
	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Tema 6

Creación, borrado y modificación de tablas

Índice

1.	Introducción	1
2.	Creación de Bases de Datos	1
3.	Borrado de bases de datos	2
4.	Tipos de datos.....	2
4.1.	Tipos de datos numéricos	3
4.2.	Tipos de datos cadena	4
4.3.	Tipos de dato Fecha y Hora	5
5.	Creación tablas	6
6.	Definición de columnas.....	7
6.1.	Obligatoriedad. La restricción de valores nulos	7
6.2.	Valores por defecto	7
6.3.	Claves primarias. La restricción PRIMARY KEY	7
6.4.	La restricción UNIQUE.....	8
6.5.	Comentarios	8
6.6.	Campos autoincrementados	8
7.	Borrado de tablas.....	9
8.	Cambios de nombre de tablas: Rename	9
9.	Copiado de tablas	9
10.	Definición de creación.....	9
10.1.	Índices	10
10.1.1.	Claves primarias.....	10
10.1.2.	Índices.....	10
10.1.3.	Claves únicas	10
10.2.	Claves foráneas.....	11
10.3.	Opciones de tabla	13
11.	Modificación de tablas	15
11.1.	Cambiar el nombre de una tabla. RENAME.....	15
11.2.	Añadir columnas. ADD.	15
11.3.	Modificación de la definición de una columna. CHANGE.	15
11.4.	Eliminar una columna. DROP.....	15
11.5.	Modificación de las claves primarias	16
11.6.	Modificación de las claves únicas	16
11.7.	Modificación de índices	16
11.8.	Modificación de claves foráneas	16
11.9.	Modificación del conjunto de caracteres y de las colecciones.....	16
11.10.	Cambiar el motor de almacenamiento	16
11.11.	Concatenación de modificaciones	16

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

1. Introducción

Para la creación, modificación y borrado de estructuras de la base de datos se usan respectivamente las instrucciones CREATE, DROP y ALTER que forman parte de la parte del lenguaje SQL denominado DDL o Lenguaje de Definición de Datos.

2. Creación de Bases de Datos

Una base de datos es la estructura básica que contiene los demás elementos que forman parte de esta: tablas, vistas, procedimientos almacenados, ...

Para crear un elemento dentro de una base de datos se usa el comando CREATE seguido del tipo de elemento que queremos crear.

En este caso como queremos crear una base de datos su sintaxis es la siguiente (recordar que los elementos entre corchetes son opcionales y de aparecer se deben eliminar):

```
CREATE [OR REPLACE] {DATABASE | SCHEMA} [IF NOT EXISTS] nombreBaseDeDatos
[[DEFAULT] CHARACTER SET [=] charset_name]
[[DEFAULT] COLLATE [=] collation_name]
```

Donde:


- CREATE: acción de creación de un elemento, en nuestro caso será una base de datos.
- OR REPLACE: si al crear la base de datos esta ya existe se elimina y se crea vacía. Disponible solo para MariaDB desde la versión 10.0.1. Es opcional.
- DATABASE o SCHEMA: después de la clausula CREATE se debe indicar el tipo de elemento a crear. En este caso una base de datos donde las clausulas DATABASE y SCHEMA, en MySQL/MariaDB, son sinónimas.
- IF NOT EXISTS: esta clausula es opcional y no pertenece al SQL estándar. Si se intenta crear una base de datos que ya existe, y no estamos usando ON REPLACE, se produce un error. Esta clausula permite comprobar que la base de datos a crear no existe para crearla únicamente en este caso. Si ya existiese generaría un Warning en vez de un error. Es muy útil para su utilización en scripts. No es compatible con OR REPLACE.
- nombreBaseDeDatos: nombre de la base de datos que se desea crear. Este nombre debe ser único dentro del mismo SGBD. En plataformas Linux el nombre es sensible al contexto.
- CHARACTER SET: conjunto de caracteres usados para representar los campos de tipo texto. Se puede consultar los posibles valores con el comando [SHOW CHARACTER SET](#) (se pueden filtrar los resultados usando like). Es opcional, si no se especifica se toma como valor el character set establecido por defecto en el SGBD.
- COLLATE: reglas para comparar y ordenar cadenas de texto usando un conjunto de caracteres concreto. Se puede consultar los posibles valores con el comando [SHOW COLLATION](#) (se pueden filtrar los resultados usando like). Es opcional, si no se especifica se toma como valor el collation establecido por defecto en el SGBD.
- DEFAULT: las tablas creadas dentro de esta base de datos usarán, si no se especifica otro, el character set y/o el collation indicado para sus columnas de tipo texto.

Visto lo anterior un ejemplo de creación de una base de datos sería el siguiente:

```
CREATE DATABASE IF NOT EXISTS BD
CHARACTER SET UTF8
COLLATE UTF8_SPANISH_CI;
```

El comando que nos permite ver las bases de datos creadas en el SGBD es:

```
SHOW DATABASES;
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Character set y collation se pueden especificar a nivel de base de datos¹ (ya visto), tablas² y columnas³ (se verán en los puntos siguientes). Se rigen por las siguientes condiciones:

- Si se especifica de forma implícita tanto character set como COLLATE se usan los valores especificados.
- Si se especifica CHARACTER SET pero no COLLATE se usa el character set especificado y el COLLATION por defecto del character set. (ver comando SHOW CHARACTER SET).
- Si se especifica COLLATE pero no el CHARACTER SET se usa como character set el character set que tiene como collation por defecto el collation indicado.
- Si no se especifica ninguno de los dos se usará:
 - En el caso de una base de datos: el CHARACTER SET y el COLLATE por defecto del SGDB.
 - En el caso de una tabla: el CHARACTER SET y el COLLATE por defecto de la base de datos en la que está contenida la tabla.
 - En el caso de una columna: el CHARACTER SET y el COLLATE por defecto de la tabla en la que está contenida la columna.

Para ver el character set y el collation de una base de datos se pueden usar los comandos:

