NULL,
5   edad int(10) unsigned default NULL,
6   clase int(10) unsigned default NULL,
7   UNIQUE KEY NOMBRE (edad)
8 );
```

Al igual que la anterior podemos definir varias claves en la misma tabla.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   UNIQUE (nombre,edad)
7 );
```

Ante esta definición podemos ver que los siguientes datos son correctos.

id	nombre	edad
1	a	2
2	a	3
3	b	3

El dato incorrecto seria cuando coincidan ambos campos.

Podemos definir también varias claves utilizando diferentes cláusulas.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   UNIQUE (nombre),
7   UNIQUE (edad)
8 );
```

Con esta definición los datos anteriores serian incorrectos ya que ninguno de los campos pueden coincidir.

Podemos crear una clave única con parte de un campo.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   UNIQUE (nombre(2))
7 );
```

Cuando ocurre algo como lo anterior, esto significa que si ocurre una coincidencia de los dos primeros caracteres ya seria incorrecto el segundo dato (por ejemplo Roberto y Roberta).

Otra de las claves que podemos definir son las principales. Como ya sabemos solo puede existir una sola clave principal. Para realizarla utilizamos la clave principal PRIMARY KEY. Recordar que aquellos campos que pertenezcan a la clave primaria son indexados sin duplicados y requeridos.

Veamos su utilización en una serie de ejemplos.

Ejemplo

En el siguiente ejemplo podemos ver como se define una clave principal de un solo campo.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   PRIMARY KEY (id)
7 );
```

Al igual que siempre podemos colocar un alias a la clave.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   PRIMARY KEY alias (id)
7 );
```

Pero en este caso no se utiliza.

Podemos definir varios campos como clave principal.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   PRIMARY KEY (id,nombre)
7 );
```

Column Name	Da
id	
nombre	
edad	

Lo que no podemos realizar es colocar varias cláusulas PRIMARY KEY en la misma tabla.

```
SQL Query Area
1 CREATE TABLE dos
2 (
3   id INTEGER UNSIGNED NOT NULL,
4   nombre VARCHAR(45),
5   edad INTEGER UNSIGNED,
6   PRIMARY KEY (id),
7   PRIMARY KEY (nombre)
8 );
```

!	Description
!	Multiple primary key defined

Bueno vamos ahora a la definición de las claves ajenas o lo que se denomina integridad referencial. Para ello recordamos que debemos tener una base de datos InnoDB.

Es imprescindible que la columna que contiene una definición de clave foránea esté indexada (nos vale un simple indexado con duplicados). Pero esto no debe preocuparnos demasiado, ya que si no lo hacemos de forma explícita, MySQL lo hará por nosotros de forma implícita.

```
SQL Query Area
1  CREATE TABLE  alumno
2  (
3    id int(10) unsigned NOT NULL,
4    nombre varchar(45) default NULL,
5    edad int(10) unsigned default NULL,
6    clase int(10) unsigned default NULL,
7    PRIMARY KEY  (id),
8    KEY ajena1 (clase),
9    FOREIGN KEY (clase)
10   REFERENCES clase (id) ON DELETE CASCADE ON UPDATE CASCADE
11 );
```

Esta forma define una clave foránea en la columna 'clase', que hace referencia a la columna 'id' de la tabla clase. La definición incluye las tareas a realizar en el caso de que se elimine una fila en la tabla 'alumno'.

Las restricciones de integridad que podemos utilizar son :

- RESTRICT: esta opción impide eliminar o modificar filas en la tabla referenciada si existen filas con el mismo valor de clave foránea.
- CASCADE: borrar o modificar una clave en una fila en la tabla referenciada con un valor determinado de clave, implica borrar las filas con el mismo valor de clave foránea o modificar los valores de esas claves foráneas.
- SET NULL: borrar o modificar una clave en una fila en la tabla referenciada con un valor determinado de clave, implica asignar el valor NULL a las claves foráneas con el mismo valor.
- NO ACTION: las claves foráneas no se modifican, ni se eliminan filas en la tabla que las contiene.

- SET DEFAULT: borrar o modificar una clave en una fila en la tabla referenciada con un valor determinado implica asignar el valor por defecto a las claves foráneas con el mismo valor.

Ejemplo

```
SQL Query Area
1  CREATE TABLE  alumno
2  (
3    id int(10) unsigned NOT NULL,
4    nombre varchar(45) default NULL,
5    edad int(10) unsigned default NULL,
6    clase int(10) unsigned default NULL,
7    PRIMARY KEY  (id),
8    KEY ajena1 (clase),
9    FOREIGN KEY (clase)
10   REFERENCES clase (id) ON DELETE RESTRICT ON UPDATE NO ACTION
11 );
```

Podemos definir lo mismo pero sin utilizar la definición explicita de la clave ajena1.

```
SQL Query Area
1  CREATE TABLE  alumno
2  (
3    id int(10) unsigned NOT NULL,
4    nombre varchar(45) default NULL,
5    edad int(10) unsigned default NULL,
6    clase int(10) unsigned default NULL,
7    PRIMARY KEY  (id),
8    FOREIGN KEY (clase)
9    REFERENCES clase (id) ON DELETE RESTRICT ON UPDATE NO ACTION
10 );
```

Podemos utilizar la palabra CONSTRAINT para darle un nombre a la clave ajena (mysql le da un nombre por defecto) de tal forma que asi podemos modificarla mas adelante.

