
UD4: MODELO FÍSICO DDL

Tema Extendido

Bases de Datos (BD)
CFGS DAM/DAW

Abelardo Martínez y Pau Miñana.
Basado y modificado de Sergio Badal y Raquel Torres.
Curso 2023-2024

ÍNDICE

- [1. Introducción](#)
- [2. Instalación de MySQL](#)
- [3. Creación de una base de datos](#)
- [4. Creación de tablas](#)
 - [4.1. Nombres en las tablas](#)
 - [4.2. Tipos de datos](#)
 - [4.3. CREATE TABLE](#)
 - [4.4. Scripts](#)
 - [4.5. Restricciones de tabla](#)
 - [4.5.1. NOT NULL](#)
 - [4.5.2. UNIQUE](#)
 - [4.5.3. PRIMARY KEY](#)
 - [4.5.3. FOREIGN KEY](#)
 - [4.5.4. CHECK](#)
- [5. Modificación de tablas](#)
 - [5.1. Cambiar de nombre una tabla](#)
 - [5.2. Cambiar de nombre una columna](#)
 - [5.3. Añadir o quitar restricciones](#)
 - [5.4. Añadir o quitar columnas](#)
 - [5.5. Eliminar PRIMARY KEY](#)
 - [5.6. Modificar columnas](#)
 - [5.7. Modificar restricciones](#)
 - [5.8. Desactivar restricciones](#)
 - [5.9. Ejemplo completo de modificaciones](#)
- [6. Borrado de Tablas](#)
 - [6.1. Borrar todo el contenido de una tabla](#)
 - [6.2. Borrar una tabla por completo](#)
- [7. Bibliografía](#)

1. Introducción

Ha llegado el momento de implementar físicamente en un SGBD las tablas que hemos desarrollado en el diseño lógico de la base de datos. Para ello usaremos **MySQL**, considerado el Sistema Gestor de Bases de Datos Relacional (SGBDR o RDBMS en inglés) de código abierto más popular del mundo y ampliamente usado en entornos de desarrollo web. De todos modos todos los SGBDRs usan el mismo lenguaje común, así que muchas de las sentencias usadas serán compatibles con todos ellos, aunque cada uno tiene comandos de gestión interna distintos, algunos tipos de datos propios y ciertas personalizaciones del lenguaje que se salen del estándar y no son compatibles con los demás.

SQL (*Structured Query Language*) es ese lenguaje específico diseñado para administrar y recuperar información de los SGBDR. Aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

Las diferentes operaciones que se pueden realizar se clasifican atendiendo a si esas operaciones se refieren a los datos o a los metadatos, entendidos estos últimos como las "estructuras" en las que se almacenan o clasifican los datos.

- Operaciones sobre los **metadatos**:
 - **DDL** (*Data Definition Language*) se encarga de la **creación, modificación y eliminación** de los **objetos de la BD**, entre los que se encuentran las tablas y los campos, principalmente. Es el lenguaje con el que accedemos a la base de datos que realiza la función de definición de datos de cualquier SGBD.
- Operaciones sobre los **datos**:
 - **DML** (*Data Manipulation Language*) se encarga de la **creación, modificación y eliminación** de los **datos de la BD**.
 - **DQL** (*Data Query Language*) se encarga de la **consulta o lectura** de los **datos de la BD**.

En este tema se estudia el lenguaje DDL y en los siguientes el DML y el DQL respectivamente.



Las instrucciones DDL generan acciones que no se pueden deshacer (salvo que dispongamos de alguna copia de seguridad).

2. Instalación de MySQL

MySQL permite crear libremente diferentes bases de datos en un mismo servidor (también se pueden denominar Schemas) y otorgar los permisos deseados según usuarios o roles de usuario. No todos los SGBD funcionan así, en Oracle por ejemplo se crea una BD a partir de un esquema de usuario.

Hay decenas de tutoriales para instalar de forma sencilla MySQL en tu equipo, aunque algunos de ellos lo hacen dentro de una distribución XAMPP, junto con otros servicios web para el despliegue de la información y esto no es necesario para este módulo. En la propia web de MySQL se puede encontrar la guía de instalación, para Windows, macOS o Linux:

- <https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/>

Como complemento, un tutorial completo para instalar en Windows:

- <https://www.solvetic.com/tutoriales/article/10435-instalar-mysql-en-windows-10/>

Para Ubuntu y otros sistemas similares que usen apt simplemente abre un terminal y escribe:

```
sudo apt update
sudo apt install mysql-server
sudo systemctl start mysql.service
sudo mysql
```

Las 2 primeras líneas corresponden con la instalación, la tercera arranca el servicio, lo que en la mayoría de sistemas será automático después de la instalación en realidad. La última línea es la que se necesita usar para abrir el terminal de mysql cada vez que lo necesites. Muchos tutoriales que puedes encontrar recomiendan ejecutar el script "mysql_secure_installation" tras la instalación, pero puesto que de momento no vamos a poner el servicio a funcionar más que desde nuestra propia máquina, no es necesario. Con ello simplemente se puede arrancar mysql en la propia maquina usando permisos de superusuario.

En el desarrollo del curso no se usa ningún entorno gráfico, ya que se trabaja principalmente con sentencias SQL, pero en caso que queráis apoyaros en alguno podéis instalar el Workbench desde la propia web de MySQL. En el tutorial de windows ya está incluido en realidad, pero de nuevo, en la propia web del SGBD se encuentran los detalles para instalarlo en cualquier sistema operativo:

- <https://dev.mysql.com/doc/workbench/en/wb-installing.html>

Aunque en muchos ejemplos que podréis encontrar en internet (sobretudo para el desarrollo web) se usa *phpMyAdmin* como cliente para conectarse a un servidor de MySQL, se desaconseja su uso por ofrecer una conexión con el SGBD poco eficiente y enmascarar las respuestas del mismo, con lo que puede complicar la resolución de errores en las sentencias SQL en vez de ayudar.

Por otro lado, si se quiere probar el SGBD de Oracle rápidamente y sin instalación existe la posibilidad de crearse un usuario gratuito y usar **Oracle Live SQL** como entorno de pruebas. Se puede crear el usuario o acceder al sistema desde el siguiente link: <https://login.oracle.com/mysso/signon.jsp>

3. Creación de una base de datos

⚠ Recuerda que todas las **sentencias SQL** siempre terminan con **;** en caso contrario aunque pulses intro los SGBD simplemente muestran una nueva línea en pantalla para organizar la sentencia a nivel visual en varias líneas, pero no afecta a su funcionamiento.