```
USE baseDeDatos;
SELECT @@character_set_database, @@collation_database;
```

3. Borrado de bases de datos

Para eliminar de forma completa una base de datos se usa el comando:

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nombreBaseDeDatos;
```

Donde:

- DROP: acción a realizar, en este caso se elimina un objeto de la base de datos.
- DATABASE o SCHEMA: tipo de elemento a eliminar. En este caso una base de datos siendo las clausulas DATABASE y SCHEMA sinónimas.
- IF EXIST: esta clausula es opcional y es complementaria a IF NOT EXIST pero en este caso comprueba que la base de datos exista antes de realizar la acción requerida.
- nombreBaseDeDatos: nombre de la base de datos que se desea eliminar. En plataformas Linux el nombre es sensible al contexto.

Cuando se borra una base de datos se borran todos los elementos creados dentro de ella: tablas, vistas, procedimientos almacenados, ... además de todos los datos que posea.

4. Tipos de datos

Cuando se crea una tabla se debe especificar el tipo de dato para cada una de sus columnas, estos definen el dominio de valores que puede contener cada columna.

Por regla general, se debería seleccionar el tipo de dato de menor tamaño, ya que de esta forma se ahorra espacio y se logra una mayor velocidad de acceso y actualización. Sin embargo, si se selecciona un tipo de columna demasiado pequeño puede dar como resultado la perdida de datos o que se recorten al ser introducidos.

¹ <https://dev.mysql.com/doc/refman/5.7/en/charset-database.html>

² <https://dev.mysql.com/doc/refman/5.7/en/charset-table.html>

³ <https://dev.mysql.com/doc/refman/5.7/en/charset-column.html>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

4.1. Tipos de datos numéricos

Las columnas numéricas están diseñadas para almacenar todo tipo de datos numéricos, como precios, edades, cantidades, ... Existen dos tipos principales de tipos numéricos: tipos enteros (números enteros sin decimales) y tipos de coma flotante.

Valores enteros:

TIPO	TAM. BYTES ⁴	CON SIGNO		SIN SIGNO	
		VALOR MÍNIMO	VALOR MÁXIMO	VALOR MÍNIMO	VALOR MÁXIMO
TINYINT(M)	1	-128	127	0	255
SMALLINT(M)	2	-32768	32767	0	65535
MEDIUMINT(M)	3	-8388608	8388607	0	16777215
INT(M)	4	-2147483648	2147483647	0	4294967295
BIGINT(M)	8	-9223372036 854775808	922337203 6854775807	0	1844674407 3709551615

Números en coma flotante de valor aproximado (ver posibles problemas⁵):

TIPO	TAM. BYTES	CON SIGNO		SIN SIGNO	
		VALOR MÍNIMO	VALOR MÁXIMO	VALOR MÍNIMO	VALOR MÁXIMO
FLOAT(Precisión) FLOAT(M, D)	4 o 8	$-3.402823466 \times 10^{+38}$	$-1.17549435 \times 10^{-38}$	$1.17549435 \times 10^{-38}$	$3.402823466 \times 10^{+38}$
Números en coma flotante de precisión simple. La precisión puede ser menor o igual que 24. M indica nº de dígitos (incluidos los decimales) y D indica el número de decimales. 4 bytes si $0 \leq \text{precisión} \leq 24$, 8 bytes si $25 \leq \text{precisión} \leq 53$					
DOUBLE(M, D)	8	$-1.7976931348623157 \times 10^{+308}$	$-2.2250738585072014 \times 10^{-308}$	$2.2250738585072014 \times 10^{-308}$	$1.7976931348623157 \times 10^{+308}$
Números en coma flotante de precisión doble. M indica nº de dígitos (incluidos los decimales) y D indica el número de decimales.					
REAL(M, D)	Sinónimo de Double				

Números en coma flotante de precisión exacta:

TIPO	TAM.	CON SIGNO		SIN SIGNO	
	BYTES	VALOR MÍNIMO	VALOR MÁXIMO	VALOR MÍNIMO	VALOR MÁXIMO
DECIMAL(M,D)	Ver https://dev.mysql.com/doc/refman/5.0/en/precision-math-decimal-characteristics.html				
	M indica nº de dígitos (incluidos los decimales) y D indica el número de decimales. Si M se omite toma el valor 10. Si se omite D toma el valor 0. DECIMAL(M) = DECIMAL(M,0)				

Valores binarios:

Tipo	Descripción
BIT	Hasta la versión 5.0.2 de MySQL es sinónimo de TINYINT a partir de esa versión representa información binaria hasta 64 bits.


Hay que tener en cuenta que si se inserta un valor que supere el máximo rango este se ajustará al valor máximo permitido.

Todos los tipos numéricos permiten dos opciones: unsigned y zerofill.

- Unsigned: no permite el uso de números negativos ampliando el rango de los positivos.
- Zerofill: rellena el valor con ceros en lugar de los espacios habituales, además de asignar el tipo unsigned de manera predeterminada.

⁴ <http://dev.mysql.com/doc/refman/5.7/en/storage-requirements.html>

⁵ <http://dev.mysql.com/doc/refman/5.7/en/problems-with-float.html>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

4.2. Tipos de datos cadena⁶

Una cadena es una secuencia de caracteres, encerrada por comillas simples (') o dobles ("). Se utilizan para almacenar todo tipo de datos compuestos de caracteres como nombres, direcciones, ...

