

BASES DE DATOS

Administración de Desarrollo de Aplicaciones

Multiplataforma

Administración de Desarrollo de Aplicaciones Web

GESTIÓN DE BASES DE DATOS

Administración de Sistemas Informáticos en Red

MODELO FÍSICO

LENGUAJE DE DESCRIPCIÓN DE DATOS



Apuntes

Luis Dorado Garcés

Basado en el trabajo de Alba Tortosa López

Contenido

1	Introducción	2
1.1	El lenguaje SQL.....	2
1.1.1	Elementos del lenguaje. Normas de escritura.	2
1.1.2	Criterios de notación	3
1.2	El valor NULL.....	4
1.3	Los índices de una tabla en MySQL	5
2	Lenguaje de definición de datos (DDL) I	6
2.1	Crear una Base de datos	6
2.2	Eliminar de una Base de datos.....	6
2.3	Listar bases de datos	7
2.4	Usar una base de datos	7
2.5	<i>Ampliación: Consultar qué tabla estamos usando actualmente</i>	7
2.6	Crear una tabla	7
2.7	Tipos de datos.....	8
2.7.1	Texto.....	8
2.7.2	Número	8
2.7.3	Temporal	9
2.7.4	Lista de valores	10
2.7.5	Si / No	11
2.7.6	Otros tipos	11
2.7.7	Ampliación de tipos de datos	12
2.8	Restricciones básicas	13
2.8.1	Restricción NOT NULL.....	14
2.8.2	Restricción PRIMARY KEY	14
2.8.3	Restricción UNIQUE	15
2.8.4	Nombres de las restricciones (cláusula CONSTRAINT)	16
2.8.5	Restricción DEFAULT.....	17
2.9	Eliminar una tabla	17
2.10	Modificar una tabla	18
2.10.1	Cambiar de nombre una tabla	18
2.10.2	Cambiar de nombre una columna.....	18
2.10.3	Modificar una columna	18
2.10.4	Añadir una columna	19
2.10.5	Eliminar una columna	19
2.10.6	Eliminar clave primaria	19
2.10.7	Crear clave primaria	20
2.10.8	Crear clave alternativa (UNIQUE).....	20
2.10.9	Eliminar clave alternativa (UNIQUE)	20
2.11	Listar tablas.....	20
2.12	Describir una tabla.....	21
2.13	Acceder al código SQL de creación de una tabla	21
3	Lenguaje de definición de datos (DDL) II	22
3.1	Restricción FOREIGN KEY.....	22
3.1.1	Concepto	22
3.1.2	Definir una clave ajena al crear una tabla	22
3.2	Integridad referencial	23
3.2.1	Creación de una tabla (CREATE TABLE) (Ver diapositivas).....	23
3.2.2	Borrado de una tabla (DROP TABLE) (Ver diapositivas)	24
3.2.3	Inserción de un registro (INSERT) (Ver diapositivas)	25
3.2.4	Actualización y borrado de un registro (UPDATE y DELETE) (Ver diapositivas).....	25
3.2.5	Acciones ante una infracción de la integridad (UPDATE/DELETE)	27
3.2.6	Modificar una tabla para incluir una clave ajena.....	29
3.2.7	Modificar una tabla para eliminar clave ajena	29
4	Lenguaje de definición de datos (DDL) III	30
4.1	Restricción AUTO_INCREMENT	30
4.2	Restricción CHECK.....	30

1 Introducción

En las unidades anteriores aprendimos a realizar el diseño conceptual de una BD mediante el Modelo E/R. También vimos cómo hacer el modelo lógico mediante el modelo relacional que se obtenía a partir del modelo E/R y aprendimos a comprobar que dicho modelo estaba normalizado.

Siguiendo con el proceso de desarrollo, lo que debemos hacer ahora es pasar al diseño físico de la BD. Es decir, implementar la Base de Datos. Para ello, se programarán las diferentes tablas que constituirán la Base de Datos, se introducirán los datos y, más adelante, se construirán las consultas. Todo ello se hará programando en el lenguaje más extendido para la definición y manipulación de datos en SGBDR: **SQL**.

Las prácticas del módulo de Gestión Bases de Datos se van a realizar utilizando el **Sistema de Gestión de Bases de Datos Relacional (RDBMS) MySQL**.

1.1 El lenguaje SQL

SQL (Structured Query Language) es el lenguaje fundamental de los SGBD relacionales. Es uno de los lenguajes más utilizados en informática en todos los tiempos. Es un lenguaje declarativo y por tanto, lo más importante es definir **qué** se desea hacer, y **no cómo** hacerlo. De esto último ya se encarga el SGBD.

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. Así es, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aunque SQL está estandarizado, siempre es recomendable revisar la documentación del SGBD con el que estemos trabajando para conocer su sintaxis concreta, ya que algún comando, tipo de dato, etc., puede no seguir el estándar.

1.1.1 Elementos del lenguaje. Normas de escritura.

Imagínate que cada programador utilizara sus propias reglas para escribir. Esto sería un caos. Es muy importante establecer los elementos con los que vamos a trabajar y unas normas que seguir. El lenguaje SQL está compuesto por comandos, cláusulas, operadores, funciones y literales. Todos estos elementos se combinan en las instrucciones y se utilizan para crear, actualizar y manipular bases de datos. Estos conceptos son bastante amplios por eso será mejor que vayamos por partes.

COMANDOS: Van a ser las instrucciones que se pueden crear en SQL. Se pueden distinguir en tres grupos que veremos con más detenimiento a lo largo de las siguientes unidades:

- De definición de datos (DDL, Data Definition Language), que permiten crear y definir nuevas bases de datos, tablas, campos, etc.
- De manipulación de datos (DML, Data Manipulation Language), que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.
- De control y seguridad de datos (DCL, Data Control Language), que administran los derechos y restricciones de los usuarios.

CLÁUSULAS: Llamadas también condiciones o criterios, son palabras especiales que permiten modificar el funcionamiento de un comando.

OPERADORES: Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (<, >, <=, >=, AND, OR, etc.).

FUNCIONES: Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario.

LITERALES: Les podemos llamar también constantes y serán valores concretos, como por ejemplo un número, una fecha, un conjunto de caracteres, etc.

Y tendremos que seguir unas normas sencillas pero primordiales:

- Todas las instrucciones terminan con un signo de punto y coma.
- No se distingue entre mayúsculas y minúsculas.
- Cualquier comando puede ser partido con saltos de línea o espacios para facilitar su lectura y comprensión.
- Los comentarios de bloque comienzan por `/*` y terminan con `*/` (excepto en algunos SGBD).
- Los comentarios de línea: comienzan por `--` y terminan al final de línea

Ejemplos:

```
/*
    Esto es un comentario
    de varias líneas.
    Fin.
*/
-- Esto es un comentario de una línea
-- La siguiente línea es una instrucción.
CREATE DATABASE pruebas;
```

1.1.2 Criterios de notación

La notación utilizada para la sintaxis de los comandos de SQL es la siguiente:

- Palabras clave de la sintaxis SQL en **MAYÚSCULAS**.
- La barra vertical `|` exige el uso de solo uno de los elementos que une.
- Los corchetes `[]` indican opcionalidad.
- Las llaves `{ }` delimitan alternativas separadas por `|` de las que se debe elegir una.
- Los puntos suspensivos `...` indican repetición varias veces de la opción anterior.
- Las palabras en *minúsculas* y *cursiva* indican que allí puede ir un nombre u identificador elegido por el usuario o una estructura compleja con su propia sintaxis.

Convención	Se usa para
MAYÚSCULAS	Palabras clave de Transact-SQL.
(barra vertical)	Separa los elementos de sintaxis escritos entre corchetes o llaves. Solo puede utilizar uno de los elementos.
[] (corchetes)	Elementos opcionales de sintaxis. No escriba los corchetes.
{ } (llaves)	Elementos obligatorios de sintaxis. No escriba las llaves.