Del mismo modo, aunque las sentencias SQL son indiferentes a las mayúsculas, se usan para distinguir fácilmente las palabras reservadas de los nombres usados en la BD (campos o tablas).

Las instrucciones de creación de base de datos suelen contener elementos propios de cada SGBD, en MySQL se usa la siguiente nomenclatura:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_bd
[especificación_create [especificación_create]...]

especificación_create:
{ [DEFAULT] CHARACTER SET juego_de_caracteres
| [DEFAULT] COLLATE nombre_de_colación }
```

Antes que nada unas aclaraciones sobre este tipo de notación:

- Cuando se puede elegir un elemento entre varios éstos van entre llaves **{ }** y separados por un pipe **|**.
- Cuando un elemento es opcional (puede incluirse u omitirse) se pone entre corchetes **[]**.

Centrándonos en la sentencia, se empieza por *CREATE*, seguido de *DATABASE* o *SCHEMA* (en este caso son sinónimos, es indiferente cual de los 2 se elija) y después se pone el *nombre* deseado para la base de datos. Como elementos adicionales (opcionales) tenemos:

- *IF NOT EXISTS* se puede añadir antes del nombre de la BD. Con esto solo la crea si no existe ya una BD con ese nombre, esto simplemente evita que salte un error y permite al proceso continuar si tenemos un script con varias sentencias. Cuidado, ya que estas se aplicarán probablemente sobre esa BD ya existente.
- Detrás del nombre de la BD se pueden añadir especificaciones sobre el conjunto de caracteres a utilizar o bien la forma de tratar ese conjunto de caracteres.
 - *CHARACTER SET* establece el juego de caracteres a usar, se recomienda *"utf8mb4"*.
 - *COLLATE* influye sobre como se tratan caracteres especiales. Esto es crucial para poder comparar o ordenar alfabéticamente campos con acentos o la letra ñ (en caso contrario, por ejemplo, las letras acentuadas se ordenan detrás de todas las demás en vez de ser equivalentes a la letra sin acento). Se recomienda usar *"utf8mb4_spanish_ci"*.

Ejemplo

Escribimos en el terminal de MySQL:

```
CREATE DATABASE prueba1 CHARACTER SET utf8mb4 COLLATE
utf8mb4_spanish_ci;
```

MySQL nos indica que la sentencia se ha ejecutado correctamente (*Query OK*). Si intentamos crear de nuevo la misma base de datos se producirá un error indicando que no se ha podido crear la base de datos porque ya existe. Para evitar este error debemos emplear la clausula opcional *IF NOT EXISTS*.

```
CREATE DATABASE IF NOT EXISTS prueba1
CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci;
```

Si volvemos a ejecutar ahora, puesto que la BD *"prueba1"* ya existe, la sentencia no hace nada pero tampoco devuelve ningún error, permitiendo que un proceso que tuviese varios pasos continúe. De hecho el sistema incluso devuelve un *Query OK*, eso sí, acompañado de un warning advirtiendo que la BD no se ha creado.

Así como los errores se detallan inmediatamente por pantalla, los warnings en MySQL solo nos advierten de su aparición, para ver los detalles de los mismos se debe lanzar el comando `SHOW WARNINGS;`.

Se puede usar el comando `SHOW DATABASES;` para ver las bases de datos que existen, y que efectivamente se ha creado la BD "*prueba1*". Aunque se haya creado la base de datos, MySQL no la usa inmediatamente, necesitamos un comando para conectar con la misma. En este caso el comando que permite cambiar la BD activa es `USE nombre_bd`, éste no necesita terminar en `;`.

En caso de querer borrar una base de datos existente se puede usar la sentencia `DROP DATABASE [IF EXISTS] nombre_bd;`. Recuerda que estas operaciones no se pueden deshacer, mucho cuidado con ellas.

```
mysql> CREATE DATABASE IF NOT EXISTS prueba1 CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci;
Query OK, 1 row affected (0.00 sec)

mysql> CREATE DATABASE IF NOT EXISTS prueba1 CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Note  | 1007 | Can't create database 'prueba1'; database exists |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| prueba1        |
| sys            |
+-----+
5 rows in set (0.01 sec)