Las tablas siguientes describen algunos de los tipos de datos cadena disponibles.

TIPO	TAM. BYTES L = Longitud cadena	TIPO CAMPO	LONGITUD MÁXIMA	SENSIBLE AL CASO	COMENTARIOS
CHAR (M) ⁷	M bytes, 0<=M<= 255	Texto	255 carac.	No	Cadena de longitud fija.
BINARY(M)	M bytes, 0<=M<=255	Binario	255 bytes	Si	Cadena binaria de longitud fija
VARCHAR (M)	Lon+1 bytes si M<255 Lon+2 bytes si M>255	Texto	65536 carac.	No	Cadena de longitud variable
VARBINARY(M)	Lon+1 bytes si M<255 Lon+2 bytes si M>255	Binario	65536 bytes	Si	Cadena binaria de longitud variable
TINYTEXT	Lon+1 byte, L < 2 ⁸	Texto	255 carac.	No	
TINYBLOB	Lon+1 byte, L < 2 ⁸	Binario	255 bytes	Si	
TEXT	Lon+2 bytes, L < 2 ¹⁶	Texto	65.535 carac.	No	
BLOB	Lon+2 bytes, L < 2 ¹⁶	Binario	65.535 bytes	Si	
MEDIUMTEXT	Lon+3 bytes, L < 2 ²⁴	Texto	16.777.215 carac.	No	
MEDIUMBLOB	Lon+3 bytes, L < 2 ²⁴	Binario	16.777.215 bytes	Si	
LONGTEXT	Lon+4 bytes, L < 2 ³²	Texto	4.294.967.295 carac.	No	
LOBLOB	Lon+4 bytes, L < 2 ³²	Binario	4.294.967.295 bytes	Si	

Si queremos que un campo de texto sea sensible al caso, es decir, distinga entre mayúsculas y minúsculas usaremos la palabra clave binary (ver punto 4.2 del tema 5).

TIPO	DESCRIPCIÓN
ENUM ('valor1', 'valor2', ...) Ocupa 1 o 2 bytes, dependiendo del número de valores de la enumeración. Máximo de 65535 valores.	Contiene un enumerado. Un objeto de tipo cadena que puede tener un único valor, entre la lista de valores 'valor1', 'valor2', ..., NULL o el valor especial de error ". Un ENUM puede tener un máximo de 65535 valores diferentes. Internamente se representan como integers. Ejemplos: ENUM('Sí', 'No', 'NS/NC') ENUM('Hombre', 'Mujer')
SET ('valor1', 'valor2', ...) Ocupa 1, 2, 3, 4, o 8 bytes, dependiendo del número de miembros del conjunto. Máximo de 64 valores.	Contiene un conjunto. Un objeto de tipo cadena que puede tener cero o más valores, cada uno de los cuales debe estar entre la lista de valores 'valor1', 'valor2', ... Un conjunto puede tener un máximo de 64 valores diferentes. Internamente se representan como integers. Ejemplo: SET('música', 'cine', 'leer', 'viajar')

Los siguientes consejos nos pueden ayudar a la hora de decidir qué tipo de cadena seleccionar:

- No se debe almacenar nunca números en columnas de cadena. Resulta mucho más eficaz hacerlo en columnas de tipo numérico. Cada dígito incluido en una cadena ocupa un byte de espacio, en contraposición a un campo numérico, que los almacena en bits. Así mismo, la ordenación de números almacenados en columnas de cadena puede generar resultados incoherentes.

⁶ <https://mariadb.com/kb/en/mariadb/blob-and-text-data-types/>

⁷ <http://dev.mysql.com/doc/refman/5.7/en/char.html>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimiento de Aplicaciones Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

- Para ahorrar espacio, utilizaremos columnas dinámicas, como VARCHAR.
- Si no se especifica la palabra clave BINARY los valores se ordenan y comparan sin distinguir mayúsculas y minúsculas.
- Para limitar los contenidos de una columna a una opción, utilizaremos ENUM.
- Para permitir más de una entrada en una columna, utilizaremos SET
- Para imágenes y otros objetos binarios, se recomienda almacenarlos en el sistema de archivos en lugar de directamente en la base de datos.
- Los campos de tipo Blob no tienen character set.

4.3. Tipos de dato Fecha y Hora

Los tipos de columna de fecha y hora están diseñados para trabajar con las necesidades especiales que exigen los datos de tipo temporal y se puede utilizar para almacenar datos tales como la hora del día o fechas de nacimiento.

En tabla siguiente se ven algunos de los tipos de datos de fecha disponibles en MariaDB/MySQL.

TIPO	FORMATO	BYTES	VALORES DISPONIBLES
YEAR	YYYY 2 o 4 dígitos	1	Con cuatro dígitos un valor entre 1901 y 2155 Con dos dígitos los valores 70-99 son convertidos a 1970-1999 y los valores a 00-69 a 2000-2069.
DATE	YYYY-MM-DD	3	Un valor entre 1000-01-01 a 9999-12-31
TIME	HH:MM:SS	3*	Un valor entre -838:59:59.999999 a 838:59:59.999999
DATETIME	YYYY-MM-DD HH:MM:SS DATE + TIME	8*	Un valor entre 01-01 00:00:00.000000 y 9999-12-31 23:59:59.000000
TIMESTAMP		4*	Un valor entre 1970-01-01 00:00:01 y 2038-01-19 05:14:07

* Estos tipos de datos admiten el uso de microsegundos que permiten aumentar su precisión a costa de ocupar más espacio. Se puede consultar el espacio adicional ocupado en la tabla adyacente.