Ejemplo:

```
UPDATE [ LOW_PRIORITY | HIGH_PRIORITY ] [ IGNORE ] table_reference
    SET assignment_list
    [ WHERE where_condition ]
    [ ORDER BY ... ]
    [ LIMIT row_count ]
```

Según esta sintaxis lo siguiente sería válido:

```
UPDATE HIGH_PRIORITY Empleados
    SET Sueldo = Sueldo * 1.10;

    WHERE Departamento = 'Ventas'
```

Nota: Si os pica la curiosidad os diré que este código sirve para aumentar un 10% el salario de todos los empleados de la tabla `Empleados` que pertenecen al departamento `Ventas`.

1.2 El valor NULL

¿Qué sucede si al guardar los datos de los Usuarios hay algún dato que no tengo o no necesito guardarlo porque no corresponde?

Independientemente del dominio al que pertenezca un campo, éste puede tomar un valor especial denominado **NULO** (**NULL** en inglés) que designará la ausencia de dato.

Cuando por cualquier motivo se desconoce el valor de un campo, por ejemplo, desconocemos el teléfono del usuario, o bien ese campo carece de sentido (siguiendo con el mismo ejemplo, puede que el usuario no tenga teléfono), podemos asignar a ese campo el valor especial NULO.

Pero ten en cuenta una cosa, no es lo mismo valor NULO que ESPACIO EN BLANCO. Tampoco será lo mismo valor NULO que el valor CERO.

Un ordenador tomará un espacio en blanco como un carácter como otro cualquiera. Por tanto, si introducimos el carácter "espacio en blanco" estaríamos introduciendo un valor que pertenecería al dominio **texto** y sería distinto al concepto "ausencia de valor" que sería **no incluir nada** (nulo).

1.3 Los índices de una tabla en MySQL

¿Qué son los índices?

Los índices de las tablas ayudan a indexar el contenido de diversas columnas para facilitar la búsqueda de contenido de cuando se ejecutan consultas sobre esas tablas.

De ahí que la creación de índices optimiza el rendimiento de las consultas y a su vez el de la BBDD.

¿Qué tipo de índices hay?

Podemos tener los siguientes tipos de índices en una tabla de MySQL:

- ✓ **Primarios:** equivalen a Únicos y No nulos. Puede ser simple o compuesto. **Tiene que haber uno y solo uno.**
- ✓ **Únicos:** Aquellos cuyos valores no se pueden repetir. Pueden ser simples o compuestos. **Puede haber varios.**
- ✓ Ordinarios
- ✓ De texto completo
- ✓ Parte de campos o columnas

2 Lenguaje de definición de datos (DDL) I

La primera fase del trabajo con cualquier base de datos comienza con sentencias DDL, puesto que antes de poder almacenar y recuperar información debemos definir las estructuras donde almacenar la información. Las estructuras básicas con las que trabaja SQL son las tablas.

Conocer el Lenguaje de Definición de Datos (DDL) es imprescindible para crear, modificar y eliminar objetos de la base de datos (es decir, los metadatos). En el mercado hay suficientes aplicaciones y asistentes que nos facilitan esta labor, a través de una interfaz visual que nos oculta el lenguaje SQL y en los cuales nos limitamos a poner nombres a los campos, elegir el tipo de datos y activar una serie de propiedades.

Es cierto que estas herramientas nos facilitan el trabajo, pero resulta imprescindible comprender y conocer en profundidad el lenguaje, ya que nos veremos en muchas situaciones donde necesitaremos crear un objeto, modificarlo o eliminarlo sin depender de esas herramientas visuales.

Las instrucciones DDL generan acciones que no se pueden deshacer, por eso es conveniente usarlas con precaución y tener copias de seguridad cuando manipulamos la base de datos.

2.1 Crear una Base de datos

Básicamente, la creación de la base de datos consiste en crear las tablas que la componen. Aunque antes de esto tendríamos que definir el contenedor donde van a estar ubicadas. A este contenedor lo denominamos "esquema" o "base de datos".

El comando SQL de creación de una base de datos es **CREATE DATABASE**. Este comando crea una base de datos con el nombre que se indique.

Sintaxis:

```
CREATE DATABASE [IF NOT EXISTS] nombre_bd;
```

Ejemplo:

```
CREATE DATABASE prueba;
/* La siguiente instrucción sólo crea la base de datos si no existe
previamente */
CREATE DATABASE IF NOT EXISTS prueba;
```

2.2 Eliminar de una Base de datos

La sentencia que se utiliza para ello es **DROP DATABASE**.

Sintaxis:

```
DROP DATABASE [IF EXISTS] nombre_bd;
```

Ejemplo:

```
DROP DATABASE prueba;
/* La siguiente instrucción sólo elimina la base de datos si existe
previamente */
DROP DATABASE IF EXISTS prueba;
```

2.3 Listar bases de datos

Podemos ver la lista de esquemas (bases de datos) disponibles en nuestro servidor utilizando el comando **SHOW DATABASES**.
Sintaxis:

```
SHOW DATABASES;
```

2.4 Usar una base de datos

Ya que podemos disponer de diferentes BBDD dentro de un mismo servidor, es necesario indicarle al sistema gestor qué BD deseamos utilizar, bien sea para crear nuevas tablas, consultar datos o cualquier otra operación.

Para seleccionar un esquema se utiliza el comando **USE**.

Sintaxis:

```
USE nombre_bd;
```

Nota Importante

Recordad que, si usamos diferentes pestañas, solo podemos usar una BD. Si usamos otra en una pestaña, se usará también en todas las demás.

Por ello siempre tenemos que anteponer `USE nombre_bd;` a cualquier código que ejecutemos.

2.5 Ampliación: Consultar qué tabla estamos usando actualmente

2.6 Crear una tabla

¿Qué necesitamos para poder guardar los datos? Lo primero será definir los objetos donde vamos a agrupar esos datos. Los objetos básicos con los que trabaja SQL son las tablas, que como ya sabemos es un conjunto de filas y columnas cuya intersección se llama celda. Es ahí donde se almacenarán los elementos de información, los datos que queremos recoger.

Antes de crear la tabla es conveniente planificar algunos detalles:

- Qué nombre le vamos a dar a la tabla.
- **Qué nombre** le vamos a dar a cada una de las **columnas**.
- **Qué tipo y tamaño de datos** vamos a almacenar en cada columna.
- **Qué restricciones** tenemos sobre los datos.
- Alguna otra **información adicional** que necesitemos.

Y debemos tener en cuenta otras **reglas** que se deben cumplir **para los nombres de las tablas**:

- No podemos tener nombres de tablas duplicados en un mismo esquema (base de datos).
- Deben comenzar por un carácter alfabético.
- Su longitud máxima es de 30 caracteres.
- Solo se permiten letras del alfabeto inglés, dígitos o el signo de guión bajo.
- No puede coincidir con las palabras reservadas de SQL (por ejemplo, no podemos llamar a una tabla WHERE).
- No se distingue entre mayúsculas y minúsculas.

Para la creación de tablas con SQL se utiliza el comando **CREATE TABLE**. Este comando tiene una sintaxis más compleja de la que aquí se expone, pero vamos a comenzar por la sintaxis básica.

```
CREATE TABLE [IF NOT EXISTS] nombre_tabla (
    columna1 tipo_dato,
    columna2 tipo_dato,
    columna3 tipo_dato
);
```

Donde:

- columna1, columna2, ..., columnaN son los nombres de las columnas que contendrá la tabla.
- Tipo_Dato indica el tipo de dato de cada columna. Los diferentes tipos de datos se explican en el siguiente apartado.

Ejemplo. Queremos crear una tabla de personas con un único campo denominado nombre, de tipo texto y de máximo 25 caracteres:

```
CREATE TABLE personas (
    nombre VARCHAR(25)
);
```

2.7 Tipos de datos

Ya explicamos que al crear una tabla debemos resolver qué campos (columnas) tendrá y qué tipo de datos almacenará cada uno de ellos, es decir, su estructura.

Aquí recogeremos los tipos más utilizados, para ver una descripción completa de todos los tipos podéis dirigirlos [aquí](#).