mysql> USE prueba1
Database changed
mysql> DROP DATABASE prueba1;
Query OK, 0 rows affected (0.01 sec)
```

4. Creación de tablas

4.1. Nombres en las tablas

Conviene seguir una serie de normas para los nombres, no sólo de las tablas sino también de campos o restricciones. Cada SGBDR tiene algunas condiciones propias pero en general una buena práctica sería seguir estos puntos:

- Los nombres deben ser auto-descriptivos (partiendo del diseño relacional ya deberían serlo).
- Deben comenzar con una letra.
- No deben tener más de 30 caracteres.
- Sólo utilizar letras minúsculas del alfabeto (inglés), no se pueden usar espacios, usar un guion bajo para "separar" palabras.
- No puede haber dos tablas con el mismo nombre para el mismo esquema/BD.
- No puede haber dos campos con el mismo nombre en la misma tabla, en tablas distintas sí.
- No pueden coincidir con el nombre de una palabra reservada SQL (por ejemplo no se puede llamar SELECT a una tabla).

Se desaconseja el uso de letras mayúsculas por compatibilidad y para distinguir fácilmente las palabras propias de las sentencias SQL y los comandos de los nombres, pero no es una regla absoluta y es común encontrar nombres con letras mayúsculas (de hecho una práctica bastante común es usar CamelCase para los nombres). Por otro lado, cada SGBD trata el reconocimiento de mayúsculas de manera distinta, así que evitar su uso evita problemas de compatibilidad. Curiosamente, en MySQL ni siquiera depende del propio SGBD sino que adopta el funcionamiento del SO en el que esté funcionando, sobre Windows no distingue si las letras son mayúsculas o no pero sobre sistemas UNIX sí lo hace.

4.2. Tipos de datos

Por desgracia, los SGBD **no** reconocen todos los tipos de datos del estándar SQL, la mayoría los adapta a alguno de sus tipos propios y los amplía con varias opciones más. Usar tipos estándar aumenta la compatibilidad aunque no la garantiza del todo, igualmente es altamente recomendable, aunque signifique también de perder algo de personalización y optimización.

Si en el diseño relacional no nos han proporcionado información (dominio), tendremos que elegirlo nosotros. Es conveniente dedicar algo de tiempo a pensar bien el tipo y tamaño de cada columna.

Los tipos de datos estándar más habituales y compatibles entre los SGBDR se pueden ver en las siguientes tablas (se ofrecen algunos comentarios y tipos adicionales de MySQL y Oracle):

Tipos numéricos:

TIPO	DESCRIPCIÓN
INTEGER	Los valores mínimo y máximo dependen del SGBD, INT suele ser equivalente
SMALLINT	Puede contener un rango de valores más pequeño
BIGINT	Puede contener un rango de valores más grande, Oracle no lo usa
DECIMAL(p, s)	Número decimal, precisión p (número de dígitos), escala s (número de decimales)
FLOAT(p)	Numérico aproximado, p precisión de mantisa
REAL	Igual que FLOAT, pero el SGBD define la precisión
DOUBLE PRECISION	REAL con mayor capacidad
BOOLEAN	0/1, TRUE/FALSE, Oracle no lo implementa
NUMBER(p,s)	Propio de Oracle, transforma los enteros y decimales estándar a este tipo
TINYINT	Usado por MySQL entre otros, rango menor a SMALLINT

Tipos textuales:

TIPO	DESCRIPCIÓN
CHAR(n)	Cadena de caracteres de longitud fija n,
VARCHAR(n)	Cadena de caracteres de longitud variable, máximo n

Tipos para fecha y hora:

TIPO	DESCRIPCIÓN
DATE	Representa una fecha
TIME	Representa un tiempo con horas, minutos, segundos (milisegundos opcionales)
DATETIME	Representa una fecha y un tiempo
TIMESTAMP	Suele tener menor rango que DATETIME y tener en cuenta la zona horaria



! Mucha atención a las **fechas** y horas pues en este aspecto los SGBD suelen tener sus propias particularidades.

- **Oracle solo usa DATE** (que funciona como un *DATE TIME*) y **TIMESTAMP**.
- Generalmente los tipos de fecha **son compatibles** entre SGBD pero **no usan el mismo formato** para mostrarse ni para interpretarse a partir de una cadena de texto, por ejemplo:
 - **MySQL:** '(AA)AA-MM-DD' Año en 2/4 cifras, mes y día en 2 cifras cada uno.
 - **Oracle:** 'DD-MES-AA(AA)' Día 2 cifras, mes en 3 letras, año 2/4 cifras
(El separador en ambos casos también puede ser '/')

Por ello se recomienda usar el formato de fecha propio de cada SGBD para introducir fechas o usar funciones donde definimos el formato que estemos usando:

- **MySQL:** *STR_TO_DATE(cadena, formato)*
https://www.w3schools.com/sql/func_mysql_str_to_date.asp
- **Oracle:** *TO_DATE(cadena, formato)*
https://www.databasestar.com/oracle-to_date/

4.3. CREATE TABLE

Esta es la instrucción que permite crear una tabla. Aunque su sintaxis completa es algo más compleja, en este curso se sigue la siguiente:

```
CREATE TABLE [IF NOT EXISTS] [nombre_bd.]nombre_tabla (
nombre_columna tipo_datos [definición_columna]
[,nombre_columna tipo_datos [definición_columna]]...
[,restricción_tabla]
[,restricción_tabla]...
)


definición_columna:
[PRIMARY KEY] [NOT NULL] [UNIQUE] [DEFAULT expr] [AUTO_INCREMENT]

restricción_tabla:
[CONSTRAINT nombre_restricción]
{ CHECK (condición)
| {UNIQUE | PRIMARY KEY} (nombre_columna [,nombre_columna]...)
| FOREIGN KEY (nombre_columna [,nombre_columna]...)
REFERENCES nombre_tabla (nombre_columna [,nombre_columna]...)
[ON DELETE | ON UPDATE {CASCADE | SET NULL | SET DEFAULT}]}
```

Básicamente, definimos la tabla con su nombre y después una lista de sus campos con su tipo de datos separados por comas y todo entre paréntesis.

Detrás del tipo de datos se pueden añadir opcionalmente una serie de definiciones adicionales de columna/restricciones en función de si el campo es la CP, ÚNICO, VNN, se le quiere dar un valor por defecto (*expr* es un valor o expresión para ese valor) o que sea un índice que se incrementa automáticamente. Aunque se pueden indicar varios juntos no tiene sentido poner *NOT NULL* o *UNIQUE* junto a *PRIMARY KEY*, pues ya están incluidos. Recuerda que una tabla solo puede tener una CP, no puedes incluir esta definición en 2 columnas.

Como dato interesante, algunos SGBD incluyen una restricción *NOT NULL* en *UNIQUE*, así que los campos únicos no pueden ser nulos, pero ese no es el caso de MySQL.

 Las definiciones adicionales de columna no pueden expresar claves/restricciones compuestas, para ellas se deben usar restricciones de tabla.

Finalmente, y aunque se pueden incluir en cualquier orden, incluso delante de otras columnas, se incluye opcionalmente una lista con las restricciones de tabla:

- Aunque darle nombre a la restricción es opcional (el SGBD le dará automáticamente uno), es altamente recomendable incluirlo siempre, ya que se necesita este nombre para modificar o eliminar esta restricción posteriormente.
- *PRIMARY KEY* y *UNIQUE* sirven para otorgar esta restricción a un campo (si no se ha puesto en las definiciones de columna) o a un grupo de campos, cuando estas claves son compuestas. En este segundo caso es obligatorio crear esta restricción de tabla, ya que las definiciones de columna son independientes y no pueden funcionar en grupo.
- Aunque otros SGBD lo permiten, MySQL no considera *NOT NULL* una restricción sino una propiedad de la columna, así que solo puede definirse arriba junto al campo, y no aquí.
- *FOREIGN KEY* permite definir una clave ajena; si tiene varios campos se ponen todos dentro del paréntesis separados por comas. Siempre va acompañada de un *REFERENCES* donde se indica la tabla y los campos a los que se referencia con la CAj. Opcionalmente puede llevar la *restricción de borrado/actualización*, si no se incluye se implementa como borrado y actualización restringidos. Estos tres/cuatro elementos (*FK, REFERENCES* y *ON DELETE/UPDATE*) van siempre juntos en una misma restricción. Los campos que sean una clave ajena deben tener el mismo dominio que el campo al que referencian.

- Para poder especificar una FK la tabla y los campos referenciados deben haber sido creados previamente.
- **CHECK** permite establecer condiciones que deben cumplir los valores introducidos en los atributos, puede afectar a varios atributos de la tabla. La condición puede ser cualquier expresión válida que sea cierta o falsa y puede contener funciones, atributos de la propia tabla, literales o una lista de valores dentro de un paréntesis separados por comas. Se pueden usar también operadores lógicos (AND, OR y NOT). No está permitido hacer referencias a campos de otras tablas, ya que esto requiere realizar consultas.

Para mostrar una lista con todas las tablas existentes en una base de datos, se puede usar `SHOW tables;` y para ver una descripción detallada de sus campos `DESC nombre_tabla;`. Cuidado, pues el comando SHOW no es compatible con todos los SGBD, en ocasiones se necesita consultar las tablas internas de los mismos para ver esta información.

Para borrar una tabla existente `DROP TABLE [IF EXISTS] nombre_tabla;` es la sentencia adecuada. Recuerda que no podrás recuperarla tras ejecutarla.

Ejemplo

Vamos a crear las siguientes tablas de ejemplo para Empleados y Departamentos, teniendo en cuenta que todo empleado debe pertenecer a un departamento. De momento vamos a utilizar pocos campos, simplemente para ir probando y ver el comportamiento de las bases de datos. Más adelante crearemos ejemplos más complejos, más cercanos a la realidad.

<u>cod_dpt</u>	nombre_dpt*	ubicacion
INF	Informática	Planta sótano U3
ADM	Administración	Planta quinta U2
COM	Comercial	Planta tercera U3
CONT	Contabilidad	Planta quinta U1
ALM	Almacén	Planta baja U1

<u>dni</u>	nombre_emp*	especialidad	fecha_alta	<u>dpt*</u>
12345678A	Alberto Gil	Contable	10/12/2010	CONT
23456789B	Mariano Sanz	Informática	04/10/2011	INF
34567890C	Iván Gómez	Ventas	20/07/2012	COM
45678901D	Ana Silván	Informática	25/11/2012	INF
5678012E	María Cuadrado	Ventas	02/04/2013	COM
67890123A	Roberto Milán	Logística	05/02/2010	ALM

Para la creación de la tabla departamentos utilizaremos la siguiente sintaxis:

```
CREATE TABLE departamentos (  
  cod_dpt VARCHAR(4) PRIMARY KEY,  
  nombre_dpt VARCHAR(20) NOT NULL,  
  ubicacion VARCHAR(30)  
);
```

Podemos probar a escribirlo directamente en la línea de comandos, pero es poco práctico, será muy fácil cometer un error y tener que volver a escribirlo todo, por ello **es aconsejable escribirlo con un editor de texto y después ejecutarlo como un script** o bien ir pegando los comandos. Otra opción es escribirlo sobre la pestaña SQL si usamos gestores gráficos. Como se ha comentado, en estos apuntes vamos a hacerlo todo por línea de comando.

Prestad atención a cómo se ha establecido el campo cod_dpt como clave primaria con **PRIMARY KEY** y se ha indicado que el campo nombre_dpt no se puede quedar sin valor con **NOT NULL**.

Se crea ahora la tabla de empleados:

```
CREATE TABLE empleados (  
  dni VARCHAR(10) PRIMARY KEY,  
  nombre_emp VARCHAR(30) NOT NULL,  
  especialidad VARCHAR(20),  
  fecha_alta DATE,  
  dpt VARCHAR(4) NOT NULL,  
  CONSTRAINT emp_fk_dpt FOREIGN KEY(dpt) REFERENCES  
  departamentos(cod_dpt)  
);
```

En esta ocasión la PK es el dni y existe una restricción de tabla de tipo *FOREIGN KEY*. Puesto que se ha dicho que todos los empleados pertenecen a un departamento, el campo correspondiente a la CAj, *dpt*, debe ser *NOT NULL*. Recuerda que empleados no puede crearse antes que departamentos, o la restricción fallará. Los detalles sobre las restricciones se ampliarán un poco más adelante en la unidad.

El proceso entero, volviendo a crear la BD prueba1, quedaría así:

```
mysql> CREATE DATABASE IF NOT EXISTS prueba1
-> CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci;
Query OK, 1 row affected (0.01 sec)

mysql> use prueba1;
Database changed
mysql> CREATE TABLE departamentos (
-> cod_dpt VARCHAR(4) PRIMARY KEY,
-> nombre_dpt VARCHAR(20) NOT NULL,
-> ubicacion VARCHAR(30)
-> );
Query OK, 0 rows affected (0.04 sec)

mysql> CREATE TABLE empleados (
-> dni VARCHAR(10) PRIMARY KEY,
-> nombre_emp VARCHAR(30) NOT NULL,
-> especialidad VARCHAR(20),
-> fecha_alta DATE,
-> dpt VARCHAR(4) NOT NULL,
-> CONSTRAINT emp_fk_dpt FOREIGN KEY(dpt) REFERENCES departamentos(cod_dpt)
-> );
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_prueba1 |
+-----+
| departamentos      |
| empleados          |
+-----+
2 rows in set (0.00 sec)

mysql> DESC departamentos;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cod_dpt    | varchar(4)    | NO   | PRI | NULL    |       |
| nombre_dpt | varchar(20)   | NO   |     | NULL    |       |
| ubicacion  | varchar(30)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

4.4. Scripts

Una forma habitual de trabajar con bases de datos es mediante scripts o guiones que contienen las órdenes que se quieren ejecutar. Éstos pueden crearse en cualquier editor de texto y guardarse con extensión SQL.

Para nuestro primer script simplemente vamos a copiar las órdenes completas del ejemplo anterior, creando la base de datos, marcando que vamos a usarla y creando las 2 tablas. El contenido del script es:

```
CREATE DATABASE IF NOT EXISTS prueba1
CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci;
use prueba1;
CREATE TABLE departamentos (
cod_dpt VARCHAR(4) PRIMARY KEY,
nombre_dpt VARCHAR(20) NOT NULL,
ubicacion VARCHAR(30)
);
CREATE TABLE empleados (
dni VARCHAR(10) PRIMARY KEY,
nombre_emp VARCHAR(30) NOT NULL,
especialidad VARCHAR(20),
fecha_alta DATE,
dpt VARCHAR(4) NOT NULL,
CONSTRAINT emp_dpt_fk FOREIGN KEY (dpt) REFERENCES
departamentos (cod_dpt)
);
```

Para ejecutar los scripts en MySQL se usa `SOURCE ruta\nombre_script` (al ser un comando no acaba en ';'). Por ejemplo si se tiene el script en C:\BD\scripts en Windows el comando sería `SOURCE C:\BD\scripts\prueba1.sql`. De nuevo, este es un comando propio de MySQL, cada SGBD tiene su propio método para lanzar scripts; en Oracle en vez de `SOURCE` se usa `@`.

