UNIDAD DIDÁCTICA 6 INTRODUCCIÓN SQL. DDL (LENGUAJE DEFINICIÓN DE DATOS)

¿QUÉ ES SQL?

El **SQL** (Structured Query Language), **lenguaje de consulta estructurado**, es un lenguaje surgido de un proyecto de investigación de IBM para el acceso a bases de datos relacionales. Actualmente se ha convertido en un **estándar** de lenguaje de bases de datos, y la mayoría de los sistemas de bases de datos lo soportan, desde sistemas para ordenadores personales, hasta grandes ordenadores.

Por supuesto, a partir del estándar cada sistema ha desarrollado su propio SQL que puede variar de un sistema a otro, pero con cambios que no suponen ninguna complicación para alguien que conozca un SQL concreto.

SQL nos **permite** realizar **consultas a la base de datos**. Pero el nombre se queda corto ya que SQL además realiza funciones de **definición, control y gestión de la base de datos.** Las sentencias SQL se clasifican según su finalidad dando origen a tres 'lenguajes' o mejor dicho sublenguajes:

- el **DDL** (Data Description Language), **lenguaje de definición** de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos y de las relaciones entre estas. (Es el que más varia de un sistema a otro).
    - **CREATE**, se usa para crear una base de datos, tabla, vistas, etc.
    - **ALTER**, se utiliza para modificar la estructura, por ejemplo añadir o borrar columnas de una tabla.
    - **DROP**, con esta sentencia, podemos eliminar los objetos de la estructura, por ejemplo un índice o una secuencia.
- el **DCL** (Data Control Language), **lenguaje de control** de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.
    - **GRANT**, permite otorgar permisos.
    - **REVOKE**, elimina los permisos que previamente se han concedido.
- el **DML** (Data Manipulation Language), **lenguaje de manipulación** de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados.
    - **SELECT**, esta sentencia se utiliza para realizar consultas sobre los datos.
    - **INSERT**, con esta instrucción podemos insertar los valores en una base de datos.
    - **UPDATE**, sirve para modificar los valores de uno o varios registros.
    - **DELETE**, se utiliza para eliminar las finas de una tabla

CARACTERÍSTICAS DEL LENGUAJE

Una sentencia SQL es como una frase (escrita en inglés) con la que decimos lo que queremos obtener y de donde obtenerlo.

Todas las sentencias empiezan con un **verbo** (palabra reservada que indica la acción a realizar), seguido del resto de **cláusulas**, algunas **obligatorias** y otras **opcionales** que completan la frase. Todas las sentencias siguen una **sintaxis** para que se puedan ejecutar correctamente, para describir esa sintaxis utilizaremos un **diagrama sintáctico** como el que se muestra a continuación.

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification [, create_specification] ...]
create_specification:
[DEFAULT] CHARACTER SET charset_name
| [DEFAULT] COLLATE collation_name
```

Las palabras que aparecen en mayúsculas son palabras reservadas se tienen que poner tal cual y no se pueden utilizar para otro fin, por ejemplo: **SELECT**, **ALL, DISTINCT, FROM, WHERE**.

Las palabras que aparecen entre corchetes son opcionales, por ejemplo: **IF NOT EXISTS.**

Las palabras en minúsculas son variables que el usuario deberá sustituir por un dato concreto. Por ejemplo: **charset_name, collation_name**

EL DDL, LENGUAJE DE DEFINICIÓN DE DATOS

El **DDL** (**D**ata **D**efinition **L**anguage) **lenguaje de definición de datos** es la parte del SQL que **más varía de un sistema a otro** ya que esa área tiene que ver con cómo se organizan internamente los datos y eso, cada sistema lo hace de una manera u otra. Las sentencias del **lenguaje de definición de datos** afectan a la estructura de los datos.

CREATE DATABASE

La sentencia **CREATE DATABASE** sirve para **crear la base de datos con el nombre indicado.**

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification [, create_specification] ...]
create_specification:
[DEFAULT] CHARACTER SET charset_name
| [DEFAULT] COLLATE collation_name
```

USE Seleccionar una base de datos

La sentencia **USE** selecciona una base de datos, para establecerla como base de datos por defecto o actual, para una conexión determinada del servidor.

```
USE db_name
```

La sentencia **USE db_name** indica a MySQL que use la base de datos *db_name* como la base de datos por defecto (actual) en sentencias subsiguientes. La base de datos sigue siendo la base de datos por defecto hasta el final de la sesión o hasta que se use otra sentencia **USE**:

DROP DATADASE Eliminar una Base de Datos

De modo parecido, se pueden eliminar bases de datos completas, usando la sentencia **DROP DATABASE.**

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE elimina todas las tablas de la base de datos y borra la base de datos. Hay que ser extremadamente cuidadoso con esta sentencia.

Se pueden usar las palabras clave IF EXISTS para evitar el error que se produce si la base de datos no existe.

ALTER TABLE DATABASE Modificar una Base de Datos

La sentencia ALTER DATABASE permite cambios en los atributos globales de una base de datos. Los únicos atributos disponibles son el conjunto de caracteres y la colección de caracteres.

```
ALTER DATABASE [IF EXISTS] db_name [CHARACTER SET
conjunto_car] [COLLATE colección];
```

CREATE TABLE

La sentencia **CREATE TABLE** sirve para **crear la estructura de una tabla** no para rellenarla con datos, nos permite **definir las columnas** que tiene **y ciertas restricciones** que deben cumplir esas columnas.