PRECISIÓN EN SEGUNDOS	BYTES OCUPADOS
0	0 bytes
1, 2	1 byte
3, 4	2 bytes
5, 6	3 bytes

Dada la siguiente tabla:

```
CREATE TABLE `fechas` (
  `datetime` DATETIME DEFAULT CURRENT_TIMESTAMP,
  `timestamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

Donde su contenido es el siguiente:

datetime	timestamp
2016-01-10 20:54:13	2016-01-10 20:54:13


Ahora cambiamos la zona horaria del servidor MariaDB/Mysql aumentándola en 3 horas y mostramos el contenido de la tabla fechas.

```
show variables like "%time_zone%";
SET time_zone = '+3:00';
show variables like "%time_zone%";
```

```
select * from fechas;
```

datetime	timestamp
2016-01-10 20:54:13	2016-01-10 22:54:13

Podemos ver que timestamp, antes de almacenarse, se convierte a una hora UTC y vuelve a convertirse a la zona horaria actual cuando se recupera mientras que el valor de datetime no varía. Vemos que la fecha de timestamp se ha actualizado a la nueva zona horaria +3 (hay que tener en cuenta que la zona horaria anterior era +1) mientras que el valor de datetime no varía.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

5. Creación tablas

Antes de crear una tabla es conveniente tener en cuenta ciertos aspectos:

- La denominación de la tabla ha de ser única y no puede ser una palabra reservada.
- El nombre de la tabla debe ser un nombre que identifique su contenido. Por ejemplo, llamamos a una tabla ALUMNOS porque contendrá datos sobre alumnos.
- El nombre de cada columna de la tabla ha de ser un nombre autodescriptivo, que identifique su contenido. Por ejemplo: DNI, NOMBRE o APELLIDOS.
- El tipo de dato y el tamaño que tendrá cada columna.
- Las columnas obligatorias, los valores por defecto, etc.

Para crear una tabla usamos la orden CREATE TABLE, cuyo formato más simple es:

```
CREATE TABLE nombreTabla ( columna1 tipoDato1, columna2 tipoDdato2 );
```

Donde:

- columna1, columna2: son los nombres de las columnas.
- tipoDato1, tipoDdato2: indica el tipo de dato de cada columna.

EJEMPLO

Creemos una tabla llamada profesores:

```
CREATE TABLE profesores (
  COD_CENTRO int(4),
  DNI varchar(10),
  APELLIDOS varchar(50),
  ESPECIALIDAD varchar(50)
) CHARACTER SET UTF8;
```

Hemos de hacer algunas observaciones:

- Las definiciones individuales de columnas se separan mediante comas.
- No se pone coma después de la última definición de columna.
- Las mayúsculas y minúsculas son indiferentes a la hora de crear una tabla.

Para mostrar las tablas de la base de datos con la que estamos trabajando usaremos el comando:

```
SHOW TABLES;
```

Para consultar los campos de una tabla y el tipo de datos que contiene se utilizan los comandos:

```
DESCRIBE nombreTabla;
SHOW [FULL] COLUMNS FROM nombreTabla;
```

Para consultar la sentencia de creación de la tabla sugerida por MariaDB/MySQL se usará:

```
SHOW CREATE TABLE nombreTabla;
```


COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

6. Definición de columnas

La sintaxis para definir una columna en el proceso de creación de una tabla es:

```
nombreColumna tipoDato [NOT NULL | NULL] [DEFAULT valorPorDefecto]
[AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string'] [definiciónReferencia]
```

6.1. Obligatoriedad. La restricción de valores nulos

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

Debemos recordar que aquellas columnas que son o forman parte de una clave primaria no pueden contener valores nulos. Si definimos una columna como clave primaria, MariaDB/MySQL automáticamente impide que pueda contener valores nulos.

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

```
CREATE TABLE alumno (
  nombre CHAR(20) NOT NULL,
  nota INT NULL
);
```

6.2. Valores por defecto

En el momento de crear una tabla podemos asignar valores por defecto a las columnas. Si especificamos la cláusula DEFAULT a una columna el valor especificado se asignará de forma automática a una columna cuando, al insertar valores en las filas, no se especifique un valor para este campo.

Si una columna puede tener un valor nulo, y no se especifica un valor por defecto, se usará NULL como valor por defecto. En el ejemplo anterior, el valor por defecto para nota es NULL.

Si queremos que el valor por defecto para nota sea 5, podemos crear la tabla como:

```
CREATE TABLE alumno2 (
  nombre CHAR(20) NOT NULL,
  nota INT NULL DEFAULT 5
);
```


6.3. Claves primarias. La restricción PRIMARY KEY

Una clave primaria dentro de una tabla es una columna que identifica de forma única a cada fila. Debe ser única, no nula (si no se especifica de forma explícita, MariaDB/MySQL lo hará de forma automática) y obligatoria. Como máximo podemos definir una clave primaria por tabla (puede contener varias columnas). Cuando se crea una clave primaria, automáticamente se crea un índice que facilita el acceso a la tabla. Para definir una clave primaria en una tabla usamos la restricción PRIMARY KEY.

Por ejemplo, si queremos establecer la columna nombre como clave primaria de la tabla de ciudades, crearemos la tabla así:

```
CREATE TABLE ciudad (
  nombre CHAR(20) NOT NULL PRIMARY KEY,
  poblacion INT NULL DEFAULT 5000
);
```

Esta sintaxis no permite crear claves primarias sobre varias columnas.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

6.4. La restricción UNIQUE

Unique evita que en una misma columna, salvo el valor nulo, tenga valores repetidos. Puede contener una o varias columnas. Es similar a la restricción PRIMARY KEY, salvo que son posibles varias columnas UNIQUE definidas en una misma tabla y que pueden contener el valor NULL.