Fíjate que si no borras las tablas anteriores el script devolverá errores, ya que la base de datos y las tablas ya existen. Para eso existen las cláusulas `IF NOT EXISTS`, de este modo la instrucción no se ejecuta y solo devuelve un warning. En la siguiente captura se han ejecutado 3 scripts; prueba1nok.sql es el script mostrado arriba, prueba1.sql es el mismo añadiendo IF NOT EXISTS a la creación de tablas y prueba2.sql crea una BD distinta.

```
mysql> source prueba1.sql
Query OK, 1 row affected, 1 warning (0.00 sec)

Database changed
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> source prueba1nok.sql
Query OK, 1 row affected, 1 warning (0.01 sec)

Database changed
ERROR 1050 (42S01): Table 'departamentos' already exists
ERROR 1050 (42S01): Table 'empleados' already exists
mysql> source prueba2.sql
Query OK, 1 row affected (0.01 sec)

Database changed
Query OK, 0 rows affected (0.05 sec)

Query OK, 0 rows affected (0.04 sec)
```

- *prueba1.sql* se ejecuta sin errores pero en realidad no ha hecho nada, si miras los warnings verás que en todas las sentencias, puesto que el elemento ya existía, simplemente ha ignorado el comando por la opción *IF NOT EXISTS*.
- *prueba1nok.sql* en cambio solo tiene *IF NOT EXISTS* en la creación de la BD, así que las demás sentencias fallan, aunque el script se sigue ejecutando en MySQL no siempre tiene por qué ser así y puede dar lugar a problemas.
- *prueba2.sql*, al usar una base de datos nueva, ejecuta normalmente todas las sentencias. También se podrían haber borrado las tablas y la BD con las sentencias *DROP* adecuadas, incluso dentro del script para asegurarse de "reiniciar" la BD; eso sí, en este caso asegúrate de incluir el *IF EXISTS* para que no devuelva errores si la BD no existe, *DROP DATABASE IF EXISTS prueba1*.

4.5. Restricciones de tabla

En realidad existen restricciones de columna también, pero puesto que MySQL no permite ponerles nombre y tienen ciertas funciones limitadas, las ignoraremos, usando siempre restricciones de tabla.

Como se ha comentado, conviene poner un nombre a todas las restricciones, ya que se usa para modificarlas o eliminarlas, procesos comunes en la gestión de BD. En caso contrario deberemos buscar el nombre que automáticamente la haya otorgado el SGBD, y cada uno usa sus propias normas y tiene su propio método para encontrar ese nombre. Los nombres de restricción no se pueden repetir para el mismo esquema/BD, debemos de buscar nombres únicos.



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.

Desde la empresa Oracle se aconseja la siguiente regla a la hora de poner nombre a las restricciones:

- *Tres letras* para el nombre de la *tabla*.
- Carácter de subrayado.
- *Tres letras por columna* afectada por la restricción.
- Carácter de subrayado.
- *Dos letras* con la abreviatura del tipo de restricción. La abreviatura puede ser:
 - *nn. NOT NULL* (En MySQL no se considera restricción, no se usa)
 - *pk. PRIMARY KEY*
 - *uk. UNIQUE*
 - *fk. FOREIGN KEY*
 - *ck. CHECK* (validación)

Por ejemplo *emp_dpt_fk* ayuda a indicar que el campo *dpt* de la tabla *empleados* tiene una restricción de clave ajena (*FK*).

4.5.1. NOT NULL



NOT NULL indica que ese campo no puede quedar con valor nulo en ninguna tupla.

Recuerda que MySQL no la considera una restricción sino una propiedad del campo así que no se le puede poner nombre. Además no permite determinar nombres en restricciones de columna, de modo que no las usamos.

En otros SGBD como Oracle se le puede poner nombre a esta restricción tanto si es de columna como si es de tabla, con la palabra **CONSTRAINT**.

4.5.2. UNIQUE



UNIQUE indica que ese campo o conjunto de campos no puede adoptar el mismo valor/es en filas distintas para toda la tabla.


En algunos SGBD incluye además la restricción **NOT NULL**, no es el caso de MySQL ni de Oracle pero debes tenerlo presente cuando lo uses. Otro factor interesante es que algunos SGBD, como Oracle **NULL** se considera un valor como cualquier otro y por tanto sólo puede haber un elemento nulo en un campo con restricción **UNIQUE**. En cambio MySQL permite repetir el valor nulo, al considerar que no es realmente un valor sino la ausencia del mismo.

En MySQL puede especificar en la columna, o como restricción de tabla si se le quiere dar nombre con **CONSTRAINT**. Si es una restricción **compuesta** debe aplicarse como **restricción de tabla**.

```
CREATE TABLE cliente(  
dni VARCHAR(9) UNIQUE  
);  
ó  
CREATE TABLE cliente(  
dni VARCHAR(9),  
CONSTRAINT cli_dni_uk UNIQUE(dni)  
);  
  
CREATE TABLE cita (  
dni VARCHAR(9),  
fecha DATE,  
CONSTRAINT cit_dnifec_uk UNIQUE(dni, fecha)  
);
```

Recuerda que las restricciones de tabla siempre deben especificar los campos a los que se refieren, aunque sea solo uno.

4.5.3. PRIMARY KEY

 PRIMARY KEY indica la columna o columnas que identifican a cada registro de la tabla. Hace que los campos que la forman sean NOT NULL y UNIQUE.

```
CREATE TABLE cliente(  
dni VARCHAR(9) PRIMARY KEY,  
nombre VARCHAR(50)  
);  
ó  
CREATE TABLE cliente(  
dni VARCHAR(9),  
nombre VARCHAR(50),  
CONSTRAINT cli_dni_pk PRIMARY KEY(dni)  
);  
  
CREATE TABLE cita (  
dni VARCHAR(9),  
fecha DATE,  
lugar VARCHAR(50),  
CONSTRAINT cit_dnifec_pk PRIMARY KEY(dni, fecha)  
);
```

Técnicamente la mayoría de SGBD permiten definir tablas sin clave primaria, pero es algo completamente a evitar, ya que rompe todas las reglas de los esquemas relacionales.