La **sintaxis** es la siguiente:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nombre_tabla
(create_definition,...)
[opciones_tabla ...]
```

La **sintaxis** para definir columnas (**create_definition**) es:

```
column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type]
(index_col_name,...)
| {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
[index_name] [index_type] (index_col_name,...)
| {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name]
(index_col_name,...)
| [CONSTRAINT [symbol]] FOREIGN KEY
[index_name] (index_col_name,...) [reference_definition]
| CHECK (expr)
```

La **sintaxis** para ***column_definition***:

```
col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string'] [reference_definition]
```

Tipos de Datos

Tipo	Formato
Cadena	
CHAR	CHAR
CHAR()	[NATIONAL] CHAR(M) [BINARY | ASCII | UNICODE]
VARCHAR()	[NATIONAL] VARCHAR(M) [BINARY]
Enteros	
TINYINT	TINYINT[(M)] [UNSIGNED] [ZEROFILL]
BIT / BOOL / BOOLEAN	Todos son sinónimos de **TINYINT**
SMALLINT	SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
MEDIUMINT	MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
INT	INT[(M)] [UNSIGNED] [ZEROFILL]
INTEGER	INTEGER[(M)] [UNSIGNED] [ZEROFILL]
BIGINT	BIGINT[(M)] [UNSIGNED] [ZEROFILL]
Coma flotante	
FLOAT	FLOAT(precision) [UNSIGNED] [ZEROFILL]
FLOAT()	FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]
DOUBLE	DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]
DOUBLE PRECISION / REAL Ambos son sinónimos de **DOUBLE**	DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL] REAL[(M,D)] [UNSIGNED] [ZEROFILL]
DECIMAL	DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]
DEC / NUMERIC / FIXED sinónimos de **DECIMAL**	DEC[(M[,D])] [UNSIGNED] [ZEROFILL] NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL] FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]

Tipo	Formato
datos para tiempos	
DATE	`DATE`
DATETIME	`DATETIME`
TIMESTAMP	`TIMESTAMP[(M)]`
TIME	`TIME`
YEAR	`YEAR[(2|4)]`
Datos sin tipo o grandes bloques de datos	
TINYBLOB / TINYTEXT	`TINYBLOB` `TINYTEXT`
BLOB / TEXT	`BLOB` `TEXT`
MEDIUMBLOB / MEDIUMTEXT	`MEDIUMBLOB` `MEDIUMTEXT`
LONGBLOB / LONGTEXT	`LONGBLOB` `LONGTEXT`
Tipos enumerados y conjuntos	
ENUM	`ENUM('valor1','valor2',...)`
SET	`SET('valor1','valor2',...)`

Una **restricción** consiste en la definición de una **característica adicional que tiene una columna** o una combinación de columnas, suelen ser características como valores no nulos (campo requerido), definición de índice sin duplicados, definición de clave principal y definición de clave foránea (clave ajena o externa, campo que sirve para relacionar dos tablas entre sí).

restricción1: una **restricción de tipo 1** es una restricción que aparece **dentro de la definición de la columna** después del tipo de dato y **afecta** a **una columna**, la que se está definiendo.

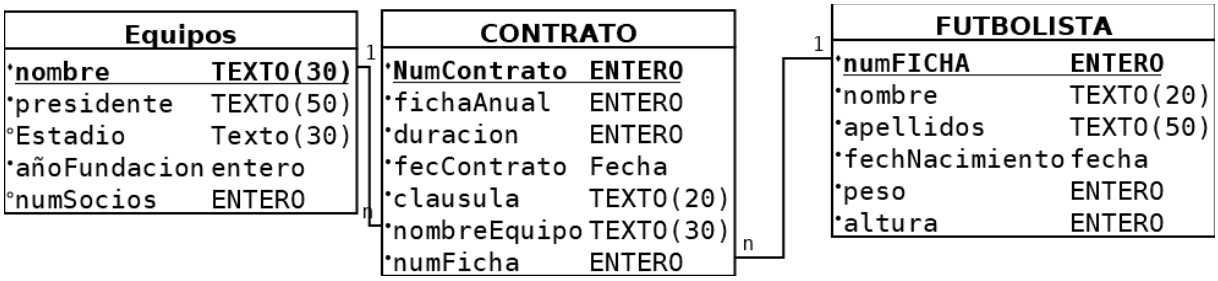
restricción2: una **restricción de tipo 2** es una restricción que se define **después de definir todas las columnas** de la tabla y **afecta** a **una** columna **o** a una **combinación de columnas.**

Para escribir una sentencia **CREATE TABLE** se empieza por indicar el **nombre de la tabla** que queremos crear y a continuación **entre paréntesis** indicamos **separadas por comas las definiciones de cada columna** de la tabla, la definición de una columna **consta de** su **nombre, el tipo de dato** que tiene **y** podemos añadir **si queremos una serie de especificaciones** que deberán cumplir los datos almacenados en la columna, **después** de definir cada una de las columnas que compone la tabla **se pueden añadir** una serie de **restricciones**, esas restricciones son las mismas que se pueden indicar para cada columna pero ahora **pueden afectar a más de una columna** por eso tienen una sintaxis ligeramente diferente.

- Una **<u>restricción de tipo 1</u>** se utiliza para indicar una característica de la columna que estamos definiendo, tiene la siguiente sintaxis:
  - La cláusula **NOT NULL** indica que **la columna no podrá contener un valor nulo**, es decir que se deberá rellenar obligatoriamente y con un valor válido (equivale a la propiedad requerido Sí de las propiedades del campo).
  - La cláusula **CONSTRAINT** sirve para definir una **restricción** que se podrá eliminar cuando queramos sin tener que borrar la columna. A cada restricción se le asigna un nombre que se utiliza para identificarla y para poder eliminarla cuando se quiera.
  - La cláusula **PRIMARY KEY** se utiliza para definir la columna como **clave principal de la tabla**. Esto supone que **la columna no puede contener valores nulos ni pueden haber valores duplicados** en esa columna, es decir que dos filas no pueden tener el mismo valor en esa columna.

    En una tabla **no puede haber varias claves principales**, por lo que no podemos incluir la cláusula **PRIMARY KEY** más de una vez, en caso contrario la sentencia da un error. No hay que confundir la definición de varias claves principales con la definición de una clave principal compuesta por varias columnas, esto último sí está permitido y se define con una restricción de tipo 2.
  - La cláusula **UNIQUE** sirve para definir un **índice único** sobre la columna. Un índice único es un índice que **no permite valores duplicados**, es decir que si una columna tiene definida un restricción de **UNIQUE** no podrán haber dos filas con el mismo valor en esa columna. Se suele emplear para que el sistema compruebe el mismo que no se añaden valores que ya existen, por ejemplo si en una tabla de clientes queremos asegurarnos que dos clientes no puedan tener el mismo D.N.I. y la tabla tiene como clave principal un código de cliente, definiremos la columna dni con la restricción de **UNIQUE**.
  - La última restricción que podemos definir sobre una columna es la de clave foránea, una **clave foránea es una columna** o conjunto de columnas **que contiene un valor que hace referencia a una fila de otra tabla**, en una restricción de tipo 1 se puede definir con la cláusula **REFERENCES**, después de la palabra reservada indicamos a qué tabla hace referencia, opcionalmente podemos indicar entre paréntesis el nombre de la columna donde tiene que buscar el valor de referencia, por defecto coge la clave principal de la tabla2, si el valor que tiene que buscar se encuentra en otra columna de tabla2, entonces debemos indicar el nombre de esta columna entre paréntesis, además sólo podemos utilizar una columna que esté definida con una restricción de **UNIQUE**, si la columna2 que indicamos no está definida sin duplicados, la sentencia **CREATE** nos dará un error.

Ejemplo: Tenemos el siguiente esquema Relacional, crear las tablas