```
SQL Query Area
1  CREATE TABLE  alumno
2  (
3    id int(10) unsigned NOT NULL,
4    nombre varchar(45) default NULL,
5    edad int(10) unsigned default NULL,
6    clase int(10) unsigned default NULL,
7    PRIMARY KEY  (id),
8    CONSTRAINT ajena1 FOREIGN KEY (clase)
9    REFERENCES clase (id) ON DELETE RESTRICT ON UPDATE NO ACTION
10 );
```

4.2.5.- Reglas de validación o CHECKS

En mysql podemos definir las reglas de validación mediante la cláusula CHECK. Recordamos que una regla de validación nos permite restringir los valores a utilizar en un determinado campo.

Veamos un ejemplo.

Ejemplo

```
SQL Query Area
1 CREATE TABLE  primera
2 (
3   id int(10),
4   nombre varchar(45),
5   edad int(11) default '0',
6   fecha date default '0000-00-00',
7   check (edad>18)
8 );
```

Sin embargo a la hora de introducir los datos la regla no funciona.

id	nombre	edad	fecha
1	ramon	5	2005-12-1

Esto es debido a que la cláusula check esta implementada pero aunque la ejecuta no funciona.

Ejemplo

```
SQL Query Area
1 CREATE TABLE  primera
2 (
3   id int(10),
4   nombre varchar(45),
5   edad int(11) default '0',
6   fecha date default '0000-00-00',
7   primary key(edad),
8   check (edad>18)
9 );
```

Podéis crear mas de un check dentro de las tablas y de cualquier campo.

```
SQL Query Area
1  CREATE TABLE  primera
2  (
3    id int(10),
4    nombre varchar(45),
5    edad int(11) default '0',
6    fecha date default '2005-1-2',
7    primary key(edad),
8    check (edad>18),
9    check (fecha>'2005-1-1')
10 );
```

4.2.6.- Aserciones

Recordamos que las aserciones son como las reglas de validación, pero con la diferencia que intervienen varios campos de la tabla.

En mysql se realizan también utilizando la cláusula CHECK. La diferencia es que ahora intervienen varios campos.

Ejemplo

```
SQL Query Area
1 CREATE TABLE  primera
2 (
3   id int(10),
4   nombre varchar(45),
5   edad int(11) default '0',
6   fecha date default '2005-1-2',
7   primary key(edad),
8   check (id>edad)
9 );
```

4.2.7.- Comentarios

En los campos podemos colocar comentarios de forma opcional. El sentido de estos elementos es de apoyo a posibles modificaciones o a facilitar el entendimiento del significado de cada uno de los campos.

Ejemplo

```
1 CREATE TABLE  alumno
2 (
3   id int(10) unsigned NOT NULL default 0 COMMENT 'clave principal',
4   nombre varchar(45) default NULL,
5   edad int(10) unsigned default NULL,
6   clase int(10) unsigned default NULL,
7   PRIMARY KEY  (id),
8   UNIQUE KEY NOMBRE (edad)
9 );
```

Podemos colocar también comentarios a la tabla. Esto lo haremos después de la definición de esta.

```
1  CREATE TABLE  alumno
2  (
3    id int(10) unsigned NOT NULL default 0 COMMENT 'clave principal',
4    nombre varchar(45) default NULL,
5    edad int(10) unsigned default NULL,
6    clase int(10) unsigned default NULL,
7    PRIMARY KEY  (id),
8    UNIQUE KEY NOMBRE (edad)
9  )COMMENT 'Esta tabla es de ejemplo';
10
11
12
```

4.3.- Motores de almacenamiento

Después de la definición de la tabla podemos colocar el motor de almacenamiento que vamos a utilizar para almacenar la base de datos. Por defecto utilizara el que este definido en el servidor.

La definición se hará con la siguiente sintaxis:

ENGINE=nombre

Los motores de almacenamiento que tenemos disponibles son los siguientes:

Tipo de tabla	Descripción
BDB o BerkeleyDB	Tablas a prueba de transacciones con el bloqueo de páginas.
HEAP	Los datos para esta tabla sólo se almacenan en memoria.
ISAM	El manejador original de mysql
InnoDB	Tablas a prueba de transacciones con bloqueo de registros y claves foraneas
MERGE	Una colección de tablas MyISAM utilizadas como una tabla.
MyISAM	El nuevo manejador de la tabla binaria que está sustituyendo ISAM.

Generalmente usaremos tablas MyISAM o tablas InnoDB. Nosotros en clase todas las tablas generadas las hemos creado con InnoDB.

Ejemplo

```
1  CREATE TABLE  alumno
2  (
3    id int(10) unsigned NOT NULL default 0 COMMENT 'clave principal',
4    nombre varchar(45) default NULL,
5    edad int(10) unsigned default NULL,
6    clase int(10) unsigned default NULL,
7    PRIMARY KEY  (id),
8    UNIQUE KEY NOMBRE (edad)
9  )ENGINE=InnoDB COMMENT 'Esta tabla es de ejemplo';
10
11
12
13
14
```

Ejercicio 1

Quiero que realicéis el script de creación del siguiente esquema y lo implementáis en mysql.

Estación(Identificador, Latitud, Longitud, Altitud)

Muestra(IdentificadorEstacion, Fecha, Temperatura mínima, Temperatura máxima, Precipitaciones, Humedad mínima, Humedad máxima, Velocidad del viento mínima, Velocidad del viento máxima)

Los dominios son :

Columna	Tipo
Identificador	MEDIUMINT UNSIGNED
Latitud	VARCHAR(14)
Longitud	VARCHAR(15)
Altitud	MEDIUMINT
IdentificadorEstacion	MEDIUMINT UNSIGNED
Fecha	DATE
Temperatura mínima	TINYINT
Temperatura máxima	TINYINT
Precipitaciones	SMALLINT UNSIGNED
Humedad mínima	TINYINT UNSIGNED
Humedad máxima	TINYINT UNSIGNED
Velocidad del viento mínima	SMALLINT UNSIGNED
Velocidad del viento máxima	SMALLINT UNSIGNED

La clave ajena se encuentra en muestra y es Identificador de estación.

Ejercicio 2

Quiero que realicéis el script de creación del siguiente esquema y lo implementáis en mysql

Libro(ClaveLibro, Título, Idioma, Formato, Categoría, ClaveEditorial)

Tema(ClaveTema, Nombre)

Autor(ClaveAutor, Nombre)

Editorial(ClaveEditorial, Nombre, Dirección, Teléfono)

Ejemplar(ClaveLibro, NúmeroOrden, Edición, Ubicación)

Socio(ClaveSocio, Nombre, Dirección, Teléfono, Categoría)

Trata_sobre(ClaveLibro, ClaveTema)

Escrito_por(ClaveLibro, ClaveAutor)

Colocar vosotros las restricciones, los dominios y las claves ajenas.

4.4.- Creación a través de una consulta

Mediante este tipo de consultas podemos generar una tabla a traves de una consulta. Para realizarlo colocamos un select detrás del create table.

CREATE TABLE ……..

[As] SELECT …….;

La palabra as es opcional por lo tanto nosotros no la colocaremos.

El formato mas sencillo es crear una tabla integra con una consulta.

Ejemplo

Tenemos la tabla editorial con los siguientes datos y estructura.