Algunos de los tipos de datos más utilizados en el lenguaje MySQL son los siguientes:

2.7.1 Texto

VARCHAR(x): define una cadena de caracteres de longitud variable en la cual determinamos el máximo de caracteres con el argumento "x" que va entre paréntesis. Para almacenar cadenas de hasta 30 caracteres, definimos un campo de tipo VARCHAR(30). Si asignamos una cadena de caracteres de mayor longitud que la definida, recibiremos un error.

2.7.2 Número

- ✓ **INTEGER o INT**: número entero.

- ✓ **DECIMAL (m,d)**: número decimal donde 'm' es el número de dígitos almacenados total y 'd' el número de dígitos decimales.

DECIMAL(5,2) puede almacenar cualquier número de 5 dígitos y dos decimales, por lo que el rango de posibles valores iría de -999.99 a 999.99.

Todos los tipos numéricos pueden tener el atributo "**UNSIGNED**", esto permite sólo valores positivos.

Si necesitamos almacenar estaturas en cm, por ejemplo, nunca guardaremos valores negativos, entonces sería adecuado definir un campo "estatura" de tipo entero sin signo:

```
CREATE TABLE personas (
    nombre VARCHAR(25),
    estatura INTEGER UNSIGNED
);
```

Si necesitamos almacenar el precio de los libros, definimos un campo de tipo "DECIMAL(5,2) UNSIGNED" porque jamás guardaremos un valor negativo.

2.7.3 Temporal

Para guardar fechas y horas dispone de varios tipos:

- ✓ **DATE**: representa una fecha con formato "YYYY-MM-DD". Es decir: año, mes y día.
- ✓ **DATETIME**: almacena fecha y hora, su formato es "YYYY-MM-DD HH:MM:SS". Es decir: año, mes día, hora, minuto y segundos.
- ✓ **TIME**: una hora. Su formato es "HH:MM:SS". Es decir: hora, minutos y segundos.
- ✓ **YEAR**: un año. Su formato es "YYYY". Es decir, un año de 4 dígitos.

Es importante elegir el tipo de dato adecuado según el caso, el más preciso. Por ejemplo, si sólo necesitamos registrar un año (sin día ni mes), el tipo adecuado es "YEAR" y no "DATE". Esto sería útil para, por ejemplo, el año de fabricación de un modelo de coche que lo distinga de modelos más antiguos o nuevos.

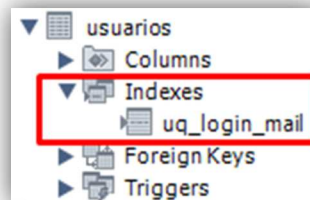
2.7.4 Nota: En "Nombres de las restricciones (cláusula CONSTRAINT)"

Existen una serie de restricciones entre las que se encuentran UNIQUE, FOREIGN KEY o CHECK que se les puede dar un nombre. Esto es especialmente útil si después nos interesa referirnos a ellas para eliminarlas o modificarlas.

Siempre que queramos poner un nombre a una restricción esta debe declararse obligatoriamente al final.

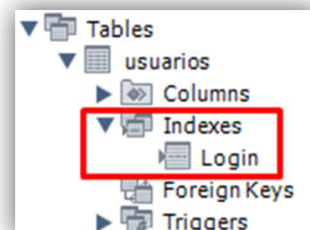
Si queremos darle un nombre a una restricción debemos usar la cláusula CONSTRAINT:

```
CREATE TABLE usuarios(
  ID INT PRIMARY KEY,
  Login VARCHAR(15) NOT NULL,
  Email VARCHAR(25) NOT NULL,
  CONSTRAINT uq_login_mail UNIQUE (Login, Email)
);
```



Ten en cuenta que, cuando no definimos un nombre para una restricción MySQL lo hace por nosotros. MySQL se asegurará de que cada restricción de un mismo tipo tenga un nombre diferente para una tabla.

Para consultar los nombres de una restricción podemos consultar el Navegador:



Dos restricciones no pueden tener el mismo nombre en una misma tabla. Por tanto, el siguiente código nos devolverá un error:

```
CREATE TABLE usuarios(
  ID INT PRIMARY KEY,
  Login VARCHAR(15) NOT NULL,
  Email VARCHAR(25) NOT NULL,
  CONSTRAINT restricción UNIQUE (Login),
  CONSTRAINT restricción UNIQUE (Email)
);
Error Code: 1061. Duplicate key name 'restricción'
```

Restricción DEFAULT" podréis leer sobre cómo asignar horas y fecha actuales automáticamente cada vez que se crea un registro en una tabla.

2.7.5 Lista de valores

- ✓ **ENUM:** El tipo de dato "ENUM" representa una lista de valores. Es un tipo de dato cadena de caracteres cuyo valor se elige de una lista enumerada de valores permitidos que se especifica al definir el campo. **Es decir, solo puede contener cadenas de caracteres.**

Por ejemplo:

Una empresa necesita personal y varias personas se han presentado para cubrir distintos cargos. La empresa almacena los datos de los candidatos a los puestos en una tabla llamada "candidatos". Le interesa, entre otras cosas, conocer los estudios que tiene cada persona, si tiene estudios ESO, Bachiller, FP, Universitario o ninguno. Para ello, crea un campo de tipo "ENUM" con esos valores.

```
CREATE TABLE personas (
    nombre VARCHAR(25),
    estudios ENUM('ESO', 'Bachiller', 'FP', 'Universitario',
                  'ninguno')
);
```

2.7.6 Si / No

BOOLEAN: valor verdadero o falso. Es decir: Si o No.

En realidad el tipo de datos booleano es un *TINYINT* (entero muy pequeño) que ocupa, por tanto, un byte.

Aunque en principio por coherencia semántica en este tipo de campo solo introduciremos 0 (falso) o 1 (verdadero), podríamos introducir un número de -127 a 127.

También acepta los valores TRUE y FALSE. Equivalen a 0 y 1, que serán los valores que se almacenarán.

Por ejemplo:

Una empresa guarda la información de sus empleados y quiere almacenar si necesitan plaza de parking o no. Para ello, crea un campo de tipo "BOOLEAN". Además establecemos FALSE (sin parking) como valor por defecto.

```
CREATE TABLE personas (
    nombre VARCHAR(25),
    parking BOOLEAN DEFAULT FALSE
);
```

2.7.7 Otros tipos

MySQL incluye muchos más tipos que pueden ser consultados en el enlace y en el siguiente resumen.

<https://www.anerbarrena.com/tipos-dato-mysql-5024/>

2.7.8 Ampliación de tipos de datos¹

Grupo	Tipo de dato	Intervalo	Almacenamiento
Booleano	BOOL (tinyint)	TRUE o FALSE	1 byte
Numéricos exactos	tinyint	De 0 a 255	1 byte
	smallint	De -2^{15} (-32.768) a $2^{15} - 1$ (32.767)	2 bytes
	mediumint	De -2^{24} (-8.388.608) a $2^{24} - 1$ (8.388.607)	3 bytes
	int	De -2^{31} (-2.147.483.648) a $2^{31} - 1$ (2.147.483.647)	4 bytes
	bigint	De -2^{63} (-9.223.372.036.854.775.808) a $2^{63} - 1$ (9.223.372.036.854.775.807)	8 bytes
	decimal, numeric, decimal (p, s)	<ul style="list-style-type: none"> p (precisión): el número total máximo de dígitos decimales que se puede almacenar, tanto a la izquierda como a la derecha del separador decimal. La precisión debe ser un valor comprendido entre 1 y la precisión máxima de 38. La precisión predeterminada es 18. s (escala): el número máximo de dígitos decimales que se puede almacenar a la derecha del separador decimal. La escala debe ser un valor comprendido entre 0 y p. Sólo es posible especificar la escala si se ha especificado la precisión. La escala predeterminada es 0. Con precisión máxima $1038 + 1$ y $1038 - 1$ 	Precisión 1 - 9: 5 bytes

¹ En negrita los más habituales

Grupo	Tipo de dato	Intervalo	Almacenamiento
Numéricos aproximados	float (n)	Por defecto: De - 1,79E+308 a -2,23E-308, 0 y de 2,23E-308 a 1,79E+308	Depende del valor de n
	real	Equivale a float(24): De - 3,40E + 38 a -1,18E - 38, 0 y de 1,18E - 38 a 3,40E + 38	4 Bytes
Fecha y hora	date	Del 1 de enero de 1753 hasta el 31 de diciembre de 9999	
	datetime	Del 1 de enero de 1753 00:00:00.000 hasta el 31 de diciembre de 9999 23:59:59.999	
Cadenas de caracteres Unicode	char (n)	Datos de carácter Unicode de longitud fija, con n caracteres. n debe estar comprendido entre 1 y 4.000	2 * n bytes
	varchar (n)	Datos de carácter Unicode de longitud variable. n indica que el tamaño máximo de almacenamiento es $2^{31} - 1$ bytes	2 * n bytes + 2 bytes

2.8 Restricciones básicas

Hay veces que necesitamos que un dato se incluya en una tabla de manera obligatoria, otras veces necesitaremos definir uno de los campos como llave primaria o ajena. Todo esto podremos hacerlo cuando definamos la tabla, además de otras opciones.