```
DROP DATABASE IF EXISTS aaEjemplos;
CREATE DATABASE IF NOT EXISTS aaEJemplos;
USE aaEjemplos;


CREATE TABLE  IF NOT EXISTS Equipos(
nombre VARCHAR(30) PRIMARY KEY,
presidente VARCHAR(50) NOT NULL,
estadio VARCHAR(50) NOT NULL,
anhoFundacion YEAR (4) NOT NULL,
numSocios INTEGER) Engine = InnoDB;

CREATE TABLE IF NOT EXISTS Futbolistas(
numFicha INTEGER PRIMARY KEY,
nombre VARCHAR(20) NOT NULL,
apellidos VARCHAR(50) NOT NULL,
fecNacimiento DATE NOT NULL,
peso INTEGER NOT NULL,
altura INTEGER NOT NULL) Engine = InnoDB;

CREATE TABLE IF NOT EXISTS Contrato(
numContrato INTEGER PRIMARY KEY,
fichaAnual INTEGER NOT NULL,
duracion INTEGER NOT NULL,
fechaContrato DATE NOT NULL,
clausula VARCHAR(20) NOT NULL,
nombreEquipo VARCHAR(30) NOT NULL REFERENCES Equipos (nombre),
numFicha INTEGER NOT NULL REFERENCES Futbolistas (numFicha)
)ENGINE = InnoDB;
```

Con este ejemplo estamos creando la base de datos aaEjemplos y en ella 3 tablas *Equipos, Futbolistas y Contrato.* La tabla *Equipo* está compuesta por: una columna llamada *nombre* de tipo VARCHAR(30) definida como clave principal, una columna *presidente* que puede almacenar hasta 50 caracteres alfanuméricos y no puede contener valores nulos, una columna *estadio* de hasta 50 caracteres, no puede ser nula y no se puede repetir (UNIQUE), una columna *anhoFundacion de tipo YEAR* con un formato de 4 dígitos y una columna *numSocios* de tipo entero sin ninguna restricción; la tabla *Futbolistas* tiene una serie de columnas que no pueden ser nulas, la columna *numFicha* es la clave principal y la columna *fecNacimiento* es de tipo DATE; la tabla *Contrato* tiene una serie de columnas, la clave principal es numFicha, la columna *nombreEquipo* de tipo VARCHAR(50) es una clave foránea

(CLAVE AJENA o FOREIGN KEY) que hace referencia a la clave principal ***nombre*** de la tabla *Equipos,* y la columna *numFicha* de tipo entero es una clave foránea (CLAVE AJENA o FOREIGN KEY) que hace referencia a la clave principal **numFicha** de la tabla *Futbolistas*.

- Una **restricción de tipo 2** se utiliza **para definir una característica que afecta a una columna o a una combinación de columnas** de la tabla que estamos definiendo, **se escribe después de haber definido todas las columnas** de la tabla.

Tiene la siguiente **sintaxis**:

La sintaxis de una restricción de tipo 2 es muy similar a la **CONSTRAINT** de una restricción 1 la diferencia es que ahora tenemos que indicar sobre qué columnas queremos definir la restricción. Se utilizan obligatoriamente las restricciones de tipo 2 cuando la restricción afecta a un grupo de columnas o cuando queremos definir más de una **CONSTRAINT** para una columna (sólo se puede definir una restricción1 en cada columna).

La cláusula **PRIMARY KEY** se utiliza para definir la **clave principal** de la tabla. Después de las palabras **PRIMARY KEY** se indica entre paréntesis el nombre de la columna o las columnas que forman la clave principal. Las columnas que forman la clave principal no pueden contener valores nulos ni puede haber valores duplicados de la combinación de columnas.