4.5.3. FOREIGN KEY

 FOREIGN KEY indica que uno o más campos de una tabla deben corresponderse con campos de otra tabla, estos se indican con REFERENCES.

Se recomienda introducir las claves ajenas siempre como restricción de tabla, de hecho MySQL es la única opción viable que tiene. Se debe incluir siempre tanto el nombre de la tabla como los campos referenciados.

```
CREATE TABLE cliente(  
dni VARCHAR(9) PRIMARY KEY,  
nombre VARCHAR(50)  
);  
  
CREATE TABLE cita (  
dni VARCHAR(9),  
fecha DATE,  
lugar VARCHAR(50),  
CONSTRAINT cit_dnifec_pk PRIMARY KEY(dni, fecha),  
CONSTRAINT cit_dni_fk FOREIGN KEY(dni) REFERENCES cliente(dni)  
);
```

Con esto, en las citas solo se pueden incluir números de dni que existan en la tabla cliente. Si la clave es compuesta se ponen todos los campos separados por comas dentro del paréntesis, tanto en el FOREIGN KEY como en el REFERENCES, supongamos una tabla con una CAJ que referencie a *cita*:

```
CONSTRAINT X_fk FOREIGN KEY(dni, fecha) REFERENCES cita(dni, fecha)
```

Se le pueden añadir restricciones de borrado/actualización si es necesario.

```
CONSTRAINT cit_dni_fk FOREIGN KEY(dni) REFERENCES cliente(dni) ON  
DELETE CASCADE
```

Recuerda que la integridad referencial es una herramienta imprescindible de las bases de datos relacionales. Por ejemplo, si borramos un registro en la tabla cliente que está relacionado con una o varias citas ocurrirá un error, ya que de


permitirnos borrar el registro ocurriría fallo de integridad (citas refiriéndose a una cliente que ya no existe).

Por ello el estándar de SQL permite añadir las opciones sobre como actuar ante estos casos, añadiéndolas **en la FOREIGN KEY tras la cláusula REFERENCES**. Estas se pueden aplicar tanto a borrados (*DELETE*), como a modificación de la CP (*UPDATE*):

- *ON {DELETE | UPDATE} SET NULL*. Coloca nulos en los registros donde aparece en la clave ajena el valor borrado/modificado en la otra tabla.
- *ON {DELETE | UPDATE} CASCADE*. Borra todos los registros donde aparece en la clave ajena el valor borrado/modificado en la otra tabla.
- *ON {DELETE | UPDATE} SET DEFAULT*. Coloca el valor por defecto en los registros donde aparece en la clave ajena el valor borrado/modificado en la otra tabla. Debe existir un valor por defecto para esos campos definidos con DEFAULT en la creación de la tabla.
- *ON {DELETE | UPDATE} {NO ACTION | RESTRICT}*. Salta un error que impide borrar/modificar el registro donde estaba la clave principal. Valor por defecto, si no se pone nada actúa así.

En muchos SGBD, como MySQL, se admite el uso tanto de ON DELETE como de ON UPDATE con todas sus opciones, pero no en todos. **Oracle** por ejemplo tiene una política aún más estricta con la integridad referencial y sólo permite *CASCADE* o *SET NULL* para los borrados y ninguna opción para las actualizaciones, ya que considera que una CP nunca debe cambiar su valor.

4.5.4. CHECK

 **CHECK (o la restricción de validación) dicta condiciones que deben cumplir los valores en los campos de la tabla. La condición se pone entre paréntesis.**

Las condiciones pueden afectar a más de un campo de la tabla pero nunca a otros campos fuera de la misma. Además se pueden añadir varias restricciones a un mismo campo.

La condición puede ser cualquier expresión válida que de como resultado *TRUE/FALSE* y puede contener funciones, atributos de la propia tabla, literales. Se pueden usar también operadores lógicos (*AND*, *OR* y *NOT*). Mención especial merece el comparador *IN*, que permite comprobar si un valor está contenido en un grupo de opciones, generalmente definidas dentro de un paréntesis y separadas por comas.

Por ejemplo:

```
CREATE TABLE cita (  
  dni VARCHAR(9),  
  fecha DATE,  
  lugar VARCHAR(50),  
  precio DECIMAL(6,2),  
  CONSTRAINT cit_dnifec_pk PRIMARY KEY(dni, fecha),  
  CONSTRAINT cit_prec_ck CHECK(precio>0 AND precio<1000),  
  CONSTRAINT cit_lug_ck CHECK(lugar IN ('A1', 'A2', 'B1', 'B2'))  
);
```

El primer **CHECK** fuerza que el precio de la cita esté entre 0 y 1000, ambos no incluidos.

El segundo obliga a que lugar solo pueda adquirir uno de los valores de esa lista, **mucha atención ya que se deben cerrar los 2 paréntesis, el del CHECK y el del IN**. Es un error común olvidarse los paréntesis o alguno de ellos y no ver donde falla la sentencia de creación al saltar el error.

Como se ha comentado, se pueden poner varias restricciones **CHECK** al mismo parámetro, de modo que la restricción del precio se puede expresar también como 2 simples, sin el **AND**.

```
CONSTRAINT cit_prec_ck1 CHECK(precio>0),  
CONSTRAINT cit_prec_ck2 CHECK(precio<1000)
```

Las restricciones además pueden afectar a varios campos en la tabla, por ejemplo, suponiendo que ahora la tabla contiene 2 campos **precio1** y **precio2**, sería válido:

```
CONSTRAINT cit_prec_ck CHECK(precio1+precio2>1000)
```

5. Modificación de tablas

Igual que ocurre con la creación de tabla, el proceso de modificación de tablas existentes se realiza mediante una única sentencia que tiene una nomenclatura compleja. No veremos aquí todas las opciones, sino una versión reducida con los siguientes parámetros:

```
ALTER TABLE nombre_tabla opción_alter [, opción_alter]...

opción_alter:
{ RENAME nombre_tabla_nuevo
| RENAME COLUMN nombre_columna TO nuevo_nombre
| ADD restricción_tabla
| ADD nombre_columna tipo_datos [definición_columna]
  [{FIRST | AFTER nombre_columna}]
| DROP {nombre_columna | CONSTRAINT nombre_restricción}
| DROP PRIMARY KEY
| MODIFY nombre_columna tipo_datos [definición_columna]
  [{FIRST | AFTER nombre_columna}] }
```

Se pueden añadir varias opción_alter a una misma sentencia para realizar varios cambios a la vez. Incluso en una misma opción existe la posibilidad de añadir varios elementos entre paréntesis separados por comas, pero creemos mejor no complicar la nomenclatura para empezar.