Una **restricción** es una condición que una o varias columnas deben cumplir obligatoriamente.

Cada restricción que creemos llevará un nombre, si no se lo ponemos nosotros lo hará MySQL o el SGBD que estemos utilizando.

Es conveniente que le pongamos un nombre que nos ayude a identificarla y que sea único para cada esquema (BD).

Es buena idea incluir de algún modo el nombre de la tabla, los campos involucrados y el tipo de restricción en el nombre de la misma. La sintaxis en SQL estándar es la siguiente:

```
CREATE TABLE [IF NOT EXISTS] nombre_tabla (
  columna1 tipo_dato [NOT NULL] [DEFAULT valor],
  columna2 tipo_dato [NOT NULL] [DEFAULT valor],
  columna3 tipo_dato [NOT NULL] [DEFAULT valor],
  PRIMARY KEY (columnaW, columnaX),
  [UNIQUE (columnaY, columnaZ)]
);
```

Veamos un ejemplo:

```
CREATE TABLE usuarios(
    Login VARCHAR(15),
    Password VARCHAR(8) NOT NULL,
    Fecha_Ingreso DATETIME DEFAULT (CURTIME()),
    PRIMARY KEY (Login)
);
```

Como puedes observar, algunas restricciones se incluyen en la definición de cada columna y otras restricciones, según el caso, se especifican al final.

Nota: La función CURTIME() nos proporciona la fecha y hora actuales. Tranquilos, también veremos qué es exactamente una función.

En los siguientes apartados veremos cada una de las restricciones, su significado y su uso.

2.8.1 Restricción NOT NULL

Con esta restricción obligaremos a que esa columna tenga un valor o lo que es lo mismo, prohíbe los valores nulos para una columna en una determinada tabla.

Podremos ponerlo cuando creamos o modificamos el campo añadiendo la palabra **NOT NULL** después de poner el tipo de dato.

Si en la tabla USUARIOS queremos que el campo "F_Nacimiento" sea obligatorio ponerlo, nos quedaría así:

```
CREATE TABLE usuarios(
    F_Nacimiento DATE NOT NULL
);
```

Esta cláusula se incluye en la misma línea en la que se declara el campo.

2.8.2 Restricción PRIMARY KEY

En el modelo relacional las tablas deben tener una **clave primaria**. Es evidente que cuando creamos la tabla tendremos que indicar a quién corresponde.

Sólo puede haber una clave primaria por tabla pero ésta puede estar formada por varios campos.

La clave primaria hace que los campos que la forman sean **NOT NULL** y que los valores de los campos sean de tipo **UNIQUE**.

Veamos cómo quedaría la tabla de usuarios si la clave fuese el campo Login:

- Si la clave la forma un único campo puede declararse en la propia línea o al final:

```
CREATE TABLE usuarios(
    Login VARCHAR(15) PRIMARY KEY,
    Email VARCHAR(25),
);
```

```
CREATE TABLE usuarios(
    Login VARCHAR(15),
    Email VARCHAR(25),
    PRIMARY KEY (Login)
);
```

- Si la clave está formada por más de un campo, por ejemplo Nombre y Apellidos, debe declararse obligatoriamente al final:

```
CREATE TABLE usuarios(
    Nombre VARCHAR(15),
    Apellidos VARCHAR(25),
    F_nacimiento DATE,
    PRIMARY KEY (Nombre, Apellidos)
);
```

2.8.3 Restricción UNIQUE

Habrán ocasiones en la que nos interese que no se puedan repetir valores en la columna, en estos casos utilizaremos la restricción **UNIQUE**. MySQL crea un índice automáticamente cuando se habilita esta restricción y lo borra al deshabilitarla.

Esta restricción, junto con **NOT NULL**, sería la implementación física de una **clave alternativa**.

Supongamos que el campo Login de nuestra tabla va a ser único. Lo incluiremos en la tabla que estamos creando.

Nos quedaría así:

```
CREATE TABLE usuarios(
    Login VARCHAR(15) NOT NULL,
    Pass varchar(15) NOT NULL,
    UNIQUE (Login)
);
```

```
CREATE TABLE usuarios(
    Login VARCHAR(15) UNIQUE NOT NULL,
    Pass varchar(15) NOT NULL,
);
```

Hay que recordar que las claves alternativas en las tablas relacionales deben llevar restricciones UNIQUE y NOT NULL.

Esta restricción puede aplicarse conjuntamente sobre más de un campo. **En el caso de una clave única compuesta, al igual que en el caso de la clave primaria, debe declararse obligatoriamente al final.**

Por ejemplo, si queremos que la unión del Login y el correo electrónico sean únicos debemos ponerlo así:

```
CREATE TABLE usuarios(
  Login VARCHAR(15) NOT NULL,
  Email VARCHAR(25) NOT NULL,
  Pass varchar(15) NOT NULL,
  UNIQUE (Login, Email)
);
```

Si te fijas, detrás del tipo de datos del campo Email hay una coma, eso es así porque la restricción es independiente de ese campo y común a varios. Por eso después de **UNIQUE** hemos puesto entre paréntesis los nombres de los campos a los que afecta la restricción.

Aunque en una tabla solo puede existir una clave primaria, sí pueden existir varios **índices únicos** o claves alternativas (**índices únicos no nulos**). Esto provoca que sea adecuado que asignemos un identificador (nombre) a la restricción UNIQUE. Esto nos permitirá poder acceder a él para, por ejemplo, eliminarlo.

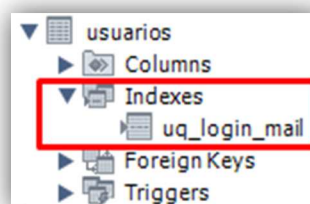
2.8.4 Nombres de las restricciones (cláusula CONSTRAINT)

Existen una serie de restricciones entre las que se encuentran UNIQUE, FOREIGN KEY o CHECK que se les puede dar un nombre. Esto es especialmente útil si después nos interesa referirnos a ellas para eliminarlas o modificarlas.

Siempre que queramos poner un nombre a una restricción esta debe declararse obligatoriamente al final.

Si queremos darle un nombre a una restricción debemos usar la cláusula CONSTRAINT:

```
CREATE TABLE usuarios(
  ID INT PRIMARY KEY,
  Login VARCHAR(15) NOT NULL,
  Email VARCHAR(25) NOT NULL,
  CONSTRAINT uq_login_mail UNIQUE (Login, Email)
);
```



Ten en cuenta que, cuando no definimos un nombre para una restricción MySQL lo hace por nosotros. MySQL se asegurará de que cada restricción de un mismo tipo tenga un nombre diferente para una tabla.

Para consultar los nombres de una restricción podemos consultar el Navegador:



Dos restricciones no pueden tener el mismo nombre en una misma tabla. Por tanto, el siguiente código nos devolverá un error:

```
CREATE TABLE usuarios(
  ID INT PRIMARY KEY,
  Login VARCHAR(15) NOT NULL,
  Email VARCHAR(25) NOT NULL,
  CONSTRAINT restricción UNIQUE (Login),
  CONSTRAINT restricción UNIQUE (Email)
);
Error Code: 1061. Duplicate key name 'restricción'
```

2.8.5 Restricción DEFAULT

A veces es muy tedioso insertar siempre lo mismo en un campo. Imagínate que casi todos los jugadores fuesen de España y tenemos un campo País. ¿No sería cómodo asignarle un valor por defecto? Eso es lo que hace la restricción **DEFAULT**.