```
CREATE TABLE ciudad2 (
  clave INT AUTO_INCREMENT PRIMARY KEY,
  nombre CHAR(20) NULL UNIQUE KEY,
  poblacion INT NULL DEFAULT 5000
);
```

6.5. Comentarios

Adicionalmente, al crear la tabla, podemos añadir un comentario a cada columna. Este comentario sirve como información adicional sobre alguna característica especial de la columna, y entra en el apartado de documentación de la base de datos:

```
CREATE TABLE ciudad3 (
  clave INT AUTO_INCREMENT PRIMARY KEY COMMENT 'Clave principal',
  nombre CHAR(50) NOT NULL,
  poblacion INT NULL DEFAULT 5000
);
```

6.6. Campos autoincrementados

En MariaDB/MySQL tenemos la posibilidad de crear una columna autoincrementada, aunque esta columna sólo puede ser de tipo entero.


Si al insertar una fila se omite el valor de la columna autoincrementada o se inserta, en esa columna, un valor nulo o un valor *default*, su valor se calcula automáticamente tomando como valor el valor más alto que haya tenido esa columna y sumándole una unidad.

Generalmente, estas columnas se usan como claves primarias '*artificiales*':

```
CREATE TABLE ciudad4 (
  clave INT AUTO_INCREMENT PRIMARY KEY,
  nombre CHAR(20) NOT NULL,
  poblacion INT NULL DEFAULT 5000
);
```

Se han de tener en cuenta las siguientes consideraciones:

- Solo se puede usar cuando se usen uno de los atributos: NOT NULL, PRIMARY KEY o UNIQUE
- Una tabla solo puede tener una columna AUTO_INCREMENT (en principio).
- La generación automática de un nuevo valor solo funciona cuando se crean nuevos registros usando INSERT y no se proporciona un valor específico.
- Para saber el último valor generado automáticamente después de una instrucción INSERT, se puede ejecutar el comando SELECT LAST_INSERT_ID(). Un detalle importante es que en un INSERT múltiple, esta función retorna el entero generado para el primer registro.
- Si el contador de AUTO_INCREMENT alcanza el valor máximo (depende del tipo de entero que estemos usando) no se incrementará más, es decir, no se permitirán más operaciones de INSERT en la tabla.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

7. Borrado de tablas

A veces es necesario eliminar una tabla, ya sea porque es más sencillo crearla de nuevo que modificarla, o porque ya no es necesaria.

Al eliminar una tabla todos sus datos serán borrados de la base de datos.

Para eliminar una tabla se usa la sentencia DROP TABLE. Su sintaxis es:

```
DROP TABLE [if exists] nombretabla;
```

8. Cambios de nombre de tablas: Rename

RENAME se utiliza para cambiar el nombre de una tabla. El nuevo nombre no puede ser una palabra reservada ni el nombre de una tabla o una vista que tenga creado el usuario.

Su formato es:

```
RENAME TABLE nombreAnterior TO nombreNuevo;
```

Así mismo este comando nos permite mover tablas entre bases de datos:

```
RENAME TABLE baseDatosOriginal.nombreTabla TO baseDatosDestino.nuevoNombreTabla
```

9. Copiado de tablas

Es posible crear una tabla nueva con el contenido de otra ya existente (copiando los registros afectados por SELECT, que podría no devolver ninguno y crear la tabla vacía):

```
CREATE TABLE nuevaTabla SELECT * FROM tablaOrigen [ WHERE ... ];
```

Pero tiene las siguientes limitaciones:

- No traspasa las claves primarias.
- No traspasa las definiciones AUTO_INCREMENT.
- No traspasa las definiciones de DEFAULT CURRENT_TIMESTAMP.
- Utiliza el gestor de almacenamiento por defecto y no el de la tabla.

La siguiente sentencia solo copia la estructura de la tabla origen sin copiar los datos:

```
CREATE TABLE nuevaTabla Like tablaOrigen;
```

Tanto si utilizamos la copia mediante SELECT como si usamos LIKE, las claves foráneas no se copian y se debe hacer un ALTER TABLE manual para incluirlas después.

10. Definición de creación


A continuación de las definiciones de las columnas podemos añadir definiciones adicionales. La sintaxis más general es:

```
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 ]
```

Las restricciones de tabla pueden definirse con un identificador útil para hacer referencias posteriores a la restricción. Por ejemplo:

```
constraint clave primary key (nombre, nacimiento)
```

Veremos ahora cada una de estas opciones.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

10.1. Índices

Los índices sirven para optimizar las consultas y las búsquedas de datos. Mediante su uso es mucho más rápido localizar filas con determinados valores de columnas. La alternativa es hacer búsquedas secuenciales, que en tablas grandes requieren mucho tiempo.

Tenemos tres tipos de índices:

10.1.1. Claves primarias

La sintaxis para definir claves primarias es:

```
definición_columnas
| PRIMARY KEY (nombreCol1, [ nombreCol2, ... ])
```

Entre los paréntesis podemos especificar varios nombres de columnas, para construir claves primarias compuestas por varias columnas:

```
CREATE TABLE miTabla (
  id1 CHAR(2) NOT NULL,
  id2 CHAR(2) NOT NULL,
  texto CHAR(30),
  PRIMARY KEY (id1, id2)
);
```

Las claves primarias introducen una restricción, ya que no puede haber dos filas en la que la clave primaria sea igual.

10.1.2. Índices

El segundo tipo de índice permite definir índices sobre una columna, sobre varias, o sobre partes de columnas. Para definir estos índices se usan indistintamente las opciones KEY o INDEX.