Veamos en detalle cada opción, para ello usaremos la tabla de empleados de prueba¹ y desharemos los cambios cada vez para volver al estado original.

5.1. Cambiar de nombre una tabla

La opción **RENAME** tras el nombre de la tabla permite directamente cambiar el nombre de la misma.

```
ALTER TABLE nombre_tabla RENAME nombre_nuevo

ALTER TABLE empleados RENAME empleado;
ALTER TABLE empleado RENAME empleados;
```

5.2. Cambiar de nombre una columna

Para cambiar de nombre una tabla se necesita añadir **COLUMN** a **RENAME** y poner un **TO** entre el nombre viejo y el nuevo

```
ALTER TABLE nombre_tabla
RENAME COLUMN nombre_columna TO nuevo_nombre;

ALTER TABLE empleados RENAME COLUMN dni TO nif;
ALTER TABLE empleados RENAME COLUMN nif TO dni;
```

5.3. Añadir o quitar restricciones

Para añadir elementos se usa *ADD*; en este caso *ADD CONSTRAINT* seguido de la definición de la restricción igual que cuando se crea con *CREATE TABLE*.

Para eliminar una restricción se usa *DROP CONSTRAINT* seguido de su nombre.

```
ALTER TABLE nombre_tabla
ADD CONSTRAINT restricción tabla

ALTER TABLE nombre_tabla DROP CONSTRAINT nombre_restricción

ALTER TABLE empleados ADD CONSTRAINT emp_dni_uk UNIQUE(dni);
ALTER TABLE empleados DROP CONSTRAINT emp_dni_uk;
```

En este ejemplo se puede observar como es posible añadir una restricción de unicidad al campo *dni* a pesar de ser una clave primaria, lo que es redundante. Recuerda que *NOT NULL* en MySQL no se considera una restricción con lo que no puede añadirse aquí sino en la modificación de columnas.

5.4. Añadir o quitar columnas

Mismo procedimiento pero cambiando la palabra *CONSTRAINT* por *COLUMN* (opcional). MySQL tiene la opción adicional de añadir esta columna en un lugar determinado, o la primera (*FIRST*) o detrás de otra columna existente (*AFTER nombre_columna*); esta opción no es común a otros SGBD.

```
ALTER TABLE nombre_tabla
ADD definición_columna [{FIRST | AFTER nombre_columna}]

ALTER TABLE nombre_tabla DROP nombre_columna

ALTER TABLE empleados ADD salario DOUBLE AFTER nombre_emp;
ALTER TABLE empleados DROP salario;
```

5.5. Eliminar PRIMARY KEY

Existe una opción específica *DROP PRIMARY KEY* para eliminar las claves primarias, tanto si están definidas como restricción de tabla como en la propia columna.

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

Esto **no elimina los campos**, solo su funcionamiento como clave primaria.

5.6. Modificar columnas

Se usa la opción *MODIFY* seguida de un *nombre_columna* ya existente, su *tipo de datos* y sus *definición_columna*. No solo las nuevas, todas las que se quieran tener. Funciona como si se hiciese un DROP de la columna y se substituyese por la nueva definición más que como una modificación. Aún así, **en MySQL no anula una definición de PRIMARY KEY** aunque no se añada al *MODIFY*. En MySQL existe también una opción similar a la del *ADD* de columnas para además mover la posición de la misma.

```
ALTER TABLE nombre_tabla
MODIFY definición_columna [{FIRST | AFTER nombre_columna}]

ALTER TABLE empleados MODIFY dni VARCHAR(9) AFTER nombre_emp;
ALTER TABLE empleados MODIFY dni VARCHAR(10) FIRST;
```

Hay que tener en cuenta que si las tablas tienen datos, hay muchos cambios que no pueden realizarse si entran en contradicción con los datos existentes, como cambiar el tipo de valores o disminuir el tamaño. Las ampliación suele funcionar.

5.7. Modificar restricciones

Aunque otros SGBD sí que incorporan una opción para *MODIFY CONSTRAINT*, MySQL no la contempla. Se puede, no obstante, borrar la restricción (*DROP CONSTRAINT*) y añadirla(*ADD CONSTRAINT*) con la nueva definición con las opciones anteriormente vistas. Con esto queda visto todo lo que haremos mediante la instrucción *ALTER TABLE*.

5.8. Desactivar restricciones

A veces conviene temporalmente desactivar una restricción para saltarse las reglas que impone y poder rellenar los datos para posteriormente reactivarla. Esto sucede sobre todo cuando 2 tablas tienen claves ajenas cruzadas, con lo que no se puede añadir un registro en ninguna de ellas hasta que se añada a la otra... lo que resulta imposible. Para ello, algunos SGBD como Oracle incluyen una opción *DISABLE/ENABLE CONSTRAINT* al *ALTER TABLE*, pero este no es el caso de MySQL.

Para MySQL la opción que existe es la de desactivar temporalmente todas las FOREIGN KEYS. Ten presente que permite introducir información que rompa las reglas de integridad de las claves ajenas y no avisa al reactivarlas de que ahora no se cumplen, con lo que es un proceso enormemente delicado.

Esta opción es simplemente una variable interna de MySQL llamada *FOREIGN_KEY_CHECKS*; se puede poner a 1 para que las claves ajenas se comprueben (valor por defecto) o a 0 para desactivarlas.

```
-- Desactivar comprobación de claves ajenas
SET FOREIGN_KEY_CHECKS = 0;
-- Ahora se pueden romper las claves ajenas hasta que no
-- recuperemos el valor 1.
SET FOREIGN_KEY_CHECKS = 1;
```

5.9. Ejemplo completo de modificaciones

Vamos a añadir a nuestra base de datos *prueba1* una tabla con los proyectos en que trabaja la empresa. Inicialmente esa tabla se crea con dos campos, el código del proyecto y el nombre del proyecto pues la empresa aún no tiene muy claro cuál va a ser su tratamiento.