En una tabla **no puede haber varias claves principales**, por lo que no podemos indicar la cláusula **PRIMARY KEY** más de una vez, en caso contrario la sentencia da un error.

La cláusula **UNIQUE** sirve para definir un **índice único** sobre una columna o sobre una combinación de columnas. Un índice único es un índice que **no permite valores duplicados**. Si el índice es sobre varias columnas no se puede repetir la misma combinación de valores en dos o más filas. Se suele emplear para que el sistema compruebe el mismo que no se añaden valores que ya existen.

La cláusula **FOREIGN KEY** sirve para definir una **clave foránea** sobre una columna o una combinación de columnas. Una clave foránea es una columna o conjunto de columnas que **contiene un valor que hace referencia a una fila de otra tabla**, en una restricción 1 se puede definir con la cláusula **REFERENCES**. Para definir una clave foránea en una restricción de tipo 2 debemos empezar por las palabras **FOREIGN KEY** después indicamos entre paréntesis la/s columna/s que es clave foránea, a continuación la palabra reservada **REFERENCES** seguida del nombre de la tabla a la que hace referencia, opcionalmente podemos indicar entre paréntesis el nombre de la/s columna/s donde tiene que buscar el valor de referencia, por defecto coge la clave principal de la tabla2, si el valor que tiene que buscar se encuentra en otra/s columna/s de tabla2, entonces debemos escribir el nombre de esta/s columna/s entre paréntesis, además sólo podemos utilizar una columna (o combinación de columnas) que esté definida con una restricción de **UNIQUE**, de lo contrario la sentencia **CREATE TABLE** nos dará un error.

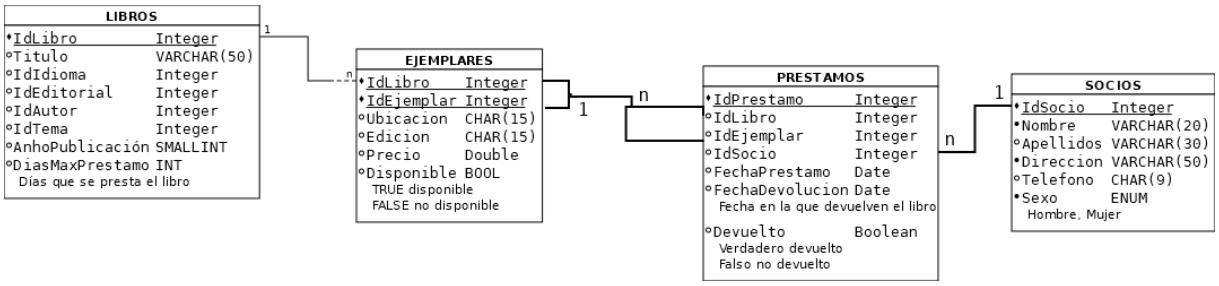
Ejemplo: Mismo ejemplo anterior utilizando una restricción de tipo 2.

```
CREATE TABLE  IF NOT EXISTS Equipos1(
nombre VARCHAR(30),
presidente VARCHAR(50) NOT NULL,
estadio VARCHAR(50) NOT NULL,
anhoFundacion YEAR (4) NOT NULL,
numSocios INTEGER,
PRIMARY KEY (nombre),
CONSTRAINT uk UNIQUE (estadio)) Engine = InnoDB;

CREATE TABLE IF NOT EXISTS Futbolistas1(
numFicha INTEGER,
nombre VARCHAR(20) NOT NULL,
apellidos VARCHAR(50) NOT NULL,
fecNacimiento DATE NOT NULL,
peso INTEGER NOT NULL,
altura INTEGER NOT NULL,
CONSTRAINT pk PRIMARY KEY (numFicha)) Engine = InnoDB;

CREATE TABLE IF NOT EXISTS Contrato1(
numContrato INTEGER,
fichaAnual INTEGER NOT NULL,
duracion INTEGER NOT NULL,
fechaContrato DATE NOT NULL,
clausula VARCHAR(20) NOT NULL,
nombreEquipo VARCHAR(30) NOT NULL,
numFicha INTEGER NOT NULL,
CONSTRAINT pk PRIMARY KEY (numContrato),
CONSTRAINT fkEquipos FOREIGN KEY (nombreEquipo) REFERENCES Equipos1
(nombre),
FOREIGN KEY (numFicha) REFERENCES Futbolistas1 (numFicha)
)ENGINE = InnoDB;
```

Ejemplo: Crear las tablas del siguiente esquema Relacional. Hay una tabla que tiene una PRIMARY KEY formada por dos columnas.



