

# SQL con SQLite

## Tablas e inserciones de datos

Máster en Data Science

M1967 - Modelos de Datos y Sistemas de Información 2019-2020

# SQL: Structured Query Language

- SQL es el lenguaje que usan los SGBDR para definición de las bases de datos así como para la manipulación de los datos y su consulta.
  - Es un lenguaje declarativo.
  - Existe un estándar del lenguaje.
  - La primera versión nace en 1986-1987 (SQL87). A partir de ese momento, ha habido numerosas revisiones:
    - SQL89, SQL92: varias revisiones.
    - SQL2000: añade expresiones regulares, disparadores, consultas recursivas y algunas características de orientación a objetos.
    - SQL2003: añade soporte para XML, columnas autonuméricas...
    - SQL2005: define de que manera SQL puede usarse conjuntamente con XML y da soporte al lenguaje XQuery.
    - SQL2008: Incluye disparadores INSTEAD OF, la clausula TRUNCATE y otros.
    - SQL2011: Mejoras en las funciones ventana.
    - SQL2016: Permite búsquedas con patrones, da soporte a JSON...

# SQL: Structured Query Language

- Hay un estándar, pero ningún SGBDR le sigue al 100%:
  - Diferentes tipos de datos.
  - Diferentes funciones para realizar agregaciones (sumas, cálculo de medias, etc.).
  - Diferentes formas de definir procedimientos.
  - Diferentes formas de definir funciones.
  - Diferentes mecanismos de seguridad.
  - Etc.
- No obstante, es fácil migrar entre unos y otros. A nivel básico, las consultas, transacciones y otras operaciones son idénticas.

# SQLite



- SQLite es un SGBDR que usa SQL como lenguaje.
- Es autocontenido: la base de datos se guarda en un fichero del sistema de archivos, con extensión .db3
  - Esto da facilidad para poder “llevar contigo” la base de datos.
  - Evita tener que instalar servicios en el Sistema Operativo a los que conectarse.
  - Permite embeber la base de datos con facilidad en cualquier programa.
  - Es “independiente” del Sistema Operativo en el que se trabaja.
- Es libre y gratuito.
- Existen multitud de programas para gestionar bases de datos en SQLite (SQLite Manager, SQLite Studio, SQLite Browser,...).
- Tiene las funcionalidades necesarias para trabajar con bases de datos relacionales.
  - Aunque limitadas. Ofrece pocos tipos de datos y carece de la potencia de otros SGBDR comerciales.

# SQLite Studio

- En esta asignatura, se recomienda trabajar con SQLite Studio:
  - Multiplataforma (Windows, Linux, Mac)
  - Enlace: <https://sqlitestudio.pl/index.rvt?act=download>



**SQLite  
Studio**

# Creación de tablas

- Creación y nombrado de la tabla:

*Creación de la tabla con nombre profesor*

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  fechaNacimiento date not null,  
  fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```

# Creación de tablas

- Atributos o columnas de la tabla:

*Atributos o columnas de la tabla:  
idprofesor, dni, nombre...*

```
CREATE TABLE Profesor(  
    idprofesor int PRIMARY KEY,  
    dni char not null UNIQUE,  
    nombre char not null,  
    apellido1 char not null,  
    apellido2 char null,  
    fechaNacimiento date not null,  
    fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```

# Creación de tablas

- Tipos de columnas:

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  fechaNacimiento date not null,  
  fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```

↓  
*Tipos de la columnas: fecha (date),  
números enteros (int), cadena de  
caractéres (char), etc.*



# Creación de tablas

- Debido a su “simplismo”, SQLite sólo tiene 4 (“5”) tipos de datos para almacenar los valores de las columnas:
  - **TEXT**: almacena datos de tipos texto. Por ejemplo, el nombre de una persona lo almacenaríamos como texto: `nombre TEXT`
  - **INTEGER**: almacena datos numéricos de tipo entero (1, 2, 3...). Por ejemplo, la edad de una persona la almacenaríamos como entero: `edad INTEGER`
  - **REAL**: almacena datos numéricos de tipo real, con decimales. Por ejemplo, el número de créditos que imparte un profesor en una asignatura lo almacenaríamos como real: `creditos REAL`
  - **BLOB**: almacena datos de tipo binario, por ejemplo, ficheros: `ficheroPDF BLOB`
  - **NULL**: tipo de datos nulo. Se puede almacenar en columnas de cualquier otro tipo.

# Creación de tablas

- No obstante, en las tablas creadas con SQLite se puede usar otras definiciones de tipos de datos que tiene sus correspondencia con los 4 tipos de datos existentes, y que permiten poder exportar las bases de datos SQLite a otros sistemas:

Tipos de datos que puede usarse para crear tablas	Correspondencia con los tipos de datos de SQLite
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT
BLOB no datatype specified	BLOB
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC

# Creación de tablas

- Correspondencia con los tipos de datos enteros:

Tipos de datos que puede usarse para crear tablas	Correspondencia con los tipos de datos de SQLite
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER

# Creación de tablas

- Correspondencia con los tipos de datos de tipo texto:

Tipos de datos que puede usarse para crear tablas	Correspondencia con los tipos de datos de SQLite
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT

# Creación de tablas

- Correspondencia con los tipos de datos BLOB:

<b>Tipos de datos que puede usarse para crear tablas</b>	<b>Correspondencia con los tipos de datos de SQLite</b>
BLOB no datatype specified	BLOB

# Creación de tablas

- Correspondencia con los tipos de datos reales:

<b>Tipos de datos que puede usarse para crear tablas</b>	<b>Correspondencia con los tipos de datos de SQLite</b>
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL

# Creación de tablas

- Correspondencia con otros tipos de datos (fecha, hora, decimal, booleano...):

<b>Tipos de datos que puede usarse para crear tablas</b>	<b>Correspondencia con los tipos de datos de SQLite</b>
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC

# Creación de tablas

- Restricciones sobre las columnas:

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  fechaNacimiento date not null,  
  fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```

*PRIMARY KEY: Clave primaria. Es única para cada fila (profesor). Es obligatoria (no admite valores nulos). Siempre hay que definirla. Puede formarse por varios campos.*

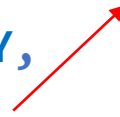


# Creación de tablas

- Restricciones sobre las columnas:

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  fechaNacimiento date not null,  
  fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```

*UNIQUE: Clave alternativa. Es única para cada fila (profesor). Admite valores nulos salvo que se exprese lo contrario con NOT NULL. Puede formarse por varios campos.*




# Creación de tablas

- Restricciones sobre las columnas:

NULL/NOT NULL: indica si es obligatorio introducir un valor en la columna (NOT NULL) o no lo es (NULL).

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  fechaNacimiento date not null,  
  fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```



# Creación de tablas

- Restricciones sobre las columnas:

```
CREATE TABLE Profesor(  
    idprofesor int PRIMARY KEY,  
    dni char not null UNIQUE,  
    nombre char not null,  
    apellido1 char not null,  
    apellido2 char null,  
    fechaNacimiento date not null,  
    fechaAlta date not null CHECK (fechaAlta>fechaNacimiento)  
);
```

*CHECK: limita los posibles valores que puede tomar una columna. En este ejemplo, se obliga a que la fecha de alta de un profesor sea posterior a su fecha de nacimiento.*



# Creación de tablas

- Sintaxis alternativa de la tabla Profesor (RESTRICCIONES explícitas):

```
CREATE TABLE Profesor(  
    idprofesor int,  
    dni char not null,  
    nombre char not null,  
    apellido1 char not null,  
    apellido2 char null,  
    fechaNacimiento date not null,  
    fechaAlta date not null,  
    PRIMARY KEY (idprofesor),  
    UNIQUE (idprofesor),  
    CHECK (fechaAlta > fechaNacimiento)  
);
```

*Las restricciones de las tablas se pueden definir al final de las mismas*

# Creación de tablas

- Se pueden definir diferentes tipos de restricciones CHECK:

- Comparación de valores entre dos columnas:

- CHECK (fechaNacimiento<fechaAlta)

- Rango de valores de una columna (numérica, fecha...)

- CHECK (creditos > 0)
    - CHECK (fechaAlta > '2010-01-01' and fechaAlta<'2018-10-28')

- Posible conjunto de valores (texto...)

- CHECK (tipoProfesor IN ('Ayudante', 'AyudanteDoctor', 'Titular'))

- Expresiones regulares (texto...)

- CHECK (email LIKE ('%@%'))

*Las fechas y los textos  
siempre entre  
comillas*

# Creación de tablas

- Ejemplo de diferentes tipos de CHECKS:

```
CREATE TABLE Profesor(  
    idprofesor int PRIMARY KEY,  
    dni char not null UNIQUE,  
    nombre char not null,  
    apellido1 char not null,  
    apellido2 char null,  
    tipoProfesor char not null,  
    email char not null UNIQUE,  
    fechaNacimiento date not null,  
    fechaAlta date not null,  
    CHECK (fechaAlta > fechaNacimiento),  
    CHECK (tipoProfesor IN ('Ayudante', 'AyudanteDoctor', 'Titular')),  
    CHECK (email LIKE ('%@%'))  
);
```

# Creación de tablas

- Podemos eliminar una tabla con la siguiente instrucción:

**DROP TABLE** Profesor;

**Si la tabla no existe, devolverá un error.**

# Inserción de datos

- Los datos se inserta con la instrucción INSERT:

```
INSERT INTO Profesor (idprofesor, dni, nombre, apellido1, apellido2, tipoProfesor,  
email, fechaNacimiento, fechaAlta)  
VALUES (1, '76576589J', 'Manuel', 'Vázquez', 'De la Sierra', 'Ayudante',  
'manu@kmail.com', '1990-01-02', '2016-06-07');
```

```
INSERT INTO Profesor (idprofesor, dni, nombre, apellido1, apellido2, tipoProfesor,  
email, fechaNacimiento, fechaAlta)  
VALUES (2, '78576182V', 'John', 'Doe', NULL, 'AyudanteDoctor', 'john@kmail.com', '1992-  
01-02', '2015-12-07');
```



# Inserción de datos

*Instrucción INSERT sobre la  
tabla Profesor*

*Declaración de las columnas de la tabla*

```
INSERT INTO Profesor (idprofesor, dni, nombre, apellido1, apellido2, tipoProfesor,  
email, fechaNacimiento, fechaAlta)  
VALUES (1, '76576589J', 'Manuel', 'Vázquez', 'De la Sierra', 'Ayudante',  
'manu@kmail.com', '1990-01-02', '2016-06-07');
```

*Valores insertados. Las fechas y los de tipo texto siempre entre comillas-*

# Inserción de datos

- Los datos se inserta con la instrucción INSERT:

```
INSERT INTO Profesor (idprofesor, dni, nombre, apellido1, apellido2, tipoProfesor,  
email, fechaNacimiento, fechaAlta)  
VALUES (2, '78576182V', 'John', 'Doe', NULL, 'AyudanteDoctor', 'john@kmail.com', '1992-  
01-02', '2015-12-07');
```



*Indica que en esta fila el valor del segundo apellido es NULO (no tiene segundo apellido)*

# Inserción de datos

- Podemos consultar los datos insertados con la siguiente instrucción:

```
SELECT * FROM Profesor;
```

34 -- Consula de datos de la tabla profesor

35 SELECT \* FROM Profesor;

36

Grid view

Form view

✓

✕

⏪

⏩

1

⏴

⏵

Total rows loaded: 2

	idprofe	dni	nombre	apellidc	apellido2	tipoProfesor	email	fechaNacin	fechaAlta
1	1	76576589J	Manuel	Vázquez	De la Sierra	Ayudante	manu@kmail.com	1990-01-02	2016-06-07
2	2	78576182V	John	Doe	NULL	AyudanteDoctor	john@kmail.com	1992-01-02	2015-12-07

# Creación de tablas: Foreign Keys

- Las Foreign Keys (FK) sirven para indicar que un campo de una tabla apunta a la PK de otra tabla:

```
CREATE TABLE dueño(  
  iddueño int PRIMARY KEY,  
  nombre char NOT NULL,  
  apellido1 char NOT NULL,  
  apellido2 char NULL  
);
```