```
SQL Query Area
1 SELECT * FROM editorial e
```

id	nombre	localidad
1	editorial1	11
2	editorial3	12
3	editorial2	13
4	editorial4	14
5	editorial5	11
6	editorial6	12
7	editorial7	13

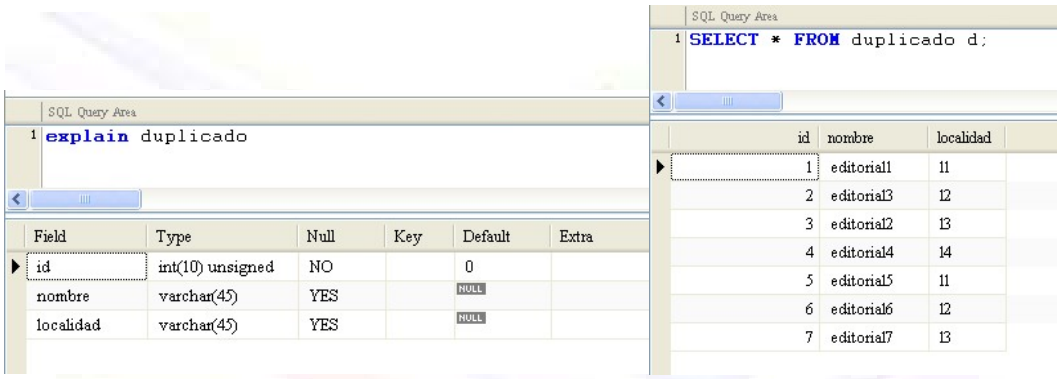
```
SQL Query Area
1 explain editorial;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES		NULL	
localidad	varchar(45)	YES		NULL	

Vamos a crear una tabla llamada duplicado con la tabla editorial.

```
SQL Query Area
1 create table duplicado
2 select * from editorial;
```

Comprobemos los datos y la estructura de la nueva tabla.



```
explain duplicado
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO		0	
nombre	varchar(45)	YES		NULL	
localidad	varchar(45)	YES		NULL	

```
SELECT * FROM duplicado d;
```

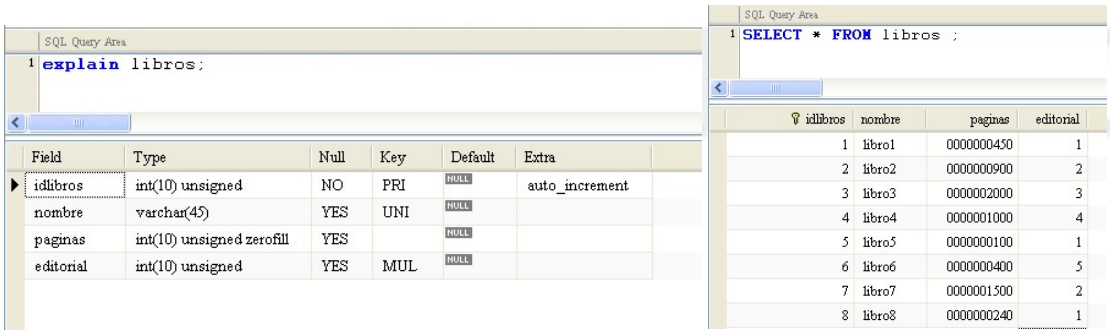
id	nombre	localidad
1	editorial1	11
2	editorial3	12
3	editorial2	13
4	editorial4	14
5	editorial5	11
6	editorial6	12
7	editorial7	13

Como podemos observar los índices en la tabla destino no se han creado.

CREATE TABLE...SELECT no creará automáticamente índices por ti. Esto se hace intencionadamente para hacer que el comando sea tan flexible como sea posible. ***Si quieres tener índices en la creación de tabla, deberías especificarlos antes de la sentencia SELECT:***

Ejemplo

Tenemos la tabla libros con la siguiente estructura y datos.



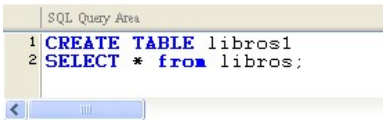
```
explain libros;
```

Field	Type	Null	Key	Default	Extra
idlibros	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES	UNI	NULL	
paginas	int(10) unsigned zerofill	YES		NULL	
editorial	int(10) unsigned	YES	MUL	NULL	