En nuestro ejemplo vamos a añadir a la tabla USUARIOS el campo País y le daremos por defecto el valor "España".

```
CREATE TABLE usuarios(
  Login VARCHAR(15) PRIMARY KEY,
  Email VARCHAR(25),
  Pais VARCHAR(25) DEFAULT 'España',
);
```

En las especificaciones de **DEFAULT** vamos a poder añadir distintas expresiones: constantes, funciones SQL y variables.

Si queremos incluir en un campo la fecha actual, independientemente del día en el que estemos, podremos utilizar la función **CURTIME()** como valor por defecto:

```
CREATE TABLE usuarios(
  Login VARCHAR(15) PRIMARY KEY,
  Email VARCHAR(25),
  Pais VARCHAR(25) DEFAULT 'España',
  Fecha_ingreso DATE DEFAULT (CURTIME())
);
```

2.9 Eliminar una tabla

Cuando una tabla ya no es útil y no la necesitamos es mejor borrarla, de este modo no ocupará espacio y podremos utilizar su nombre en otra ocasión.

La sentencia en SQL para eliminar tablas es **DROP TABLE**. Su sintaxis es:

```
DROP TABLE [IF EXISTS] nombre_tabla;
```

El borrado de una tabla es irreversible y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación. Al borrar una tabla se borran todos los datos que contiene.

Ejemplos:

```
DROP TABLE COCHES;
```

2.10 Modificar una tabla

Es posible que después de crear una tabla nos demos cuenta que se nos ha olvidado añadir algún campo o restricción, quizás alguna de las restricciones que añadimos ya no es necesaria o tal vez queramos cambiar el nombre de alguno de los campos. ¿Es posible esto? Ahora veremos que sí y en casi todos los casos utilizaremos el comando **ALTER TABLE**.

2.10.1 Cambiar de nombre una tabla

La sintaxis para cambiar el nombre de una tabla es la siguiente:

```
RENAME TABLE nombre_actual TO nombre_nuevo;
```

Ejemplo:

```
RENAME TABLE COCHES TO AUTOMOVILES;
```

La tabla COCHES a partir de ese momento se llamará AUTOMOVILES.

2.10.2 Cambiar de nombre una columna

La sintaxis para cambiar el nombre de una columna es la siguiente:

```
ALTER TABLE tabla RENAME COLUMN nombre_actual TO nombre_nuevo;
```

Ejemplo:

```
ALTER TABLE clientes RENAME COLUMN Login TO Usuario;
```

La columna "Login" a partir de ese momento se llamará "Usuario".

2.10.3 Modificar una columna

Podemos modificar el tipo de datos y las propiedades de una columna. Todos los cambios son posibles si la tabla no contiene datos. En general, si la tabla no está vacía podremos aumentar la longitud de una columna, aumentar o disminuir en número de posiciones decimales en un tipo DECIMAL o reducir la anchura siempre que los datos no ocupen todo el espacio reservado para ellos. También permite añadir o quitar restricciones de columna como NOT NULL o DEFAULT.

La sintaxis para realizar modificaciones en una columna es la siguiente:

```
ALTER TABLE nombre_tabla MODIFY COLUMN
nombre_columna nuevo_tipo_dato [nuevas restricciones];
```

Ejemplo:

```
ALTER TABLE AUTOMOVILES MODIFY COLUMN color VARCHAR(20) NOT NULL;
```

La columna "color" a partir de este momento será un texto de máximo 20 caracteres y obligatorio.

2.10.4 Añadir una columna

Las nuevas columnas que se añadan a una tabla se incluirán al final de la tabla.

La sintaxis para añadir una nueva columna a una tabla es la siguiente:

```
ALTER TABLE nombre_tabla
ADD [COLUMN] nombre_columna tipo_dato [restricciones];
```

Ejemplo:

```
ALTER TABLE VEHICULOS ADD COLUMN fechaMatric DATE NOT NULL;
```

La columna "fechaMatric" a partir de ahora existe en la tabla VEHÍCULOS.

2.10.5 Eliminar una columna

Esta acción no se puede deshacer. Además de la definición de la columna, se eliminarán todos los datos que contuviera. No se puede eliminar una columna si es la única que forma la tabla, para ello tendremos que borrar la tabla directamente.

La sintaxis para eliminar una nueva columna de una tabla es la siguiente:

```
ALTER TABLE nombre_tabla DROP [COLUMN] nombre_columna;
```

Ejemplo:

```
ALTER TABLE VEHICULOS DROP COLUMN fechaMatric;
```

La columna "fechaMatric" a partir de ahora ya no existirá en la tabla VEHICULOS y los datos que contuviera se han eliminado de manera irreversible.

2.10.6 Eliminar clave primaria

La sintaxis para eliminar la clave primaria de una tabla es la siguiente:

```
ALTER TABLE nombre_tabla DROP PRIMARY KEY;
```

Recuerda que una tabla SIEMPRE debe tener una clave primaria.

2.10.7 Crear clave primaria

Sabemos que una clave primaria es una condición de obligado cumplimiento para una o más columnas de la tabla. Hemos visto que se pueden añadir al crear la tabla, o bien, podemos hacerlo mediante modificación posterior de la tabla.

La sintaxis para crear una clave primaria es:

```
ALTER TABLE nombre_tabla ADD PRIMARY KEY (columnas);
```

No se puede crear una clave primaria en una tabla que ya posee una. Es necesario eliminar primero la existente antes de crear una nueva.

2.10.8 Crear clave alternativa (UNIQUE)

La sintaxis para crear una clave alternativa o restricción UNIQUE puede hacerse:

Con ALTER TABLE

```
ALTER TABLE nombre_tabla ADD UNIQUE (columnas);  
O bien  
ALTER TABLE nombre_tabla ADD CONSTRAINT nomb_rest UNIQUE (columnas);
```

Con CREATE INDEX

```
CREATE UNIQUE INDEX nomb_rest ON nombre_tabla (columnas);
```

Recuerda que para crear una clave alternativa, además de establecer una restricción o índice único, el campo o combinación de campos que participan en la restricción **deben a su vez poseer la restricción NOT NULL**.

2.10.9 Eliminar clave alternativa (UNIQUE)

Previamente a la eliminación debemos conocer el nombre que se le ha otorgado a la restricción, la hayamos nombrado nosotros o bien lo haya nombrado MySQL.

La sintaxis para eliminar la clave alternativa de una tabla es la siguiente:

Con ALTER TABLE

```
ALTER TABLE nombre_tabla DROP INDEX nombre_índice_único;
```

Con DROP INDEX

```
DROP INDEX nombre_índice_único ON nombre_tabla;
```

2.11 Listar tablas

Podemos ver la lista de tablas de una base de datos utilizando el comando **SHOW TABLES**.

Sintaxis:

```
SHOW TABLES;
```

2.12 Describir una tabla

El comando **DESCRIBE**, permite obtener la estructura de una tabla.

Ejemplo:

```
DESCRIBE COCHES;
```

Y aparecerán los campos de la tabla COCHES junto su información.

2.13 Acceder al código SQL de creación de una tabla

El comando SHOW CREATE TABLE permite obtener el código SQL que dio lugar a la creación de una tabla. Ojo, el código no tiene porqué ser exactamente igual que el que introdujimos nosotros a la hora de crearla. Pero debe ser equivalente.

```
SHOW CREATE TABLE COCHES;
```

Y aparecerá el código SQL que dio lugar a la creación de la tabla COCHES.

3 Lenguaje de definición de datos (DDL) II

3.1 Restricción FOREIGN KEY

3.1.1 Concepto

Ya hemos visto durante la fase de diseño lógico que las claves ajenas, secundarias o foráneas son campos de una tabla que se relacionaban con la clave primaria (o incluso con la clave candidata) de otra tabla.

Cuando creamos la tabla tendremos que indicar de alguna forma qué campos son clave ajena. Lo haremos "haciendo referencia" a la tabla y los campos de donde proceden.