```
CREATE TABLE Libros(
IdLibro INTEGER PRIMARY KEY,
Titulo VARCHAR(50),
IdIdioma INTEGER,
IdEditorial INTEGER,
IdAutor INTEGER,
IdTema INTEGER,
AnhoPublicacion SMALLINT,
DiasMaxPrestamo SMALLINT
)ENGINE = InnoDB;

CREATE TABLE EJEMPLARES(
IdLibro INTEGER,
IdEjemplar INTEGER,
Ubicacion VARCHAR(20),
Edicion VARCHAR(15),
Precio DOUBLE,
PRIMARY KEY (IdLibro, IdEjemplar),
FOREIGN KEY (IdLibro) REFERENCES Libros (IdLibro))ENGINE = InnoDB;

CREATE TABLE SOCIOS(
IdSocio INTEGER PRIMARY KEY,
Nombre VARCHAR(20),
Apellidos VARCHAR(30),
Direccion VARCHAR(50),
Telefono CHAR(9),
Sexo ENUM ('Hombre', 'Mujer')ENGINE = InnoDB;

CREATE TABLE PRESTAMOS(
IdPrestamo INTEGER PRIMARY KEY,
IdLibro INTEGER,
IdEjemplar INTEGER,
IdSocio INTEGER,
FechaPrestamo DATE,
FechaDevolucion DATE,
Devuelto BOOL,
FOREIGN KEY (IdLibro, IdEjemplar) REFERENCES Ejemplares (IdLibro,
IdEjemplar),
FOREIGN KEY (IdSocio) REFERENCES Socios (IdSocio))ENGINE = InnoDB;
```

Motores de almacenamiento

La sintaxis de esta opción es:

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

Podemos usar indistintamente *ENGINE* o *TYPE*, pero la forma recomendada es *ENGINE* ya que la otra desapareció en la versión 5 de **MySQL**.

Hay seis motores de almacenamiento disponibles. 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**. Los motores son:

- **BerkeleyDB o BDB:** tablas de transacción segura con bloqueo de página.
- **HEAP o MEMORY:** tablas almacenadas en memoria.
- **ISAM:** motor original de **MySQL**.
- **InnoDB:** tablas de transacción segura con bloqueo de fila y claves foráneas.
- **MERGE o MRG_MyISAM:** una colección de tablas MyISAM usadas como una única tabla.
- **MyISAM:** el nuevo motor binario de almacenamiento portable que reemplaza a ISAM.

Generalmente usaremos tablas **MyISAM** o tablas **InnoDB**.

A veces, cuando se requiera una gran optimización, creemos tablas temporales en memoria.

- Algunos motores de almacenamiento permiten especificar un tipo de índice cuando este es creado. La sintaxis para el especificador de tipo_indice es

  USING nombre_tipo.

  Por ejemplo:

  CREATE TABLE lookup (id INT, INDEX USING BTREE (id)) ENGINE = MEMORY;

- Sólo las tablas de tipos **MyISAM**, **InnoDB**, **BDB** y **MEMORY** soportan índices en columnas que contengan valores *NULL*. En otros casos se deben declarar esas columnas como *NOT NULL* o se producirá un error.
- Con la sintaxis *col_name(longitud)* en una especificación de un índice, se puede crear un índice que use sólo los primeros 'longitud' bytes de una columna *CHAR* o *VARCHAR*. Esto puede hacer que el fichero de índices sea mucho más pequeño. Las tablas de tipos **MyISAM** y (desde MySQL 4.0.14) **InnoDB** soportan indexación en columnas *BLOB* y *TEXT*. Cuando se usa un índice en una columna *BLOB* o *TEXT* se debe especificar siempre la longitud de los índices. Por ejemplo:

  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));

  Los prefijos pueden tener hasta 255 bytes de longitud (o 1000 bytes para tablas **MyISAM** y **InnoDB**). Los límites de los prefijos se miden en bytes, aunque la longitud del prefijo en sentencias **CREATE INDEX** se interpretan como número de caracteres. Hay que tener esto en cuenta cuando se especifica una longitud de prefijo para una columna que usa un conjunto de caracteres multi-byte.

- Se pueden crear índices especiales *FULLTEXT*. Se usan para búsquedas de texto completo. Sólo las tablas de tipo **MyISAM** soportan índices *FULLTEXT*. Sólo pueden ser creados desde columnas *CHAR*, *VARCHAR* y *TEXT*. La indexación siempre se hace sobre la columna completa; la indexación parcial no está soportada y cualquier longitud de prefijo es ignorada si se especifica.
- Se pueden crear índices *SPATIAL* en columnas de tipo espacial. Los tipos espaciales se soportan sólo para tablas **MyISAM** y las columnas indexadas deben declararse como *NOT NULL*.
- Las tablas **InnoDB** soportan verificación para restricciones de claves foráneas. Hay que tener en cuenta que la sintaxis para *FOREIGN KEY* en **InnoDB** es mucho más restrictiva que la sintaxis presentada antes: las columnas en la tabla referenciada deben ser nombradas explícitamente. **InnoDB** soporta las acciones *ON DELETE* y *ON UPDATE* para claves ajenas. Para otros tipos de tablas, el servidor MySQL verifica la sintaxis de *FOREIGN KEY*, *CHECK* y *REFERENCES* en comandos *CREATE TABLE*, pero no se toma ninguna acción.

Motor de almacenamiento	Descripción
BDB	Tablas de transacción segura con bloqueo de página.
BerkeleyDB	Alias para BDB.
HEAP	Los datos para esta tabla sólo se almacenan en memoria.
ISAM	El motor de almacenamoento original de MySQL.
InnoDB	Tablas de transacción segura con bloqueo de fila y claves foráneas.
MEMORY	Alias para HEAP.
MERGE	Una colección de tablas **MyISAM** usadas como una tabla.
MRG_MyISAM	Un alias para **MERGE**.
MyISAM	El nuevo motor binario de almacenamiento portable que reemplaza a **ISAM**.

Si se especifica un tipo de tabla, y ese tipo particular no está disponible, MySQL usará el tipo **MyISAM**. Por ejemplo, si la definición de la tabla incluye la opción *ENGINE=BDB* pero el servidor MySQL no soporta tablas **BDB**, la tabla se creará como una tabla **MyISAM**. Esto hace posible tener un sistema de réplica donde se tienen tablas operativas en el maestro pero las tablas creadas en el esclavo no son operativas (para obtener mayor velocidad). En MySQL 4.1.1 se obtiene un aviso si el tipo de tabla especificado no es aceptable.