```
SELECT * FROM libros ;
```

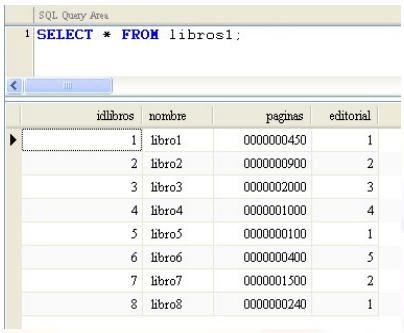
idlibros	nombre	paginas	editorial
1	libro1	0000000450	1
2	libro2	0000000900	2
3	libro3	0000002000	3
4	libro4	0000001000	4
5	libro5	0000000100	1
6	libro6	0000000400	5
7	libro7	0000001500	2
8	libro8	0000000240	1

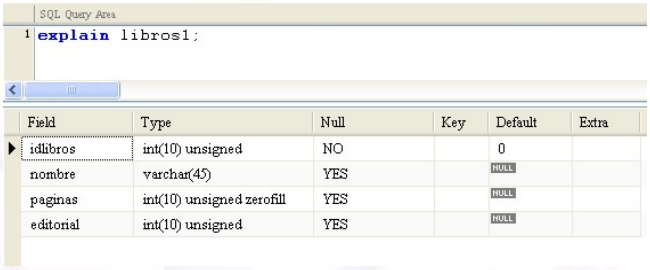
Realizamos el siguiente comando.



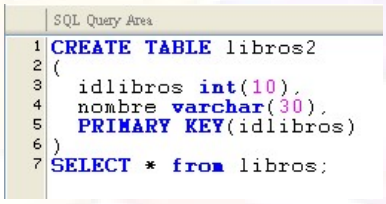
```
CREATE TABLE libros1
SELECT * from libros;
```

La tabla libros1 tendria.

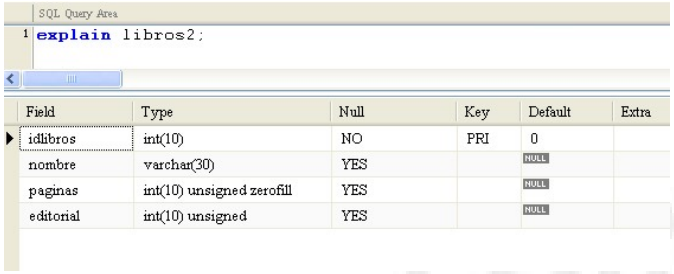


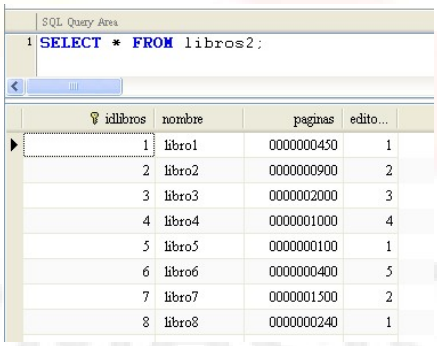


Si realizamos el siguiente comando:



De esta forma el campo idlibros será la clave principal en la tabla libros2.





Comprobar que los campos idlibros y nombre no se crean, simplemente los rellena con la salida del SELECT.

Con este ejemplo podemos explicar el segundo formato del comando, cuando la tabla ya tiene campos generados, si la tabla ya contiene campos estos se rellenaran con los datos del select si el nombre de los campos coinciden. Los tipos y claves de la tabla seran los colocados en la sentencia CREATE. Por ejemplo en el caso anterior la tabla libros2 tiene una clave principal en el campo idlibros y el campo nombre es un VARCHAR(30) y no un VARCHAR(45) como en la tabla libros.

Los campos que no se rellenen con la sentencia SELECT se colocaran con los valores por defecto.

Ejemplo

Supongamos el siguiente comando.

```
SQL Query Area
1 CREATE TABLE libros1
2 (
3   idlibros int(10),
4   nombre varchar(30),
5   tipo_tapas varchar(10) DEFAULT 'duras',
6   ISBN int(10),
7   PRIMARY KEY(idlibros)
8 )
9 SELECT * from libros;
```

Ahora observemos la tabla libros1.

```
SQL Query Area
1 SELECT * FROM libros1
```

tipo_tapas	ISBN	idlibros	nombre	paginas	editorial
duras	NULL	1	libro1	0000000450	1
duras	NULL	2	libro2	0000000900	2
duras	NULL	3	libro3	0000002000	3
duras	NULL	4	libro4	0000001000	4
duras	NULL	5	libro5	0000000100	1
duras	NULL	6	libro6	0000000400	5
duras	NULL	7	libro7	0000001500	2
duras	NULL	8	libro8	0000000240	1

```
SQL Query Area
1 explain libros1;
```

Field	Type	Null	Key	Default	Extra
tipo_tapas	varchar(10)	YES		duras	
ISBN	int(10)	YES		NULL	
idlibros	int(10)	NO	PRI	0	
nombre	varchar(30)	YES		NULL	
paginas	int(10) unsigned zerofill	YES		NULL	
editorial	int(10) unsigned	YES		NULL	

Cuidado cuando los tipos de datos no coincidan entre los datos introducidos y los campos existentes. Mysql normalmente te dara una serie de avisos (warnings) pero te creara la tabla.

Ejemplo

```
SQL Query Area
1 CREATE TABLE libros2
2 (
3   idlibros int(10),
4   nombre int(10),
5   tipo_tapas varchar(10) DEFAULT 'duras',
6   ISBN int(10),
7   PRIMARY KEY(idlibros)
8 )
9 SELECT * from libros;
```

El comando te dara los siguientes avisos.

!	Description
!	Out of range value adjusted for column 'nombre' at row 1
!	Out of range value adjusted for column 'nombre' at row 2
!	Out of range value adjusted for column 'nombre' at row 3
!	Out of range value adjusted for column 'nombre' at row 4
!	Out of range value adjusted for column 'nombre' at row 5
!	Out of range value adjusted for column 'nombre' at row 6
!	Out of range value adjusted for column 'nombre' at row 7
!	Out of range value adjusted for column 'nombre' at row 8

La tabla libros2 tendrá la siguiente estructura y los siguientes datos.

```
SQL Query Area
1 Explain libros2;
```

Field	Type	Null	Key	Default	Extra
tipo_tapas	varchar(10)	YES		duras	
ISBN	int(10)	YES		NULL	
idlibros	int(10)	NO	PRI	0	
nombre	int(10)	YES		NULL	
paginas	int(10) unsigned zerofill	YES		NULL	
editorial	int(10) unsigned	YES		NULL	

```
SQL Query Area
1 SELECT * FROM libros2;
```

tipo_tapas	ISBN	idlibros	nombre	paginas	editorial
duras	NULL	1	0	0000000450	1
duras	NULL	2	0	0000000900	2
duras	NULL	3	0	0000002000	3
duras	NULL	4	0	0000001000	4
duras	NULL	5	0	0000000100	1
duras	NULL	6	0	0000000400	5
duras	NULL	7	0	0000001500	2
duras	NULL	8	0	0000000240	1

4.5.- Creación de una tabla con la estructura de otra

Podemos en MySQL crear una tabla con la estructura de otra, sin copiar los datos. La sintaxis del comando es :

SELECT nueva_tabla LIKE tabla_ya_creada;

La estructura de la nueva tabla es igual que la vieja pero no tiene los datos.

Ejemplo

Ejecutamos el siguiente comando.