Es importante resaltar que una clave ajena debe referenciar a una **clave primaria/candidata completa** de la tabla padre, y nunca a un subconjunto de los atributos que forman esta clave primaria.

3.1.2 Definir una clave ajena al crear una tabla

Vamos a tomar como ejemplo nuestra tabla de usuarios y vamos a crear una tabla de PARTIDAS de un juego. Cada partida estará asociada a un solo usuario, por lo que la tabla de partidas tendrá una clave ajena que guarde el Login del usuario.

```
CREATE TABLE usuarios (
    Login VARCHAR(15),
    Email VARCHAR(25),
    Pais VARCHAR(25) DEFAULT 'España',
    PRIMARY KEY (Login)
);

CREATE TABLE partidas (
    Id_partida INTEGER PRIMARY KEY,
    Login_usuario VARCHAR(15),
    FOREIGN KEY(Login_usuario) REFERENCES usuarios(Login)
);
```


La definición de la clave ajena se pone al final, después de definir la clave primaria y se utiliza el texto **FOREIGN KEY** con la clave ajena entre paréntesis seguido por la palabra REFERENCES y la tabla origen con el campo de origen entre paréntesis.

Es muy importante que al definir un campo que va a ser clave ajena, nos fijemos en que tenga el mismo tipo de dato y longitud que el campo de origen al que va a referenciar.

De igual manera que para la restricción UNIQUE si no especificados un nombre, MySQL establecerá un nombre por nosotros. Si queremos otorgarle un nombre específico usaremos:

```
CREATE TABLE partidas (
    Id_partida INTEGER,
    Login_usuario VARCHAR(15),
    PRIMARY KEY (Id_partida),
    CONSTRAINT fk_partidas_usuarios FOREIGN KEY(Login_usuario)
    REFERENCES usuarios(Login)
);
```

3.2 Integridad referencial



ID	CustomerName	CustomerAge	CustomerCountry
1	Salvador	23	Brazil
2	Lawrence	60	China
3	Ernest	38	India

ID	OrderDate	CustomerID	Amount
1	2019-04-29 00:00:00.000	1	968
2	2019-05-10 00:00:00.000	2	898
3	2019-10-21 00:00:00.000	3	47

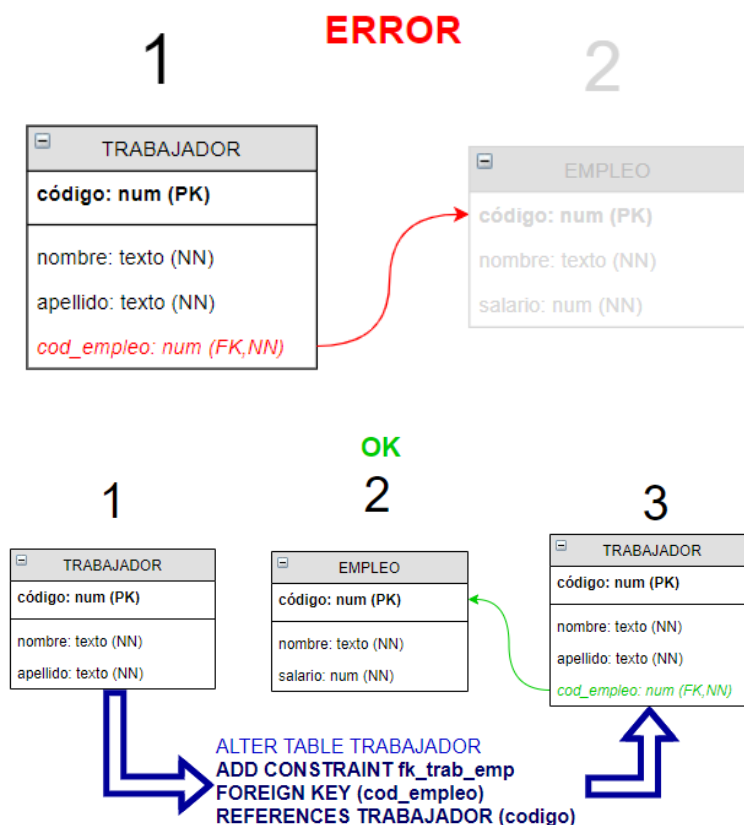
Al relacionar campos necesitamos que el dato del campo que es clave ajena en una tabla (que llamaremos secundaria) previamente haya sido incluido en su tabla de procedencia donde es clave primaria o candidata. En nuestro ejemplo, cualquier login de usuario que incluyamos en la tabla PARTIDAS, debería estar previamente en la tabla de la que procede, es decir, en la tabla USUARIOS. **A esto se le llama Integridad Referencial.**

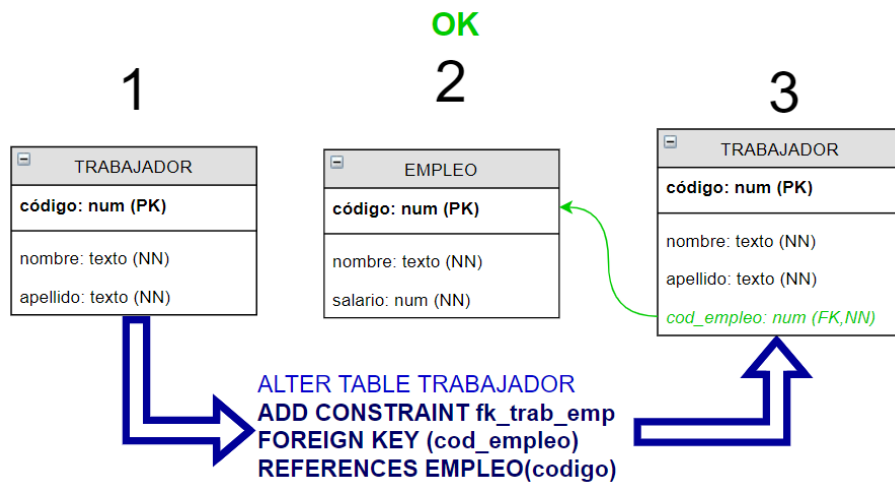
Esto puede crear algunos problemas o errores al realizar las siguientes operaciones:

3.2.1 Creación de una tabla (CREATE TABLE) (Ver diapositivas)

Al crear una tabla: Si intentamos crear una tabla que hace referencia a una tabla que aún no está creada, MySQL buscará la tabla referenciada y al no encontrarla dará error.

- ✓ Esto se soluciona creando en primer lugar las tablas que no tengan claves ajenas.



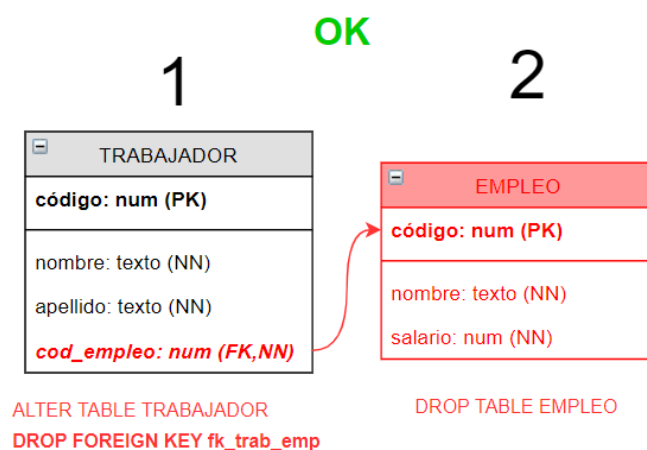
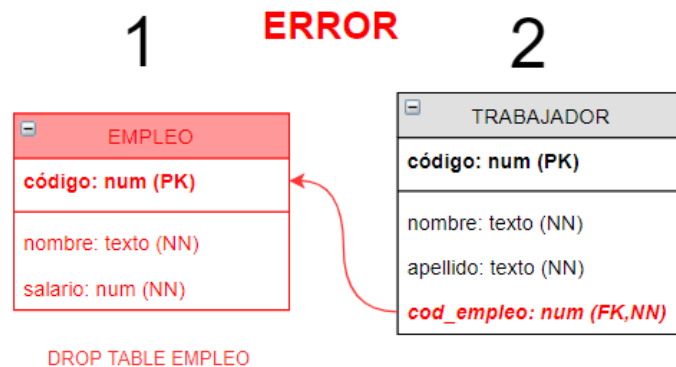