```
CREATE TABLE mascota(  
  idmascota int PRIMARY KEY,  
  nombre char NOT NULL,  
  iddueño int NOT NULL,  
  FOREIGN KEY (iddueño) REFERENCES dueño(iddueño)  
);
```

*Campo FK de la tabla mascota*

*Tabla a la que apunta la FK*

*Campo de la tabla a la que apunta la FK*

# Creación de tablas: Foreign Keys

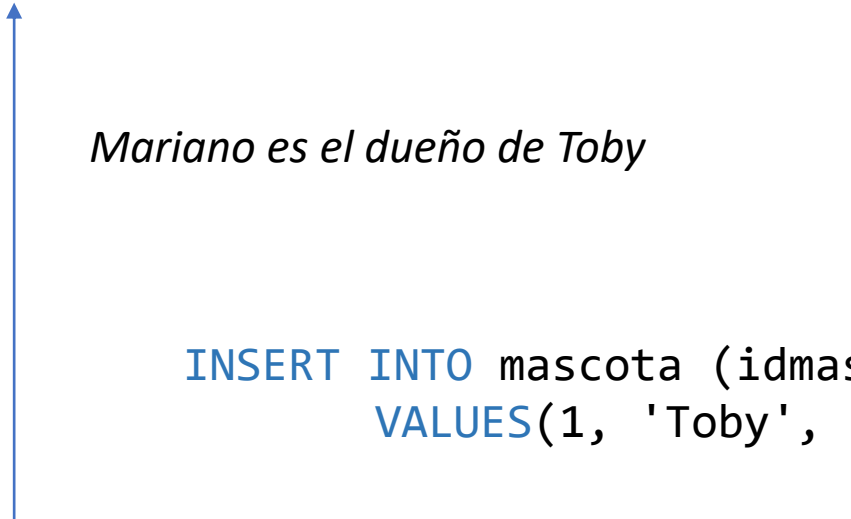
- Al insertar, el valor del campo de la FK tiene que existir en la tabla a la que se apunta:

```
INSERT INTO duenio (idduenio, nombre, apellido1, apellido2)  
VALUES(1, 'Marco', 'Polo', NULL);
```

```
INSERT INTO duenio (idduenio, nombre, apellido1, apellido2)  
VALUES(2, 'Mariano', 'Zapatero', 'Suárez');
```

*Mariano es el dueño de Toby*

```
INSERT INTO mascota (idmascota, nombre, idduenio)  
VALUES(1, 'Toby', 2);
```



# Creación de tablas: Ejercicio 1

**Ahora toca realizar el Ejercicio 1, accesible en Moodle.**


# Creación de tablas: ¿CHECK o TABLA con FK?

- ¿Qué pasaría si quisiésemos ampliar las posibles figuras de profesorado?

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  tipoProfesor char not null,  
  email char not null UNIQUE,  
  fechaNacimiento date not null,  
  fechaAlta date not null,  
  CHECK (fechaAlta > fechaNacimiento),  
  CHECK (tipoProfesor IN ('Ayudante', 'AyudanteDoctor', 'Titular')),  
  CHECK (email LIKE ('%@%'))  
);
```

Tendríamos que ampliar el CHECK  
y añadir nuevos posibles valores:  
Catedrático, Asociado, etc...

**¡PERO LA TABLA YA ESTÁ CREADA,  
MODIFICAR LA RESTRICCIÓN ES  
SUMAMENTE ENGORROSO!**



# Creación de tablas: ¿CHECK o TABLA con FK?

- ¿Qué pasaría si quisiésemos ampliar las posibles figuras de profesorado?

```
CREATE TABLE Profesor(  
  idprofesor int PRIMARY KEY,  
  dni char not null UNIQUE,  
  nombre char not null,  
  apellido1 char not null,  
  apellido2 char null,  
  idtipoprofesor int not null,  
  email char not null UNIQUE,  
  fechaNacimiento date not null,  
  fechaAlta date not null,  
  CHECK (fechaAlta > fechaNacimiento),  
  FOREIGN KEY (idtipoprofesor) REFERENCES  
    TipoProfesor(idtipoprofesor),  
  CHECK (email LIKE ('%@%'))  
);
```

```
CREATE TABLE TipoProfesor(  
  idtipoprofesor int PRIMARY KEY,  
  tipo char not null UNIQUE  
);
```