```
SQL Query Area
1 CREATE TABLE e1 LIKE libros;
```

La tabla e1 tendra lo siguiente:

```
SQL Query Area
1 describe e1
```

Field	Type	Null	Key	Default	Extra
idlibros	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES	UNI	NULL	
paginas	int(10) unsigned zerofill	YES		NULL	
editorial	int(10) unsigned	YES	MUL	NULL	

```
SQL Query Area
1 SELECT * FROM e1;
```

idlibros	nombre	paginas	editorial

5.- Comando ALTER TABLE

Este comando nos permite modificar la estructura de una tabla que ya hemos generado anteriormente, sin necesidad de destruirla y volverla a crear.

ALTER TABLE permite modificar la estructura de una tabla existente. Por ejemplo, se pueden añadir liminar columnas, crear y destruir índices, cambiar el tipo de una columna existente o renombrar columnas o la propia tabla. También es posible modificar el comentario y el tipo de la tabla.

Si después de realizar una modificación, comprobamos y resulta que no se ha realizado, puede ser un fallo del servidor. Será necesario volverla a realizar.

ALTER TABLE trabaja haciendo una copia temporal de la tabla original. La modificación se realiza durante la copia, a continuación la tabla original se borra y la nueva se renombra. Esto se hace para realizar que todas las actualizaciones se dirijan a la nueva tabla sin ningún fallo de actualización. Mientras ALTER TABLE se ejecuta, la tabla original permanece accesible en lectura para otros clientes. Las actualizaciones y escrituras en la tabla se retrasan hasta que la nueva tabla esté preparada.

La sintaxis de este comando es la siguiente:

ALTER [IGNORE] TABLE table_name modificacion [, modificacion...]

La forma de especificar las modificaciones es :

ADD [COLUMN] create_definition [FIRST|AFTER column_name]

ó ADD [COLUMN] (create_definition,create_definition,...)

ó ADD INDEX [index_name] (index_col_name,...)

ó ADD PRIMARY KEY (index_col_name,...)

ó ADD UNIQUE [index_name] (index_col_name,...)

ó ADD FULLTEXT [index_name] (index_col_name,...)

ó ADD [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...) [reference_definition]

ó ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}

ó CHANGE [COLUMN] old_col_name, create_definition

[FIRST | AFTER column_name]

- ***MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]***
- ***DROP [COLUMN] col_name***
- ***DROP PRIMARY KEY***
- ***DROP FOREIGN KEY***
- ***DROP INDEX index_name***
- ***DISABLE KEYS***
- ***ENABLE KEYS***
- ***RENAME [TO] new_table_name***
- ***ORDER BY col***
- ***table_options***

IGNORE es una extensión MySQL a SQL-92. Controla el modo de trabajar de ALTER TABLE si hay claves duplicadas o únicas en la nueva tabla. Si no se especifica IGNORE, la copia se aborta y se deshacen los cambios. Si se especifica IGNORE, en las filas duplicadas en una clave única sólo se copia la primera fila; el resto se eliminan.

Se pueden usar múltiples claúsulas ADD, ALTER, DROP y CHANGE en una sola sentencia ALTER TABLE. Esto es una extensión MySQL a SQL, que permite sólo una aparición de cada cláusula en una sentencia ALTER TABLE.

Como nos paso con las sentencia CREATE TABLE tenemos un comando con un numero de opciones bastante elevado. Por ello vamos a dividirlo en varios bloques según la funcion.

5.1.- Añadir nuevas columnas o campos

Para poder añadir una columna utilizamos la siguiente sintaxis.

ALTER [IGNORE] TABLE ADD [COLUMN] create_definition [FIRST|AFTER column_name] | ADD [COLUMN] (create_definition,create_definition,...)

La palabra COLUMN es opcional.

Veamos algún ejemplo para entender este comando.

Ejemplo

Veamos la estructura de la tabla libros.

```
SQL Query Area
1 SHOW COLUMNS FROM libros;
```

Field	Type	Null	Key	Default	Extra
idlibros	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES	UNI	NULL	
paginas	int(10) unsigned zerofill	YES		NULL	
editorial	int(10) unsigned	YES	MUL	NULL	

Ahora vamos a añadir un campo llamado tipo_tapas.

```
SQL Query Area
1 ALTER TABLE libros ADD COLUMN
2 tipo_tapas varchar(20) DEFAULT 'blandas';
```

```
SQL Query Area
1 ALTER TABLE libros ADD
2 tipo_tapas varchar(20) DEFAULT 'blandas';
```

El campo por defecto se coloca al final. Si deseas colocarlo al principio o después de un campo.

```
SQL Query Area
1 ALTER TABLE libros ADD tipo_tapas varchar(20) FIRST;
```

```
SQL Query Area
1 ALTER TABLE libros ADD tipo_tapas varchar(20) AFTER nombre;
```

Ejemplo

Vamos a ver como podemos generar mas de un campo.

```
SQL Query Area
1 ALTER TABLE libros ADD (tipo_tapas varchar(20), ISBN decimal(10));
```

```
SQL Query Area
1 describe libros;
```

Field	Type	Null	Key	Default	Extra
idlibros	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES	UNI	NULL	
paginas	int(10) unsigned zerofill	YES		NULL	
editorial	int(10) unsigned	YES	MUL	NULL	
tipo_tapas	varchar(20)	YES		NULL	
ISBN	decimal(10,0)	YES		NULL	

También podemos realizarlo de la siguiente forma:

```
SQL Query Area
1 ALTER TABLE libros ADD (tipo_tapas varchar(20)),ADD ISBN decimal(10);
```

5.2.- Eliminación de una columna o campo

Vamos a ver como podemos borrar uno o mas campos de nuestras tablas.

La sintaxis del comando es bastante sencilla:

ALTER [IGNORE] TABLE table_name DROP [COLUMN] col_name

Como podéis observar en la sintaxis la palabra COLUMN es opcional.

Ejemplo

Tenemos la siguiente tabla.