```
CREATE TABLE miTabla2 (
  id INT,
  nombre CHAR(19),
  INDEX (nombre)
);
```

```
CREATE TABLE miTabla3 (
  id INT,
  nombre CHAR(19),
  key (nombre(4))
);
```

Como vemos en el ejemplo de la derecha también se puede crear un índice sobre parte de una columna. Vemos que sólo usa los cuatro primeros caracteres de la columna 'nombre' para crear el índice.

Para mostrar los índices de una tabla se utiliza el comando:

```
SHOW INDEX FROM nombreTabla;
```

10.1.3. Claves únicas


El tercero tipo de índice permite definir índices con claves únicas sobre una columna, sobre varias o sobre partes de columnas. Para definir índices con claves únicas se usa la opción UNIQUE.

A diferencia de un índice normal, un índice único no permite la inserción de filas con valores repetidos. La excepción es el valor NULL, que se puede repetir.

Una clave primaria equivale a un índice de clave única, en la que el valor de la clave no puede tomar valores NULL.

Las claves únicas introducen la restricción de que dos filas de una tabla no pueden tomar valores iguales, a excepción del valor nulo, para una columna que se defina como unique.

```
CREATE TABLE mitabla4 (
  id INT,
  nombre CHAR(19),
  UNIQUE (nombre)
);
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

10.2. Claves foráneas

En MariaDB/MySQL sólo existe soporte para claves foráneas en tablas de tipo InnoDB. Sin embargo, esto no impide usarlas en otros tipos de tablas. La diferencia consiste en que en esos tipos de tablas no se verifica la integridad referencial (si una clave foránea existe realmente en la tabla referenciada).

Hay dos maneras de definir claves foráneas en bases de datos MariaDB/MySQL.

- La primera, sólo sirve para documentar. Esta forma consiste en definir una referencia al mismo tiempo que se define una columna:

```
CREATE TABLE personas (
  id INT PRIMARY KEY,
  nombre VARCHAR(40),
  fecha DATE
);
```

```
CREATE TABLE telefonos (
  id INT NOT NULL,
  numero CHAR(12),
  persona INT NOT NULL REFERENCES personas (id)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT
);
```

Hemos usado una definición de referencia para la columna 'persona' de la tabla 'telefonos', indicando que es una clave foránea correspondiente a la columna 'id' de la tabla 'personas'. Sin embargo, aunque la sintaxis se comprueba, esta definición no implica ningún comportamiento por parte de MariaDB/MySQL.

- La otra forma es mucho más útil, aunque sólo se aplica a tablas InnoDB.

En esta forma no se añade la referencia en la definición de la columna, sino después de la definición de todas las columnas.


Tenemos la siguiente sintaxis:

```
CREATE TABLE nombre
definición_de_columnas
[ CONSTRAINT [ simbolo ] ] FOREIGN KEY [ nombre_index ] (nombre_col_index,...)
[ REFERENCES nombre_tabla [ (nombre_col,...) ]
[ ON DELETE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT ]
[ ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT ]
]
```

El ejemplo anterior, usando tablas InnoDB y la definición de claves foráneas será:

```
CREATE TABLE personas (
  id INT,
  nombre VARCHAR(40),
  fecha DATE,
  PRIMARY KEY (id)
) ENGINE=InnoDB;
```

```
CREATE TABLE telefonos (
  id INT NOT NULL,
  persona int not null,
  numero CHAR(12),
  PRIMARY KEY (id),
  INDEX (persona),
  FOREIGN KEY (persona) REFERENCES personas (id)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT
) ENGINE=InnoDB;
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Hay que tener en cuenta los siguientes aspectos:

- Se ha de definir un índice sobre la clave foránea de la siguiente forma: INDEX (id).
Es imprescindible que la columna que contiene una definición de clave foránea esté indexada. Pero esto no debe preocuparnos demasiado, ya que si no lo hacemos de forma explícita, MariaDB/MySQL lo hará por nosotros de forma implícita.
- Si queremos utilizar en nuestras tablas la integridad referencial debemos utilizar el motor de almacenamiento InnoDB mediante ENGINE=InnoDB.
- Definir la clave foránea: FOREIGN KEY (persona) REFERENCES personas (id)
Esta forma define una clave foránea en la columna 'persona' de la tabla teléfonos, que hace referencia a la columna 'id' de la tabla 'personas'.
- Indicar las opciones que tenemos para mantener la integridad referencial.

Las claves foráneas introducen otra restricción, ya que estás solo pueden tomar un valor que exista en la tabla referenciada.

Opciones para el mantenimiento de la integridad referencial

La integridad referencial permite definir como se comportaran antes eliminaciones y actualizaciones:

- ON DELETE <opción>, indica que acciones se deben realizar en la tabla actual si se borra una fila en la tabla referenciada.
- ON UPDATE <opción>, es análogo pero para modificaciones.

Existen cinco opciones diferentes:

- RESTRICT: Es el comportamiento por defecto. Impide eliminar o modificar filas en la tabla referenciada si la clave foránea en la tabla que referencia usa ese valor. En el ejemplo anterior no se podrán borrar una fila en persona si su valor se está usando en teléfonos.
- CASCADE: Borrar o modificar una clave en una fila en la tabla referenciada propaga los cambios o borrados a la tabla que referencia.
- SET NULL: Borrar o modificar una clave en una fila en la tabla referenciada con un valor determinado, implica asignar el valor NULL a las claves foráneas con el mismo valor en la tabla que referencia.
- 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 en la tabla que referencia.

En el ejemplo anterior (el de las tablas personas y teléfonos), si hemos introducido un número de teléfono para una persona con id=10 en la tabla teléfonos, no podremos eliminar a esa persona de la tabla personas. Además no podremos introducir el número de teléfono de la persona con id =20, en la tabla teléfonos, si no ha sido dado de alta previamente en la tabla personas.

Por ejemplo, veamos esta definición de la tabla 'telefonos2':