SOLUCIÓN: Añadimos una tabla TipoProfesor que almacene los tipos de profesor que existen (Ayudante, AyudanteDoctor...), y en la tabla Profesor les hacemos referencia con un campo FK:



# Creación de tablas: ¿CHECK o TABLA con FK?

```
INSERT INTO Profesor (idprofesor, dni, nombre, apellido1, apellido2,
tipoProfesor, email, fechaNacimiento, fechaAlta)
VALUES (1, '76576589J', 'Manuel', 'Vázquez', 'De la Sierra', 2,
'manu@kmail.com', '1990-01-02', '2016-06-07');
```

```
INSERT INTO Profesor (idprofesor, dni, nombre, apellido1, apellido2,
idtipoprofesor, email, fechaNacimiento, fechaAlta)
VALUES (2, '78576182V', 'John', 'Doe', NULL, 1, 'john@kmail.com', '1992-01-
02', '2015-12-07');
```

*Manuel es Ayudante Doctor, y  
John es Ayudante.*

*De esta forma, si se añade una nueva figura  
de profesorado, sólo hay que hacer un  
INSERT en la tabla TipoProfesor con los datos  
de esta nueva figura, sin modificar la  
estructura de la BD.*

```
INSERT INTO TipoProfesor (idtipoprofesor, tipo)
VALUES(1, 'Ayudante');
INSERT INTO TipoProfesor (idtipoprofesor, tipo)
VALUES(2, 'AyudanteDoctor');
```

# Creación de tablas: Ejercicio 2


**Ahora toca realizar el Ejercicio 2, accesible en Moodle.**

# Creación de tablas: ¿relación N a N?

- Hemos visto como hacer referencias con FKs, cuando estas referencias son 1 a 1 o 1 a N (una plan de estudios tiene un solo profesor responsable, una mascota sólo tiene un dueño...)
- ¿Qué tendríamos que hacer si tuviésemos que implementar una relación N a N?
  - Vamos a suponer, siguiendo el ejemplo de las mascotas, que una mascota puede tener varios dueños.
  - La implementación utilizada anteriormente, y que se muestra a continuación, no es válida, ya que obliga a que una mascota tenga un solo dueño:

```
CREATE TABLE duenio(  
  idduenio int PRIMARY KEY,  
  nombre char NOT NULL,  
  apellido1 char NOT NULL,  
  apellido2 char NULL  
);
```

```
CREATE TABLE mascota(  
  idmascota int PRIMARY KEY,  
  nombre char NOT NULL,  
  idduenio int NOT NULL,  
  FOREIGN KEY (idduenio) REFERENCES duenio(idduenio)  
);
```



# Creación de tablas: ¿relación N a N?

- Solución: crear una tabla intermedia que almacene las relaciones entre mascotas y dueños:

```
CREATE TABLE mascotaDuenio(  
  idmascota int NOT NULL,  
  idduenio int NOT NULL,  
  FOREIGN KEY (idmascota) REFERENCES mascota(idmascota)  
  FOREIGN KEY (idduenio) REFERENCES duenio(idduenio),  
  PRIMARY KEY (idmascota, idduenio)  
);
```

```
CREATE TABLE duenio(  
  idduenio int PRIMARY KEY,  
  nombre char NOT NULL,  
  apellido1 char NOT NULL,  
  apellido2 char NULL  
);
```

*En la tabla intermedia, se almacenan pares de valores (mascota/dueño). De esta forma, una mascota puede tener varios dueños, y un dueño tener varias mascotas. La PK es el par de campos.*

```
CREATE TABLE mascota(  
  idmascota int PRIMARY KEY,  
  nombre char NOT NULL  
);
```

# Creación de tablas: ¿relación N a N?

- Con esta solución, podríamos realizar la siguiente inserción de datos:

```
INSERT INTO duenio (idduenio, nombre, apellido1, apellido2)
VALUES(1, 'Marco', 'Polo', NULL);
```

```
INSERT INTO duenio (idduenio, nombre, apellido1, apellido2)
VALUES(2, 'Mariano', 'Zapatero', 'Suárez');
```

```
INSERT INTO mascota (idmascota, nombre)
VALUES(1, 'Toby');
```

```
INSERT INTO mascota (idmascota, nombre)
VALUES(2, 'Balto');
```

```
INSERT INTO mascota (idmascota, nombre)
VALUES(3, 'Jake the dog');
```

```
INSERT INTO MascotaDuenio(idmascota, idduenio)
VALUES(1, 1);
```