```
SQL Query Area
1 explain editorial
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES		NULL	
localidad	varchar(45)	YES		NULL	
poblacion	varchar(10)	YES		NULL	

Borremos una determinada columna.

```
SQL Query Area
1 ALTER TABLE editorial DROP poblacion;
```

Si deseamos borrar dos campos:

```
SQL Query Area
1 ALTER TABLE editorial DROP poblacion, DROP localidad;
```

5.3.- Modificación de la definición de una columna o campo

Ahora suponeros que deseamos modificar la estructura de un campo, sin perder los datos que contiene (lógicamente dependerá de la nueva definición de datos).

La sintaxis del comando será la siguiente:

ALTER [IGNORE] TABLE table_name] ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT} | CHANGE [COLUMN] old_col_name, create_definition [FIRST|AFTER column_name]|MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]

Cuando se cambia un tipo de columna usando CHANGE o MODIFY, MySQL intenta convertir datos al nuevo tipo lo mejor posible

---

Ejemplo

Tenemos la siguiente tabla.

```
SQL Query Area
1 explain alumno
```

Field	Type	Null	Key	Default
id	int(10) unsigned	NO	PRI	0
nombre	varchar(45)	YES		NULL
edad	int(10) unsigned	YES	UNI	NULL
clase	int(10) unsigned	YES		NULL

Con este comando colocamos un valor por defecto para el campo clase.

```
SQL Query Area
1 ALTER TABLE alumno ALTER clase SET DEFAULT 2;
```

Con el siguiente quitamos el valor por defecto.

```
SQL Query Area
1 ALTER TABLE alumno ALTER clase DROP DEFAULT;
```

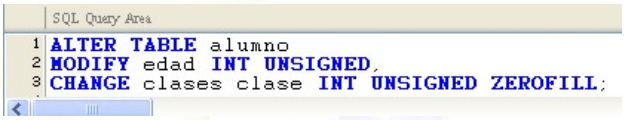
Por lo tanto ALTER COLUMN especifica un nuevo valor por defecto para una columna o elimina el valor por defecto anterior. Si se elimina el anterior valor por defecto y la columna puede ser NULL, el nuevo valor por defecto es NULL. Si la columna no puede ser NULL, MySQL asigna un nuevo valor por defecto.

Ejemplo

Con este comando podemos cambiar el nombre de un campo y ademas el tipo de datos.

```
SQL Query Area
1 ALTER TABLE alumno CHANGE clase clases INTEGER UNSIGNED;
2
```

Podemos cambiar mas de un campo con un solo comando.



Las clausulas CHANGE y MODIFY son prácticamente iguales, la principal diferencia es ***que CHANGE se utiliza cuando se quiere cambiar el nombre del campo.***

Cuando se cambia un tipo de columna usando CHANGE o MODIFY, MySQL intenta convertir datos al nuevo tipo lo mejor posible.

5.4.- Renombrar una tabla

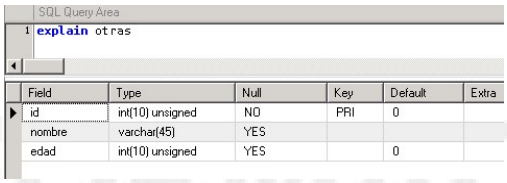
Este comando me permite realizar lo mismo que el RENAME TABLE.

La sintaxis del comando es:

ALTER [IGNORE] TABLE table_name RENAME [TO] new_table_name

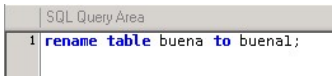
Ejemplo

Tenemos la siguiente tabla



Vamos a cambiar el nombre :

Podemos realizarlo también de la siguiente forma:



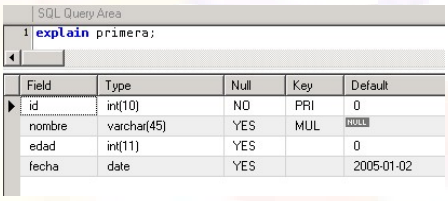
5.5.- Borrado de claves

Para borrar las claves utilizaremos la siguiente sintaxis :

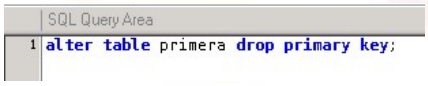
ALTER [IGNORE] TABLE table_name DROP PRIMARY KEY|DROP INDEX index_name|DROP FOREIGN KEY index_name

Con la cláusula primary key borramos la clave principal y con la cláusula INDEX borramos las claves únicas, múltiples. Para borrar claves ajenas utilizamos FOREIGN KEY.

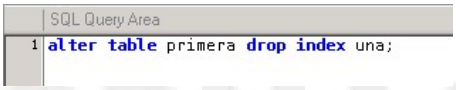
Ejemplo



Para poder borrar una clave primaria.



Para borra la clave múltiple (indexado con duplicados)



5.6.- Creación de claves

Para crear claves utilizaremos la siguiente sintaxis :

ALTER [IGNORE] TABLE table_name ADD INDEX [index_name] (index_col_name,...)|ADD PRIMARY KEY (index_col_name,...)|ADD UNIQUE [index_name] (index_col_name,...)|ADD FULLTEXT [index_name] (index_col_name,...)|ADD [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...) [reference_definition]

El funcionamiento del comando es similar al de CREATE TABLE simplemente se indica el tipo de clave que deseas crear y a continuación el alias que quieres colocar.