Otras opciones de la sentencia CREATE TABLE

Las otras opciones de tabla se usan para optimizar el comportamiento de la tabla. En la mayoría de los casos, no se tendrá que especificar ninguna de ellas. Las opciones trabajan con todos los tipos de tabla, salvo que se indique otra cosa:

Option	**Descripción**
AUTO_INCREMENT	El valor inicial *AUTO_INCREMENT* que se quiere seleccionar para la tabla. Sólo funciona en tablas **MyISAM**. Para poner el primer valor de un auto-incrementado para una tabla **InnoDB**, insertar una fila vacía con un valor una unidad menor, y después borrarla.
AVG_ROW_LENGTH	Una aproximación de la longitud media de fila de la tabla. Sólo se necesita esto para tablas largas con tamaño de registro variable.
CHECKSUM	Ponerlo a 1 si se quiere que MySQL mantenga un checksum para todas las filas (es decir, un checksum que MySQL actualiza automáticamente cuando la tabla cambia. Esto hace la tabla un poco más lenta al actualizar, pero hace más sencillo localizar tablas corruptas. La sentencia **CHECKSUM TABLE** devuelve el valor del checksum. (Sólo **MyISAM**).
COMMENT	Un comentario de 60 caracteres para la tabla.
MAX_ROWS	Número máximo de filas que se planea almacenar en la tabla.
MIN_ROWS	Mínimo número de filas que se planea almacenar en la tabla.
PACK_KEYS	Ponerlo a 1 si se quiere tener índices más pequeños. Esto normalmente hace que las actualizaciones sean más lentas y las lecturas más rápidas. Ponerlo a 0 desactiva cualquier empaquetado de claves. Ponerlo a *DEFAULT* indica al motor de almacenamiento que sólo empaquete columnas *CHAR /VARCHAR* largas. (Sólo **MyISAM** y **ISAM**). Si no se usa *PACK_KEYS*, por defecto sólo se empaquetan cadenas, no números. Si se usa *PACK_KEYS=1*, los números serán empaquetados también. Cuando se empaquetan claves de números binarios, MySQL usa compresión con prefijo: • Cada clave necesita un byte extra para indicar cuantos bytes de la clave anterior son iguales en la siguiente. • El puntero a la fila se almacena guardando primero el byte de mayor peso directamente después de la clave, para mejorar la compresión. Esto significa que si existen muchas claves iguales en dos filas consecutivas, todas las claves "iguales" que sigan generalmente sólo necesitarán dos bytes (incluyendo el puntero a la fila).

	Comparar esto con el caso corriente, donde las claves siguientes tendrán tamaño_almacenamiento_clave + tamaño_puntero (donde el tamaño del puntero normalmente es 4). Por otra parte, sólo se obtendrá un gran beneficio de la compresión prefija si hay muchos números que sean iguales. Si todas las claves son totalmente distintas, se usará un byte más por clave, si la clave no es una que pueda tener valores NULL. (En ese caso, la longitud de la clave empaquetada será almacenada en el mismo byte que se usa para marcar si la clave es NULL.)
PASSWORD	Encripta el fichero `.frm' con un password. Esta opción no hace nada en la versión estándar de MySQL.
DELAY_KEY_WRITE	Poner esto a 1 si se quiere posponer la actualización de la tabla hasta que sea cerrada (sólo **MyISAM**).
ROW_FORMAT	Define cómo deben almacenarse las filas. Actualmente esta opción sólo funciona con tablas **MyISAM**. El valor de la opción puede ser *DYNAMIC* o *FIXED* para formatos de fila estático o de longitud variable.
RAID_TYPE	La opción *RAID_TYPE* permite exceder el límite de 2G/4G para un fichero de datos **MyISAM** (no el fichero de índice) en sistemas operativos que no soportan ficheros grandes. Esta opción es innecesaria y no se recomienda para sistemas de ficheros que soporten ficheros grandes.
UNION	*UNION* se usa cuando se quiere usar una colección de tablas idénticas como una. Esto sólo funciona con tablas *MERGE*. Es necesario tener los privilegios *SELECT*, *UPDATE* y *DELETE* en las tablas a mapear en una tabla *MERGE*.
INSERT_METHOD	Si se quiere insertar datos en una tabla *MERGE*, se debe especificar con *INSERT_METHOD* dentro de qué la tabla se insertará la fila. *INSERT_METHOD* es una opción frecuente sólo en tablas *MERGE*.
DATA DIRECTORY INDEX DIRECTORY	Usando *DATA DIRECTORY='directory'* o *INDEX DIRECTORY='directory'* se puede especificar dónde colocará el motor de almacenamiento el fichero de datos y el de índices. Este directorio debe ser un camino completo al directorio (no un camino relativo). Esto sólo funciona con tablas **MyISAM** desde MySQL 4.0, cuando no se está usando la opción **--skip-symlink**.

ALTER TABLE

La sentencia **ALTER TABLE** sirve para **modificar la estructura de una tabla** que ya existe. Mediante esta instrucción podemos añadir columnas nuevas, eliminar columnas. Al eliminar una columna se pierden todos los datos almacenados en ella.

También nos permite crear nuevas restricciones o borrar algunas existentes. La sintaxis puede parecer algo complicada pero sabiendo el significado de las palabras reservadas la sentencia se aclara bastante; ADD (añade), ALTER (modifica), DROP (elimina), COLUMN (columna), CONSTRAINT (restricción).

La **sintaxis** es la siguiente:

```
ALTER [IGNORE] TABLE tbl_name alter_specification [,
alter_specification ...]

Sintaxis para alter_specification:
ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
| ADD [COLUMN] (create_definition, create_definition,...)
| ADD INDEX [index_name] (index_col_name,...)
| ADD [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
| ADD [CONSTRAINT [symbol]] 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]
| 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]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP INDEX index_name
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col
| CHARACTER SET character_set_name [COLLATE collation_name]
| table_options
```

La **sintaxis** de **restriccion1** es idéntica a la restricción1 de la sentencia **CREATE TABLE.**

La **sintaxis** de **restriccion2** es idéntica a la restricción2 de la sentencia **CREATE TABLE.**

ADD

- La cláusula **ADD COLUMN** (la palabra **COLUMN** es opcional) permite **añadir una columna nueva** a la tabla. Como en la creación de tabla, hay que definir la columna indicando su nombre, tipo de datos que puede contener, y si lo queremos alguna restricción de valor no nulo, clave primaria, clave foránea, e índice único, **restriccion1 es opcional** e indica una restricción de tipo 1 que afecta a la columna que estamos definiendo

Ejemplo: Añadir la columna email de tipo VARCHAR(50) puede ser nula y debe ser única en la tabla socios

```
ALTER TABLE Socios ADD COLUMN email VARCHAR(50) UNIQUE;
```

Cuando añadimos una columna lo mínimo que se puede poner sería:

```
ALTER TABLE Socios ADD fecNacimiento DATE;
```

En este caso la nueva columna admite valores nulos y duplicados

Ejemplo: Añadir la columna sinopsis al principio de la tabla Libros

```
ALTER TABLE Libros ADD COLUMN sipnosis  BLOB first;
```

Ejemplo: Añadir la columna Estudios VARCHAR(40) después de la columna Direccion en la tabla Socios

```
ALTER TABLE Socios ADD COLUMN Estudios VARCHAR(40) AFTER Direccion;
```

- Para **añadir una nueva restricción** en la tabla podemos utilizar la cláusula **ADD restriccion2 (ADD CONSTRAINT…).**

Ejemplo: La columna Ubicacion de la table Ejemplares es única

```
ALTER TABLE Ejemplares ADD CONSTRAINT uk UNIQUE(ubicacion);
```

- Para **añadir un índice** en la tabla podemos utilizar la cláusula **ADD INDEX…….**

Ejemplo: Añadir un índice a tabla Socios para la columna Apellidos

```
ALTER TABLE Socios ADD INDEX indApellidos (apellidos);
```

Ejemplo: Añadir un índice a tabla Socios para las columnas Apellidos y nombre

```
ALTER TABLE Socios ADD INDEX indNbApe (apellidos, nombre);
```

Ejemplo: Añadir un índice a tabla Libros para la columna titulo sin ponerle nombre al índice.

```
ALTER TABLE Libros ADD INDEX (titulo);
```

DROP

- Para **borrar una columna** basta con utilizar la cláusula **DROP COLUMN** (**COLUMN)** es opcional) y el nombre de la columna que queremos borrar, se perderán todos los datos almacenados en la columna.

Ejemplo: Borrar la columna Estudios de la tabla Socios

```
ALTER TABLE Socios DROP COLUMN Estudios;
```

También podemos escribir:

```
ALTER TABLE Socios DROP Estudios;
```

El resultado es el mismo, la columna Estudios desaparece de la tabla Socios.

- Para **borrar una restricción** basta con utilizar la cláusula **DROP CONSTRAINT** y el nombre de la restricción que queremos borrar, en este caso sólo se elimina la definición de la restricción pero los datos almacenados no se modifican ni se pierden.

Ejemplo: Borrar la de clave primaria de la tabla contratos

```
ALTER TABLE Contrato DROP PRIMARY KEY;
```

- Para **borrar un índice DROP INDEX** y el nombre del índice que queremos borrar, en este caso sólo se elimina la definición del índice pero los datos almacenados no se modifican ni se pierden.

Ejemplo: Borrar el índice indApellidos de la tabla Socios

```
ALTER TABLE Socios DROP INDEX indApellidos;
```

Ejemplo: Borrar el índice de la tabla Libros no tiene nombre utilizamos el nombre de la columna

```
ALTER TABLE Libros DROP INDEX titulo;
```

ALTER

ALTER COLUMN Se utiliza para poner o sacar el valor default para una columna

Ejemplo: Poner a la columna duración el valor por defecto 4