<u>cod_proy</u>	nombre_proy
MAD20	Repsol, S.A.
TO451	Consejería de Educación
V324	Oceanográfico

⚠ Intenta realizar los pasos por tu cuenta antes de comprobar la solución

- Crea la tabla.
- Modifica la estructura de la tabla mediante ALTER TABLE para cambiar los siguientes detalles:
 - El campo_proy requiere ahora un tamaño de 30 caracteres.
 - Se necesitan los campos fecha_inicio y fecha_fin, de tipo fecha.
 - Después de crear el campo fecha_fin, se decide finalmente que no es necesario.
 - Se requiere otro campo más (dpt) para indicar el departamento al que pertenece el proyecto. Además este campo actuará como clave ajena de la tabla departamentos con puesta a nulo en caso de eliminación del departamento y con actualización en cascada.

Primero la instrucción para crear la tabla, recuerda que puedes ejecutarla directamente o añadirla a un script para realizar todas las operaciones de golpe cada vez. Asumimos que la base de datos ya existe y tiene las tablas de empleados y departamentos.

```
use prueba1;
DROP TABLE IF EXISTS proyectos;
CREATE TABLE proyectos (
cod_proy VARCHAR(5) PRIMARY KEY,
nombre_proy VARCHAR(20)
);
SHOW TABLES;
DESC proyectos;
```

Mediante *SHOW TABLES* y *DESC proyectos* podemos comprobar que todo está como se espera.

Para modificar las características de un campo utilizaremos la cláusula *ALTER TABLE ... MODIFY* sobre el campo nombre. Procedemos a modificar el tamaño del *VARCHAR nombre_proy* a 30 caracteres y comprobamos mediante *DESC* que el cambio es correcto.

```
ALTER TABLE proyectos MODIFY nombre_proy VARCHAR(30);
DESC proyectos;
```

Se añaden ahora los campos de tipo *DATE*, lo haremos con una sola sentencia *ALTER* para los dos. Después de comprobar que se han creado procedemos a borrar *fecha_fin*.

```
ALTER TABLE proyectos ADD fecha_inicio DATE, ADD fecha_fin DATE;
DESC proyectos;
ALTER TABLE proyectos DROP fecha_fin;
DESC proyectos;
```

Finalmente se añade el campo *dpt*, que se situará detrás del *cod_proy* y se le añade la restricción de clave foránea.

```
ALTER TABLE proyectos ADD dpt VARCHAR(4) AFTER cod_proy;
ALTER TABLE proyectos ADD CONSTRAINT pro_dpt_fk
FOREIGN KEY(dpt) REFERENCES departamentos(cod_dpt)
ON DELETE SET NULL ON UPDATE CASCADE;
DESC proyectos;
SHOW CREATE TABLE proyectos;
```

En la última línea se ha añadido una nueva opción del *SHOW* (*SHOW CREATE TABLE*), esta muestra la orden completa interna que define la creación de una tabla en su estado actual. Así se pueden comprobar las restricciones e incluso averiguar el nombre de restricciones de la tabla en caso que no le demos uno. Esta nomenclatura es exclusiva de MySQL y además permite ver muchas otras opciones del *CREATE TABLE* que no se han incluido aquí.

```
-----+
| proyectos | CREATE TABLE `proyectos` (
| `cod_proy` varchar(5) COLLATE utf8mb4_spanish_ci NOT NULL,
| `dpt` varchar(4) COLLATE utf8mb4_spanish_ci DEFAULT NULL,
| `nombre_proy` varchar(30) COLLATE utf8mb4_spanish_ci DEFAULT NULL,
| `fecha_inicio` date DEFAULT NULL,
| PRIMARY KEY (`cod_proy`),
| KEY `pro_dpt_fk` (`dpt`),
| KEY `nombre_proy` (`nombre_proy`),
| CONSTRAINT `pro_dpt_fk` FOREIGN KEY (`dpt`) REFERENCES `departamentos` (`cod_dpt`)
ON DELETE SET NULL ON UPDATE CASCADE,
| CONSTRAINT `proyectos_ibfk_1` FOREIGN KEY (`nombre_proy`) REFERENCES `departamentos`
| (`cod_dpt`) ON DELETE SET NULL ON UPDATE CASCADE
| ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci |
+-----+
```



`SHOW CREATE TABLE nombre_tabla` proporciona una descripción completa del comando necesario para recrear una tabla en su estado actual. Permite ver detalles y nombres de todos los campos y restricciones. **Úsalo si necesitas conocer el nombre de una restricción.**

6. Borrado de Tablas

6.1. Borrar todo el contenido de una tabla

Oracle y MySQL disponen de una orden no estándar para eliminar definitivamente todos los datos contenidos dentro de una tabla, pero no la estructura de la tabla en sí. Incluso borra del archivo de datos el espacio ocupado por la tabla.

```
TRUNCATE TABLE nombre_tabla
```

6.2. Borrar una tabla por completo

Recuerda que con `DROP TABLE [IF EXISTS] nombre_tabla;` se pueden borrar las tablas. Ten en cuenta que al borrar una tabla:

- Desaparecen todos los datos.
- Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionará hay que eliminarlos a parte. Ya vemos sobre éstos cuando

estudiemos las consultas en próximas unidades.

- Lógicamente, sólo se pueden eliminar las tablas sobre las que tenemos permiso de borrado.
- Aún así, en MySQL no se pueden borrar tablas que estén referenciadas en claves foráneas para no romper la integridad de los datos. En Oracle existe la opción *CASCADE CONSTRAINTS*, a añadir al final de la sentencia anterior que elimina estas claves foráneas en las otras tablas (elimina la restricción, no el campo).
- Ten presente que no existe ninguna petición de confirmación y el cambio es irreversible, excepto que se dispongan de copias de seguridad de la base de datos entera.

7. Bibliografía

- MySQL 8.0 Reference Manual.
<https://dev.mysql.com/doc/refman/8.0/en/>
- Oracle Database Documentation
<https://docs.oracle.com/en/database/oracle/oracle-database/index.html>
- Comparación entre Oracle vs MySQL.
<http://arbo.com.ve/oracle-vs-mysql/>
- Adam McGurk.How to change a foreign key constraint in MySQL
<https://dev.to/mcgurkadam/how-to-change-a-foreign-key-constraint-in-mysql-1cma>
- Sqlines. MySQL - SET FOREIGN_KEY_CHECKS.
http://www.sqlines.com/mysql/set_foreign_key_checks