Ejemplo

Tenemos la siguiente base de datos:

```
SQL Query Area
1 explain alumno;
```

Field	Type	Null	Key	Default
id	int(10) unsigned	NO	PRI	0
nombre	varchar(45)	YES		NULL
edades	int(20)	YES	UNI	NULL
clase	int(20)	YES		NULL

Si queremos crear una clave.

```
SQL Query Area
1 ALTER TABLE alumno
2 ADD INDEX launo (nombre);
```

Field	Type	Null	Key	Default
id	int(10) unsigned	NO	PRI	0
nombre	varchar(45)	YES	MUL	NULL
edades	int(20)	YES	UNI	NULL
clase	int(20)	YES		NULL

Borramos todas las claves.

```
SQL Query Area
1 ALTER TABLE alumno
2 DROP PRIMARY KEY, DROP INDEX launo, DROP INDEX nombre;
```

Si deseamos crearlas de nuevo.

```
SQL Query Area
1 ALTER TABLE alumno
2 ADD PRIMARY KEY (id), ADD INDEX launo (nombre), ADD UNIQUE lados (edades);
```

6.- *Sentencia DROP INDEX*

Con este comando podemos borrar las claves que existan en una determinada tabla. La sintaxis es la siguiente :

DROP INDEX nombre_indice ON nombre_tabla;

Ejemplo

Tenemos la siguiente tabla.

```
SQL Query Area
1 explain persona;
```

Field	Type	Null	Key	Default
NIF	varchar(10)	NO	PRI	
Nombre	varchar(20)	NO	MUL	
Fecha_nacimiento	date	YES		NULL
Poblacion	varchar(20)	YES		NULL

Borremos todas las claves:

- Borro la clave múltiple (indexado sin duplicados)

```
SQL Query Area
1 drop index nombre on persona;
```

- Borro la clave principal

```
SQL Query Area
1 DROP INDEX `PRIMARY` on persona
```

Fijaros que para colocar el nombre de la clave lo coloco entre acentos porque si no tomaría la palabra como cláusula y daría error.

Recordar que todas esta operaciones las podemos realizar con la sentencia ALTER TABLE.

```
SQL Query Area
1 ALTER TABLE persona DROP PRIMARY KEY;
```

Con este comando no podemos borrar claves foráneas.

7.- Comando Create index

Con ayuda de esta sentencia podemos generar indices en la tabla. Nosotros utilizaremos normalmente ALTER TABLE porque nos permite una mayor flexibilidad y nos permite realizar mas operaciones.

Su sintaxis es la siguiente:

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name ON tbl_name (index_col_name,...)

Con este comando no podemos generar claves ajenas.

La sentencia CREATE INDEX no hace nada en las versiones anteriores a la 3.22. En la versión 3.22 o posterior, CREATE INDEX se dirige a una sentencia ALTER TABLE para crear índices.

Normalmente, puedes crear todos los índices en una tabla en el momento en el que la tabla es creada con CREATE TABLE. CREATE INDEX te permite añadir índices a tablas existentes.

Una lista de columnas de la forma (col1,col2,...) crea un índice de múltiples columnas. Los valores de los índices se crean concatenando los valores de las columnas dadas.

Para las columnas CHAR y VARCHAR, los índices pueden crearse utilizando sólo una parte de la columna, utilizando la sintaxis col_name(length). Utilizar columnas parciales para índices puede hacer que el archivo de índice tenga un tamaño mucho menor, lo que podría salvar un montón de disco, y acelerar además las operaciones de inserción.

Ejemplo

SQL Query Area

```
1 describe editorial
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	YES		NULL	

Ahora vamos a crear una clave única en nombre.

SQL Query Area

```
1 create unique index mia on editorial (nombre);
```

Si borramos ambas claves.

- Para borrar la clave unica

SQL Query Area

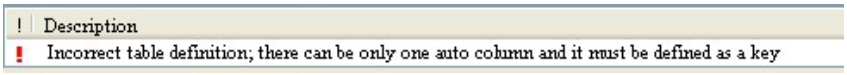
```
1 alter table editorial drop index mia
```

- Para borrar la clave primaria

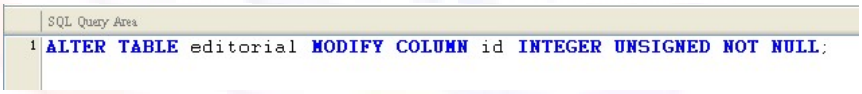
SQL Query Area

```
1 alter table editorial drop primary key;
```

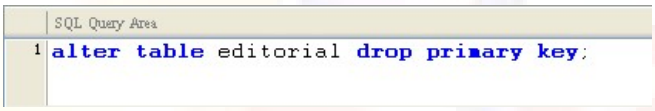
El problema que nos indica es que el campo id, que es clave principal, es de tipo autonumerico. Por lo tanto no nos permite eliminar la clave principal porque un campo autonumerico debe ser clave principal.



Modificamos el tipo de campo.



Ahora podemos borrar la clave principal sin ningún tipo de problemas.



8.- *Las vistas en MYSQL*

Una vista es una consulta, cuya formulación, no su resultado, se almacena en el catalogo dándole un nombre. Para que os pueda ayudar seria similar a lo que podemos conseguir en Access con la pestaña consultas. Puesto que el resultado de ejecutar un SELECT será una tabla, nada impide que se puedan especificar consultas sobre ella, para lo cual basta con mencionar su nombre en la cláusula FROM de otra sentencia SELECT, como si de cualquier tabla se tratase.

Una vista en SQL se define utilizando la orden create view. Para definir una vista se le debe dar un nombre y se debe construir la consulta que genere dicha vista.

Ejemplo

Tenemos la siguiente consulta.

```
SQL Query Area
1 SELECT * FROM editorial;
```

id	nombre
1	editorial1
3	editorial2
2	editorial3
4	editorial4
5	editorial5
6	editorial6
7	editorial7

Suponeros que queremos almacenar esta consulta en la base de datos sin ejecutarla. La forma de realizarlo seria utilizando una vista. El comando sera el siguiente

```
SQL Query Area
1 CREATE VIEW CONSULTA1 AS SELECT * FROM editorial;
```

Este comando no devolverá ningún resultado, simplemente almacena la vista en la base de datos. Podemos observarlo en el propio cliente.