```
ALTER TABLE Contrato ALTER COLUMN duracion SET DEFAULT 4;
```

Ejemplo: Borrar el valor por defecto

```
ALTER TABLE Contrato ALTER COLUMN duracion DROP DEFAULT;
```

CHANGE

CHANGE COLUMN se utiliza para cambiar el nombre a una columna, cambio su datatype, o moverlo dentro el schema. Hay que poner siempre el nombre nuevo, el nombre antiguo y el tipo de dato.

Ejemplo: Cambia el nombre de la columna sipnosis por el de sinopsis y lo cambia a un INT

```
ALTER TABLE Libros CHANGE COLUMN sipnosis sinopsis INT;
```

Ejemplo: Cambiar la columna sinopsis después de titulo

```
ALTER TABLE Libros CHANGE COLUMN sinopsis sinopsis BLOB AFTER titulo;
```

MODIFY

MODIFY COLUMN se utiliza para hacer todo CAMBIA la columna , pero sin rebautizarla .

Ejemplo: Cambia el tipo de la columna de Double a INT en la tabla Ejemplares y no admite nulos.

```
ALTER TABLE Ejemplares MODIFY COLUMN Precio INT NOT NULL;
```

Ejemplo: Cambia el tipo de la columna Ubicación a VARCHAR(30) y lo mueve después de la columna precio-

```
ALTER TABLE Ejemplares MODIFY COLUMN Ubicacion VARCHAR(30) AFTER precio;
```

DROP TABLE Eliminar una tabla

La sentencia **DROP TABLE** sirve para **eliminar una tabla**. No se puede eliminar una tabla si está abierta, tampoco la podemos eliminar si el borrado infringe las reglas de integridad referencial (si interviene como tabla padre en una relación y tiene registros relacionados).

La **sintaxis** es la siguiente:

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

Ejemplo: Borrar la tabla Equipos de la Base de datos aaEjemplos

```
DROP TABLE Equipos;
```

```
DROP TABLE IF EXISTS Equipos;
```

Se produce un error debido a las reglas de integridad referencial. Primero deberíamos borrar la tabla Contrato.

```
DROP TABLE IF EXISTS Contrato;
```

```
DROP TABLE IF EXISTS Equipos;
```

CREATE INDEX

La sentencia **CREATE INDEX** sirve para **crear un índice** sobre una o varias columnas de una tabla.

CREATE INDEX permite añadir la mayoría de los índices excepto PRIMARY KEY.

La **sintaxis** es la siguiente:

```
CREATE INDEX nb_indice ON nb_tabla (col_indice);

CREATE UNIQUE INDEX nb_indice ON nb_tabla (col_indice);

CREATE FULLTEXT INDEX nb_indice ON nb_tabla (col_indice);

CREATE SPATIAL INDEX nb_indice ON nb_tabla (col_indice);
```

nb_indice no es opcional como en ALTER TABLE no se puede crear más de un índice con una misma sentencia.

nbindice: **nombre del índice** que estamos definiendo. **En una tabla no pueden haber dos índices con el mismo nombre** de lo contrario da error.

nbtabla: **nombre de la tabla donde definimos el índice**. A continuación entre paréntesis se indica la composición del índice (las columnas que lo forman).

Podemos formar un índice **basado en varias columnas**, en este caso después de indicar la primera columna con su orden, se escribe una coma y la segunda columna también con su orden, así sucesivamente hasta indicar todas las columnas que forman el índice.

Ejemplo: Crear un índice para las columnas apellidos y nombre de la tabla Futbolistas

```
CREATE INDEX ind1 ON Futbolistas (apellidos, nombre);
```

Este índice permite tener ordenadas las filas de la tabla *Futbolistas* de forma que aparezcan los futbolistas ordenados por *apellidos*, dentro de los que tengan los mismos apellidos por el nombre.

Ejemplo: Crear un índice para las columnas apellidos Ascendente, nombre Ascendente y fecha de nacimiento descendente de la tabla Futbolistas

```
CREATE INDEX ind2 ON Futbolistas (apellidos ASC, nombre ASC, fecNacimiento DESC);
```

Crea un índice llamado *ind2* sobre la tabla *Futbolistas* formado por las columnas *apellidos, nombre y fecNacimiento*. Este índice permite tener ordenadas las filas de la tabla *Futbolistas* de forma que aparezcan los futbolistas ordenados por *apellidos de forma ascendente*, dentro de los mismos apellidos por el *nombre* y dentro del mismo *nombre* por *fecNacimiento* y del más joven al más mayor.

```
CREATE UNIQUE INDEX ind3 ON Socios (apellidos, nombre);
```

Al añadir la cláusula **UNIQUE** el índice no permitirá duplicados por lo que no podría tener dos socios con los mismos apellidos y nombre:

```
CREATE INDEX ind3 ON Socios (apellidos, nombre);
```

DROP INDEX

La sentencia **DROP INDEX** sirve para **eliminar un índice** de una tabla. Se elimina el índice pero no las columnas que lo forman.

La **sintaxis** es la siguiente

```
DROP INDEX nb_indice ON nb_tabla;

DROP INDEX 'PRIMARY' ON nb_tabla (col_indice);
```

Ejemplo: Borrar el índice ind3 de la tabla socios

```
DROP INDEX ind3 ON Socios
```

Elimina el índice que habíamos creado en el ejemplo anterior.