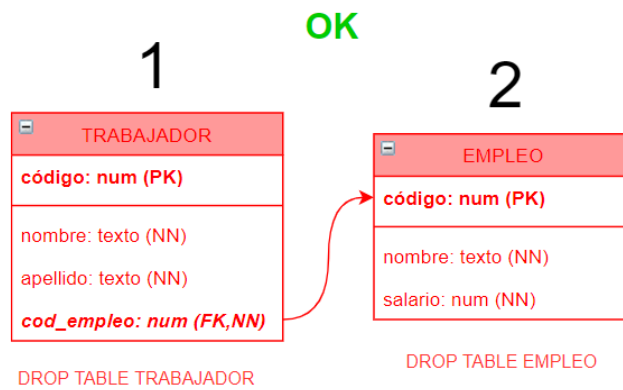
Siempre debemos CREAR primero la tabla padre o referenciada (la que contiene la clave primaria) antes que la restricción de clave ajena en la tabla hija (tabla que referencia).

3.2.2 Borrado de una tabla (DROP TABLE) (Ver diapositivas)

Al borrar una tabla: Si queremos borrar las tablas tendremos que proceder al contrario:

- ✓ Borraremos las restricciones de clave ajena o las propias tablas que apunten a la tabla que queremos borrar.



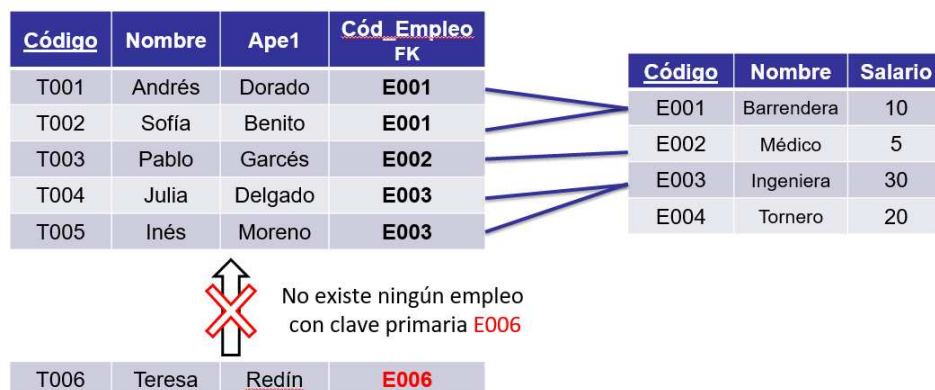


Siempre debemos BORRAR primero la tabla que referencia/hija o su restricción de clave ajena antes que borrar la tabla referenciada/padre.

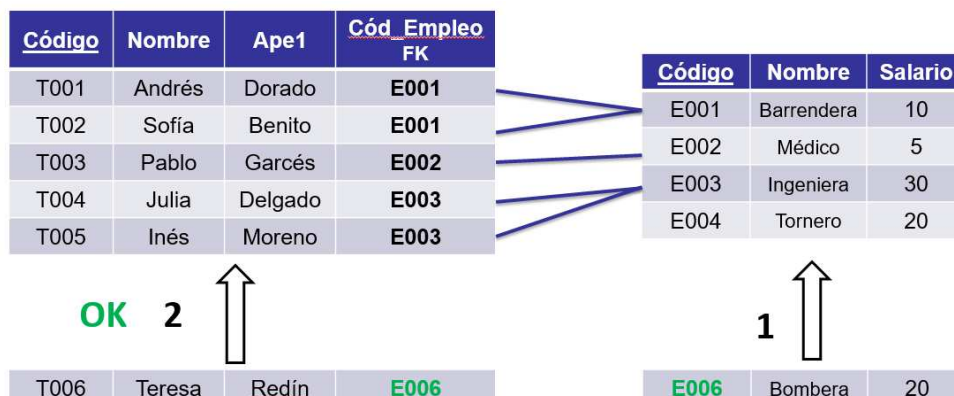
3.2.3 Inserción de un registro (INSERT) (Ver diapositivas)

Al insertar un registro: Si intentamos insertar un registro con datos en un campo que es clave ajena, y dichos datos no existen en la tabla de referencia, dará error. Esto se soluciona insertando primero los datos de las tablas de origen.

ERROR



OK



3.2.4 Actualización y borrado de un registro (UPDATE y DELETE) (Ver

diapositivas)

Al actualizar/borrar un registro: Si actualizamos la clave primaria de un registro y está referenciado en otra tabla, o si intentamos borrar un registro cuya clave primaria está referenciada en otra tabla, MySQL dará un error, ya que detecta que hay valores en la segunda tabla que ya no van a tener sentido. Esto se soluciona cambiando las acciones por defecto cuando se define la clave ajena.

Borrado de registro referenciado

Ejemplo: Quiero borrar el empleo "Médico"

ERROR: Pablo Garcés referencia a E002

<u>Código</u>	Nombre	Ape1	<u>Cód_Empleo</u> FK
T001	Andrés	Dorado	E001
T002	Sofía	Benito	E001
T003	Pablo	Garcés	E002
T004	Julia	Delgado	E003
T005	Inés	Moreno	E003

<u>Código</u>	Nombre	Salario
E001	Barrendera	10
E002	Médico	5
E003	Ingeniera	30
E004	Tornero	20

¿Qué opciones tenemos disponibles si queremos eliminar una profesión que es referenciada por un trabajador (o más)?

1. Podemos denegar el borrado del registro (genera error).
2. Podemos borrar a "Pablo Garcés" (y a otros que apuntaran a médico) antes de borrar el empleo "Médico".
3. Podemos hacer que Pablo Garcés (y otros) no apunte a ningún Empleo (valor NULL en su campo de clave ajena). Para eso la FK tendría que admitir nulos.

Actualización de registro referenciado

Ejemplo: Quiero modificar la clave primaria del empleo "Médico"

ERROR: Pablo Garcés referencia a una PK que ha sido modificada.

<u>Código</u>	Nombre	Ape1	<u>Cód_Empleo</u> FK
T001	Andrés	Dorado	E001
T002	Sofía	Benito	E001
T003	Pablo	Garcés	E002
T004	Julia	Delgado	E003
T005	Inés	Moreno	E003

<u>Código</u>	Nombre	Salario
E001	Barrendera	10
E555	Médico	5
E003	Ingeniera	30
E004	Tornero	20

¿Qué opciones tenemos disponibles si queremos modificar la clave primaria de una profesión que es referenciada por un trabajador (o más)?

1. Podemos denegar la actualización de la PK del registro. (generar error).
2. Podemos actualizar el campo FK de "Pablo Garcés" (y de otros que apuntaran a médico) para que sea coherente con la nueva PK de "Médico".

3. Podemos hacer que Pablo Garcés (y otros) no apunte a ningún Empleo (valor NULL en si campo de clave ajena). Para eso la FK tendría que admitir nulos.

3.2.5 Acciones ante una infracción de la integridad (UPDATE/DELETE)

- ✓ **ON DELETE NO ACTION: Opción por defecto.** El Motor de base de datos genera un error y se revierte la acción de eliminación de la fila de la tabla principal.
- ✓ **ON DELETE CASCADE:** Si se elimina una fila de la tabla principal, se eliminan todas las filas coincidentes en la tabla secundaria.
- ✓ **ON DELETE SET NULL:** Si se elimina una fila de la tabla principal, todas las columnas de referencia en todas las filas coincidentes de la tabla secundaria se establecen en NULL.
- ✓ **ON UPDATE NO ACTION: Opción por defecto.** El Motor de base de datos genera un error y se revierte la acción de modificación de la fila de la tabla principal.
- ✓ **ON UPDATE CASCADE:** Cualquier cambio en una columna referenciada en la tabla primaria provoca el mismo cambio en la columna de referencia correspondiente en las filas coincidentes de la tabla secundaria.
- ✓ **ON UPDATE SET NULL:** Cualquier cambio en una columna referenciada en la tabla primaria provoca que la columna de referencia correspondiente en las filas coincidentes de la tabla secundaria se establezca como nula.