```
CREATE TABLE telefonos2 (
  numero CHAR(12),
  persona INT NOT NULL,
  INDEX (persona),
  FOREIGN KEY (persona) REFERENCES personas (id)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
) ENGINE=InnoDB;
```

COLEXIO VIVAS _{S.L.}	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Si se intenta borrar una fila de 'personas' con un determinado valor de 'id', se producirá un error si existen filas en la tabla 'telefonos2' con mismo valor en la columna 'id'. La fila de 'personas' no se eliminará, a no ser que previamente eliminemos las filas con el mismo valor de clave foránea en 'teléfonos2'.

Si se modifica el valor de la columna 'id' en la tabla 'personas', se modificarán los valores de la columna 'personas' de la tabla telefonos2 para mantener la integridad.

Veamos un ejemplo práctico:

Personas			Teléfonos2	
IdPers	Nombre	Fecha	Número	Persona(fk)
1	Lucia	1995/05/08	11111	1
2	José	1985/09/01	22222	1
3	Eva	1992/03/02	33333	3
			44444	3

Si intentamos borrar la fila correspondiente a 'Lucia' se producirá un error, ya que existen dos filas en teléfonos2 con el valor 1 en la columna Persona por estar definido como restrict en los borrados.

Sí será posible borrar la fila correspondiente a 'José', ya que no existe ninguna fila en la tabla teléfonos con el valor 2 en la columna Persona.

Si modificamos el valor de "IdPers" en la fila correspondiente a 'Eva', por el valor 5, se actualizara a 5 las filas 3ª y 4ª en la columna 'Persona' de la tabla teléfonos por estar definido como cascade en las actualizaciones:

Personas			Teléfonos2	
Id	Nombre	Fecha	Número	Persona(fk)
1	Lucia	1995/05/08	11111	1
2	José	1985/09/01	22222	1
5	Eva	1992/03/02	33333	5
			44444	5

10.3. Opciones de tabla

La parte final de la sentencia CREATE TABLE permite especificar varias opciones para la tabla.

Sólo comentaremos la opción del motor de almacenamiento

La sintaxis de esta opción es:

```
[ {ENGINE} = { InnoDB | MyISAM | CSV | MEMORY } ]
```

Tenemos disponibles varios motores de almacenamiento. Algunos de ellos serán de uso obligatorio si queremos tener ciertas opciones disponibles. Por ejemplo, ya hemos comentado que el soporte para claves foráneas sólo está disponible para el motor InnoDB.

Algunos de los motores más habituales son:

- MyISAM: Son tablas no transaccionales. Proporciona almacenamiento y recuperación rápida de datos.
- MEMORY: Tablas almacenadas en memoria.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

- InnoDB: Soporta transacciones tipo ACID, bloqueo de registros e integridad referencial. InnoDB ofrece una fiabilidad y consistencia muy superior a MyISAM, si bien el mejor rendimiento de uno u otro formato dependerá de la aplicación específica.
- CSV: Guarda datos en ficheros de texto usando formato de valores separados por comas.

Generalmente usaremos tablas MyISAM o tablas InnoDB y a veces, cuando se requiera una gran optimización, se crearán tablas temporales en memoria

Podemos consultar los motores de almacenamiento disponibles mediante el comando:

```
SHOW [STORAGE] ENGINES;
```


Engine	Support	Comment	Transactions	XA	Savepoints
CSV	YES	CSV storage engine	NO	NO	NO
InnoDB	DEFAULT	Percona-XtraDB, Supports transactions, row-level locking...	YES	YES	YES
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
MRG_MyISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
Aria	YES	Crash-safe tables with MyISAM heritage	NO	NO	NO

Ventajas e inconvenientes de los motores de almacenamiento MyISAM y InnoDB:

- MyISAM
 - Mayor velocidad en general a la hora de recuperar datos.
 - Recomendable para aplicaciones en las que dominan las sentencias SELECT ante los INSERT / UPDATE.
 - Ausencia de características de atomicidad ya que no tiene que hacer comprobaciones de la integridad referencial, ni bloquear las tablas para realizar las operaciones, esto nos lleva como los anteriores puntos a una mayor velocidad pero menor seguridad de datos.
- InnoDB
 - InnoDB provee bloqueo a nivel de registro, en contra del bloqueo a nivel tabla de MyISAM. Esto es, que mientras una consulta está actualizando o insertando una fila, otra consulta puede actualizar una fila diferente al mismo tiempo. Estas características incrementan el rendimiento en concurrencia de múltiples usuarios.
 - Es probable que si nuestra aplicación hace un uso elevado de INSERT y UPDATE notemos un aumento de rendimiento con respecto a MyISAM.
 - Soporte de transacciones
 - Permite tener las características ACID (Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad), garantizando la integridad de nuestras tablas.
 - Recuperación de datos perdidos. Después de una caída del sistema, las tablas InnoDB se intentan llevar a un estado consistente de forma automática y muy rápida.
 - Requerimiento de espacio. Las tablas InnoDB ocupan mucho más espacio que las tablas MyISAM.

Para modificar el motor de almacenamiento de una tabla podemos usar la sentencia:

```
ALTER TABLE nombreTabla ENGINE = MotorDeAlmacenamiento;
```


	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

11. Modificación de tablas

Se puede modificar una tabla mediante la orden ALTER TABLE.

La modificación de tablas nos permite: añadir, modificar o eliminar columnas de una tabla, modificar definiciones, cambiar el nombre de las tablas, ...

11.1. Cambiar el nombre de una tabla. RENAME.

Se utiliza RENAME para cambiar el nombre de una tabla.

El formato es el siguiente:

```
ALTER TABLE nombreTabla RENAME [TO] nuevoNombre;
```

11.2. Añadir columnas. ADD.

Se utiliza ADD para añadir columnas a una tabla.

Si necesitamos crear una columna nueva en la tabla ventas para almacenar las cantidades vendidas. UPDATE no serviría, ya que esta instrucción solo modifica los datos, no la estructura.

Para realizar este cambio, es necesario utilizar la instrucción ALTER junto con ADD. Si se añade más de una columna las definiciones de estas se tienen que encerrar entre paréntesis y separarse por comas:

```
ALTER TABLE nombreTabla ADD columnaNueva tipo;
ALTER TABLE nombreTabla ADD (columnaNueva1 tipo1, columnaNueva2 tipo2);
ALTER TABLE ventas ADD cantidad INT;
```

Si se desea crear una columna al inicio de una tabla se utiliza la palabra FIRST:

```
ALTER TABLE nombreTabla ADD columnaNueva tipo FIRST;
```

Para colocar columnas después de una columna dada se utiliza la palabra AFTER nombreColumna:

```
ALTER TABLE nombreTabla ADD columnaNueva tipo AFTER columna;
```

11.3. Modificación de la definición de una columna. CHANGE.

Para modificar la definición de una columna, podemos usar la instrucción CHANGE. Esta instrucción permite cambiar el nombre de la columna:

```
ALTER TABLE nombreTabla CHANGE nombreColumnaAntigua nombreColumnaNueva definicionColumna;
ALTER TABLE ventas CHANGE cantidad cantidad2 INT DEFAULT 6;
```

Tras la clausula CHANGE se incluye el nombre de la antigua columna seguido del nombre de la nueva columna y de su definición.

También puede utilizar la clausula MODIFY, sin que resulte necesario cambiar el nombre de la columna, de la siguiente forma:

```
ALTER TABLE localidad MODIFY ciudad VARCHAR(10) DEFAULT 'Vigo';
```

Mediante MODIFY podemos mover de posición una columna dentro de la tabla.


```
ALTER TABLE nombreTabla MODIFY nombreColumna definicionColumna {FIRST | AFTER} [otraColumna];
ALTER TABLE ventas MODIFY columna varchar(10) NOT NULL FIRST;
```

11.4. Eliminar una columna. DROP.

Se utiliza DROP para borrar una columna de una tabla.

El formato es el siguiente:

```
ALTER TABLE nombreTabla DROP nombreColumna;
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

11.5. Modificación de las claves primarias

Una vez creadas las claves primarias estas se pueden modificar con la siguiente sentencia:

```
ALTER TABLE nombreTabla ADD PRIMARY KEY (columna1,...) ;
```

Y se puede eliminar con:

```
ALTER TABLE nombreTabla DROP PRIMARY KEY;
```

11.6. Modificación de las claves únicas

Una vez creadas las claves únicas estas se pueden modificar con la siguiente sentencia:

```
ALTER TABLE nombreTabla ADD UNIQUE KEY [nombreIndice] (columna1,...) ;
```

Y se pueden eliminar con:

```
ALTER TABLE nombreTabla DROP INDEX nombreIndice;
```

11.7. Modificación de índices

Una vez creados los índices estos se pueden modificar con las siguientes sentencias:

```
CREATE INDEX nombreIndice ON nombreTabla (columna);
```

```
ALTER TABLE nombreTabla ADD INDEX [nombreIndice] (columna1,...) ;
```

Y se puede eliminar con:

```
ALTER TABLE nombreTabla DROP INDEX nombreIndice;
```

11.8. Modificación de claves foráneas

Una vez creadas las claves foráneas estas se pueden modificar con la siguiente sentencia:

```
ALTER TABLE nombreTabla ADD FOREIGN KEY [nombreFK] (columna) REFERENCES tabla(columna) [on ...];
```

Y se puede eliminar con:

```
ALTER TABLE nombreTabla DROP FOREIGN KEY nombreFK;
```

11.9. Modificación del conjunto de caracteres y de las colecciones

Para cambiar la colección de caracteres y la colección de una tabla se utiliza:

```
ALTER TABLE nombreTabla [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
```

Y para convertir el contenido de una tabla se utiliza:

```
ALTER TABLE nombreTabla CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
```

11.10. Cambiar el motor de almacenamiento

Para cambiar la colección de caracteres y la colección de una tabla se utiliza:

```
ALTER TABLE nombreTabla ENGINE = nombreMotor;
```

Donde entre otros el nombre del motor puede ser: MyISAM, InnoDB,

11.11. Concatenación de modificaciones

Se pueden concatenar varias modificaciones de una tabla en una única sentencia separándolas por comas.

```
ALTER TABLE nombreTabla DROP PRIMARY KEY, ADD id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY(id);
```