```
SQL Query Area
1 SHOW TABLES FROM a;
```

Tables_in_a
consulta1
el
editorial
editorial1
libros

Para esto ultimo podemos utilizar el comando SHOW TABLES

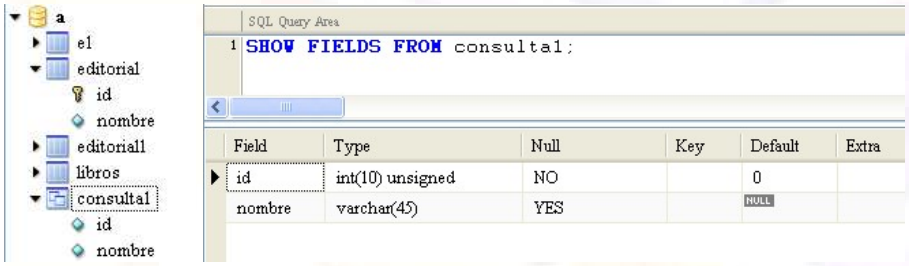
El problema con este comando es que no podemos distinguir que elementos son tablas y cuales son vistas.

Para solucionarlo utilizaremos el comando SHOW FULL TABLES

```
SHOW FULL TABLES FROM a;
```

Tables_in_a	Table_type
consulta1	VIEW
e1	BASE TABLE
editorial	BASE TABLE
editorial1	BASE TABLE
libros	BASE TABLE

La vista es como si fuera una tabla.



Para ejecutar esta vista simplemente podemos utilizarla como si fuera una tabla normal.

```
SELECT * FROM consulta1;
```

id	nombre
1	editorial1
3	editorial2
2	editorial3
4	editorial4
5	editorial5
6	editorial6
7	editorial7

8.1.- Sentencia CREATE VIEW

Mediante este comando podemos crear vistas. La sintaxis de la orden create view es la siguiente:

CREATE VIEW view_name [(column_list)] AS select_statement

En una vista no podemos utilizar SELECT anidados en la cláusula FROM:

```
SQL Query Area
1 CREATE VIEW CONSULTA2 AS SELECT nombre FROM (SELECT * FROM EDITORIAL) D;
```

Si podemos utilizar SELECT anidados en la cláusula SELECT:

```
SQL Query Area
1 VIEW CONSULTA2 AS SELECT (SELECT min(id) FROM editorial) uno,(SELECT max(id) FROM editorial) dos;
```

Ejemplo

Suponeros que tenemos el siguiente esquema:

estructura coches | estructura personas | estructura comprados

```
SQL Query Area
1 describe coches;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	NO			

estructura coches | estructura personas | estructura comprados

```
SQL Query Area
1 describe personas;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(45)	NO			
edad	int(3) unsigned	NO			

estructura coches | estructura personas | estructura comprados

```
SQL Query Area
1 describe comprados;
```

Field	Type	Null	Key	Default	Extra
persona	int(10) unsigned	NO	PRI		
coche	int(10) unsigned	NO	PRI		

Creamos las claves ajenas en la tabla comprados.

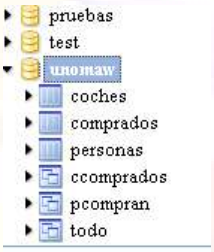
Vamos a ver como podemos almacenar tres vistas que a la larga nos pueden facilicitar y agilizar el trabajo con este esquema.

```
SQL Query Area
1 create view ccomprados as select * from coches join comprados on id=coche;
```

```
SQL Query Area
1 create view pcompran as select * from personas join comprados on id=persona;
```

```
SQL Query Area
1 create view todo as
2 select co.nombre coche,pe.nombre persona,edad from personas pe join comprados on pe.id=persona
3 join coches co on co.id=coche;
```

Estas vistas aparecerán como una tabla mas en el cliente, pero su icono será diferente.



Si por ejemplo necesitaríamos saber cuantos coches existen seria

```
SQL Query Area
1 SELECT count(*) FROM coches;
```

Para saber cuantos coches se han comprado:

```
SQL Query Area
1 SELECT count(*) FROM todo;
```

También podría ser:

```
SQL Query Area
1 SELECT count(*) FROM ccomprados;
```

¿Qué es lo que realizan las siguientes consultas?

```
SQL Query Area
1 SELECT count(*) FROM pcompran;
```

```
SQL Query Area
1 SELECT (SELECT count(*) FROM pcompran),(SELECT count(*) FROM ccomprados);
```

¿Las siguientes consultas de creación de vistas son correctas?

```
SQL Query Area
1 CREATE VIEW una AS SELECT (SELECT count(*) FROM pcompran),(SELECT count(*) FROM ccomprados);
```

```
SQL Query Area
1 CREATE VIEW una AS SELECT (SELECT * FROM (SELECT count(*) FROM ccomprados)alias);
```

```
SQL Query Area
1 CREATE VIEW unica AS (UPDATE coches SET nombre='ford' WHERE nombre='fort')
```

8.2.- Sentencia DROP VIEW

Con este comando podemos borrar una determinada vista. Su sintaxis es la siguiente:

DROP VIEW [IF EXISTS] view_name [, view_name] ...;

Ejemplo

```
SQL Query Area
1 DROP VIEW consulta2;
```

8.3.- Modificar una vista

Cuando se ha creado una vista tenemos un comando que nos permite visualizar la sentencia que ha generado la vista. Este comando tiene la siguiente sintaxis:

SHOW CREATE VIEW view_name

Ejemplo

Tenemos una vista denominada consulta1. Observemos como funciona el comando explicado.

```
SQL Query Area
1 SHOW CREATE VIEW consulta1;
```

View	Create View
consulta1	CREATE VIEW "consulta1" AS select "editorial"."id" AS "id","editorial"."nombre" AS "nombre" from "editorial"

Si queremos modificar una vista disponemos también de un comando. Veamos su sintaxis:

SELECT OR REPLACED VIEW view_name [(column_list)] AS select_statement

---

Ejemplo

```
SQL Query Area
1 CREATE OR REPLACE VIEW consulta1 AS SELECT id FROM editorial;
```

Con este comando creamos una vista denominada consulta1 si no existe y si existe la modificamos.