En nuestro ejemplo de la tabla PARTIDAS, nos interesa que al actualizar el Login de un usuario, se actualice en todas las partidas (ON UPDATE CASCADE), mientras que si borramos un usuario queremos que se borren todas sus partidas:

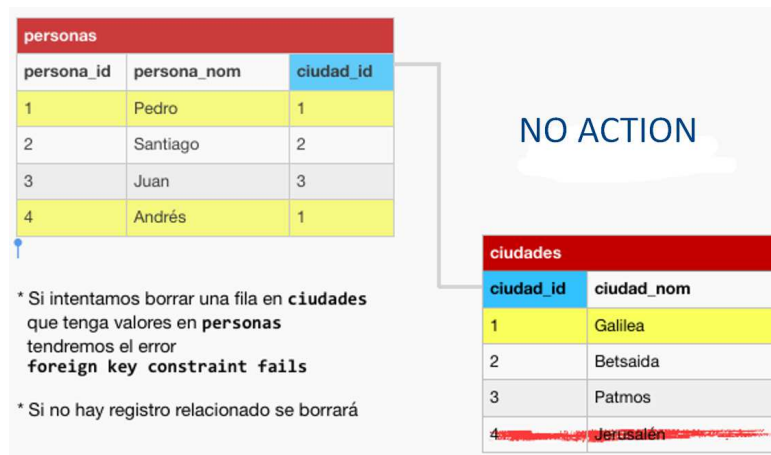
```
CREATE TABLE partidas (
  Id_partida INTEGER,
  Login_usuario VARCHAR(15),
  PRIMARY KEY (Id_partida),
  FOREIGN KEY(Login_usuario) REFERENCES usuarios(Login)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

Ejemplos

- ✓ BD de ejemplo:



- ✓ **NO ACTION:** No se permite la actualización o borrado (opción por defecto).
En el ejemplo se permitiría el borrado de Jerusalén porque no infringe la restricción de clave ajena al no existir ninguna persona de Jerusalén.
Sin embargo, **no se permitiría el borrado de Galilea.**



- ✓ **CASCADE:** Se borra/actualiza el registro relacionado.
En el ejemplo, al borrar Galilea se borrarían también todas las personas cuya ciudad es Galilea.



- ✓ **SET NULL:** Se inserta un valor nulo en el campo de clave ajena del registro relacionado (siempre que permita nulos).
En el ejemplo, al borrar Galilea se establecerá el valor nulo en el campo de clave ajena de todas las personas cuya ciudad es Galilea.



3.2.6 Modificar una tabla para incluir una clave ajena

Sabemos que una clave ajena es una condición de obligado cumplimiento para una o más columnas de la tabla. Hemos visto que se pueden añadir al crear la tabla, o bien, podemos hacerlo mediante modificación posterior de la tabla.

La sintaxis para crear una clave ajena es:

```
ALTER TABLE nombre_tabla ADD FOREIGN KEY (columna_origen)
REFERENCES nombre_tabla_destino (columna_destino) opciones_integridad;
```

Si queremos incluir un nombre para la clave ajena:

```
ALTER TABLE nombre_tabla ADD CONSTRAINT nombre_fk FOREIGN KEY (columna_origen)
REFERENCES nombre_tabla_destino (columna_destino) opciones_integridad;
```

3.2.7 Modificar una tabla para eliminar clave ajena

La sintaxis para eliminar una clave ajena de una tabla es la siguiente:

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

4 Lenguaje de definición de datos (DDL) III

4.1 Restricción AUTO_INCREMENT

Un campo de tipo entero puede tener otro atributo extra 'AUTO_INCREMENT'. Los valores de un campo 'AUTO_INCREMENT', se inician en 1 y se incrementan en 1 automáticamente.

Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.

Sólo puede haber un campo "AUTO_INCREMENT" y debe ser clave primaria.

Para establecer que un campo autoincrementa sus valores automáticamente, éste debe ser entero (integer) y debe ser clave primaria:

```
CREATE TABLE libros (
  codigo INTEGER PRIMARY KEY AUTO_INCREMENT,
  titulo VARCHAR(25),
  autor VARCHAR(25),
  editorial VARCHAR(25)
);
```

Para definir un campo autoincrementable colocamos "AUTO_INCREMENT" después de la definición del campo al crear la tabla.

Hasta ahora, al introducir registros, colocamos el nombre de todos los campos antes de los valores.

Cuando un campo tiene el atributo "AUTO_INCREMENT" no es necesario introducir valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para introducir registros omitimos el campo definido como "AUTO_INCREMENT", por ejemplo:

```
INSERT INTO libros (titulo,autor,editorial)
VALUES ('El aleph','Borges','Planeta');
```

Este primer registro introducido guardará el valor 1 en el campo correspondiente al código.

Si continuamos introduciendo registros, el código (dato que no introducimos) se cargará automáticamente siguiendo la secuencia de autoincremento.

4.2 Restricción CHECK

Un check es una restricción, o sea una limitación que deben cumplir los datos para que sean considerados válidos y puedan ser almacenados.

Sintaxis:

```
[CONSTRAINT [nombre_restriccion]] CHECK (condicion)
```

La expresión (condición) devolverá los siguientes valores:

- ✓ **TRUE:** La condición se cumple y por tanto se autorizará la inserción/modificación (no habrá error).
- ✓ **UNKNOWN:** el resultado es desconocido al encontrarse un valor nulo (NULL) en la expresión. Se autorizará la inserción/modificación (no habrá error).
- ✓ **FALSE:** La condición no se cumple, la inserción/modificación generará un error.

Se pueden definir restricciones que afecten a los valores de una sola columna a los valores de varias.

Por ejemplo, si los precios no pueden ser negativos podríamos tener un check que evite guardar precios negativos. Si las notas de los alumnos deben estar entre 0 y 100 podemos tener un check que evite ingresar notas fuera de ese rango. Si las fechas no pueden ser anteriores al día 1 de enero de 2013 podemos tener un check que evite ingresar fechas anteriores.

Ejemplo con definición en la propia declaración de la columna:

```
CREATE TABLE libros (
    codigo INTEGER PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(25) NOT NULL,
    primera_edicion YEAR NOT NULL,
    ultima_edicion YEAR NOT NULL,
    num_pag INTEGER NOT NULL CHECK (num_pag > 0 AND num_pag > 5000)
);
```

Ejemplo con definición al final de la declaración de la tabla:

```
CREATE TABLE libros (
    codigo INTEGER PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(25) NOT NULL,
    primera_edicion YEAR NOT NULL,
    ultima_edicion YEAR NOT NULL,
    num_pag INTEGER NOT NULL
    CONSTRAINT ck_num_pag CHECK (num_pag > 0 AND num_pag > 5000)
);
```

En caso de definir una restricción CHECK que incluya dos o más columnas deberá declararse al final de la declaración de la tabla (tal y como ocurre con las claves).

```
CREATE TABLE libros (
    codigo INTEGER PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(25) NOT NULL,
    primera_edicion YEAR NOT NULL,
    ultima_edicion YEAR NOT NULL,
    num_pag INTEGER NOT NULL CHECK (num_pag > 0 AND num_pag > 5000)
    CHECK (primera_edicion <= ultima_edicion)
);
```

En caso de querer añadir o eliminar una restricción CHECK lo haremos de manera similar a las restricciones de clave primaria, única o ajena.

Ejemplos de inclusión de una restricción CHECK:

```
ALTER TABLE Exámenes  
ADD CONSTRAINT ck_exámenes CHECK (NOTA >= 0 AND NOTA <= 100);
```

```
ALTER TABLE Usuarios  
ADD CONSTRAINT ck_fecha_alta CHECK (Fecha_Alta >= Fecha_Nacimiento);
```

Ejemplo de eliminación de una restricción CHECK:

```
ALTER TABLE Usuarios  
DROP CHECK ck_fecha_alta;
```