→ Marco es dueño de Toby

```
INSERT INTO MascotaDuenio(idmascota, idduenio)
VALUES(1, 2);
```

→ Mariano es dueño de Toby

```
INSERT INTO MascotaDuenio(idmascota, idduenio)
VALUES(2, 1);
```

→ Marco es dueño de Balto

*Ahora, los dueños de Toby son Marco y Mariano. Además, Mariano es el único dueño de Balto. Por ahora, Jake the dog no tiene dueño.*

# Creación de tablas: Ejercicio 3

**Ahora toca realizar el Ejercicio 3, accesible en Moodle.**

# Modificación de tablas con ALTER TABLE

- La estructura de las tablas puede ser modificada.
- En SQLite, las opciones son bastante limitadas. Podemos, por ejemplo, añadir nuevas columnas con sus restricciones.
- La siguiente instrucción añade una nueva columna a la tabla Mascota:

The diagram illustrates the components of the SQL statement `ALTER TABLE mascota ADD genero char NOT NULL CHECK (genero in ('M','F')) DEFAULT 'M';`. Brackets and labels identify each part: *Tabla a modificar* points to `ALTER TABLE mascota`; *Columna nueva junto con tipo y NULL/NOT NULL* points to `ADD genero char NOT NULL`; *Restricción de valores de la columna* points to `CHECK (genero in ('M','F'))`; and *Valor por defecto de la columna* points to `DEFAULT 'M';`.

```
ALTER TABLE mascota ADD genero char NOT NULL CHECK (genero in ('M','F')) DEFAULT 'M';
```

*Tabla a modificar*

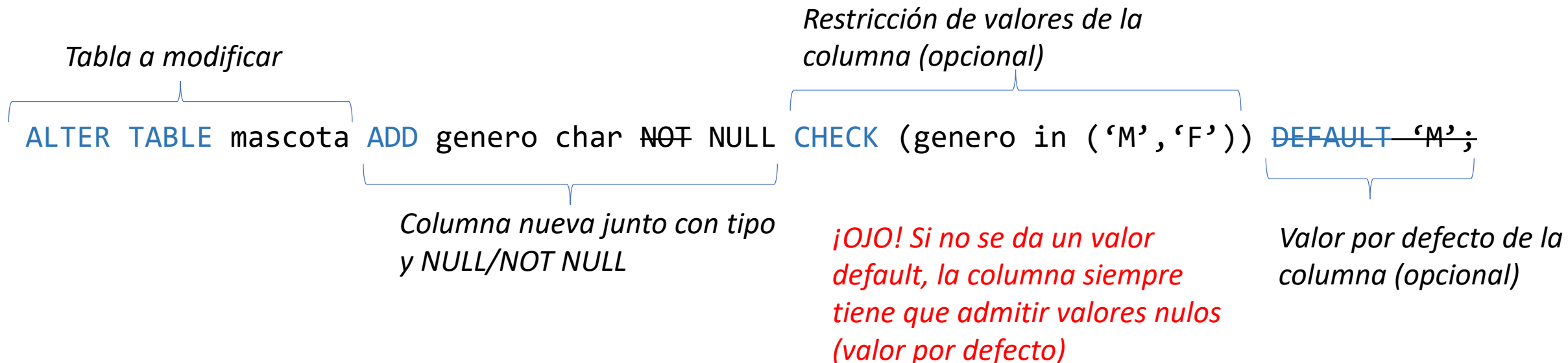
*Restricción de valores de la columna*

*Columna nueva junto con tipo y NULL/NOT NULL*

*Valor por defecto de la columna*

# Modificación de tablas con ALTER TABLE

- La estructura de las tablas puede ser modificada.
- En SQLite, las opciones son bastante limitadas. Podemos, por ejemplo, añadir nuevas columnas con sus restricciones.
- La siguiente instrucción añade una nueva columna a la tabla Mascota:





# Creación de tablas: Práctica 1 (evaluable)

**Ahora toca realizar la Práctica 1 (evaluable), accesible en Moodle.**