Apuntes accesibles UT4

Sitio: FRANCISCO DE GOYA
Curso: Bases de Datos Pendientes
Libro: Apuntes accesibles UT4

Imprimido por: Lucía Hernández Montero
Día: lunes, 24 de octubre de 2022, 09:32

Tabla de contenidos

1. Lenguaje DDL

Se denomina lenguaje de definición de datos (DDL, Data Definition Language) a un lenguaje único a través del cual se pueden expresar todas las operaciones relacionadas con la definición de datos. Está a disposición del administrador de la base de datos y quizás de otros usuarios avanzados.

La mayoría de los DDL comerciales relacionales están basados en el lenguaje SQL. En él la definición de datos se realiza a través de las sentencias de esquema (schema statements).

En el lenguaje DDL de SQL, se emplean sentencias CREATE para crear elementos mientras que se utilizarán sentencias DROP para eliminarlos.

Podremos crear, modificar y eliminar:

- Bases de datos o esquemas: que son los contenedores de las tablas. Una base de datos sería un conjunto de tablas relacionadas con un propósito concreto. Suelen ser la información persistente de una aplicación, como puede ser cualquiera de los ejercicios realizados en unidades anteriores. Pero para cada aplicación o proyecto tendremos una base de datos separada de las demás.
- Tablas: equivalentes a las relaciones vistas en la unidad anterior. Compuestas por columnas, que serán los atributos, cada uno de un tipo de datos concreto, y por filas, que serán un ejemplar único de la tabla.
- Restricciones de integridad, que representan las ya estudiadas en la unidad anterior, aunque solo a modo teórico. Tendremos claves primarias, claves únicas, claves ajenas y otras restricciones.

Además de todo esto por lo general los SGBD también crean, modifican y eliminan usuarios de la base de datos, pero esto se verá en unidades posteriores, o más en detalle en el caso de los estudiantes de ASIR en el segundo curso.

2. Instalación y uso de MySql

Para abordar esta unidad será necesario instalar y conocer las herramientas que se instalan junto al servidor MySql

2.1. Instalación del servidor mysql

El servidor mysql es un servicio, y no un programa como otro cualquiera. Una vez instalado estará siempre en segundo plano esperando atender las conexiones que vaya recibiendo desde los diferentes clientes.

Por ello, un usuario no maneja directamente el servicio como si fuese un programa que se abre y se cierra. El programa que se abre y se cierra que accede al servidor es el cliente, que se instala a la vez que el servidor. Pero además uno se puede conectar desde red al servidor.

La instalación de mysql dependerá del sistema operativo que se esté utilizando.

Ubuntu

En el caso de ubuntu y otros sistemas GNU/Linux similares se limita a los siguientes comandos:

- apt update
- apt upgrade
- apt install mysql-server
- mysql_secure_installation

En el último comando se ha de indicar que se desea usar el plugin de contraseñas. Se elegirá la fortaleza mínima de las contraseñas, que para una base de datos de pruebas y ejercicios puede ser la mínima, pero en el caso de producción debería ser la máxima.

Posteriormente se da contraseña al usuario root, aunque no será necesaria ya que se puede acceder mediante sudo. Se puede permitir en el caso de la asignatura que el usuario root se conecte a través de red, pero es aconsejable no hacerlo generalmente.

Windows

Hay alguna guía muy útil para instalar windows, donde se indica de dónde descargar el instalador y los pasos de la instalación:

https://www.profesionalreview.com/2018/12/13/mysql-windows-10/

Una de las partes más importantes es no olvidar la contraseña que se da al usuario root durante la instalación.

2.2. Inicio del cliente mysql

Hay diferentes formas de utilizar el servidor mysql.

- Las aplicaciones y programas tienen plugins específicos (según el lenguaje de programación con el que fueron creados) que conectan al SGBD.
- Hay clientes específicos para ser usados por usuarios de bases de datos:
  - Gráficos: en muchas ocasiones (como en windows) se instala Mysql Workbench. Es útil y completo.
  - Textuales: en todo caso se instala el cliente que se usa a través de consola. Esta forma de conectarse será la que se seguirá en el módulo, pues una vez se maneje con soltura es fácil usar herramientas gráficas. Pero el camino a la inversa es más complicado, y hay ocasiones en las que se dispone solo de conexiones por consola.

Ubuntu

Iniciar el cliente de mysql una vez instalado el servidor es sencillo con el usuario administrador del sistema:

- sudo mysql

En las ocasiones en las que usemos otro usuario, con conexión local sería:

- mysql -u nombre_usuario -p

Y si nos conectamos con un usuario desde la red:

- mysql -u nombre_usuario -p -h ip_o_dns_del_servidor

Windows

En windows si accedemos desde inicio aparecerá un programa llamado mysql shell. No es del todo igual que el de ubuntu, y hay que usar un comando "sql" para que cambie al modo equivalente.

Pero se instala también el cliente equivalente, y se puede usar desde consola (CMD). Para ello hay que abrir un CMD, ir a un directorio y ejecutarlo:

- cd \Program Files\Mysql\Mysql Server 8.0\bin

Una vez en ese directorio se puede ejecutar de la misma forma que en windows.

También es posible ejecutarlo sin tener que ir al directorio antes indicado, pero para ello hay que cambiar la variable de entorno PATH, como se indica aquí:

http://chuwiki.chuidiang.org/index.php?title=Path_de_ejecutables_en_Windows

2.3. Uso del cliente mysql

Una vez hemos iniciado el cliente aparecerá el prompt de mysql:

```
mysql>
```

Podremos entonces utilizar sentencias SQL y comandos para utilizar las direrentes bases de datos.

El uso básico necesario es el siguiente.

```
show databases;
```

Este comando mostrará las bases de datos existentes en el SGBD. Son los contenedores de las tablas y cada una representa un proyecto. En el módulo lo habitual será utilizar una base de datos por ejercicio.

Inicialmente no se está en ninguna base de datos, por lo que no se pueden crear elementos.

```
create database nombre_bd;
```

Mediante este comando se creará una base de datos vacía.

```
use nombre_bd;
```

Mediante se usará la base de datos, por lo que en adelante se podrá crear elementos como tablas dentro de esta.

Si ya estábamos en una, se cambiará de base de datos y los cambios se aplicarán sobre la nueva.

```
drop database;
```

Este comando elimina una base de datos completa, con todas sus tablas y otros elementos que contenga. Es relativamente irreversible, por lo que hay que usarlo debidamente.

```
drop table nombre_tabla;
```

Se eliminará una tabla completa, no solamente sus filas. Dejará de existir en la base de datos utilizada.

3. Creación de tablas

La creación de tablas es la parte central de la unidad. La creación de una tabla se realiza a través de la sentencia CREATE TABLE, indicando el nombre de la tabla, el esquema correspondiente (lista de columnas y características), y restricciones de integridad (constraints) de tabla. Las características de una columna comprenden el tipo de datos y las restricciones de integridad de columnas.

La sintaxis de creación de tablas es la siguiente:

```
CREATE TABLE nombre_tabla ({desc_col|restricción} [,{desc_col|restricción}])
```

Indica que se ha de usar la sentencia CREATE TABLE, indicar un nombre, y posteriormente todo irá entre paréntesis: descripción de columnas o restricciones.

Internamente una columna puede tener algunas restricciones, pero no todas, por lo que será necesario en ocasiones realizar las restricciones por separado.

Podría tener una forma similar a la siguiente:

```
create table tabla1(
   columna1 tipo restricción1,
   columna 2 tipo,
   columna3 tipo restricción2,
   restricción3);
```

Separando cada columna y restricción en una línea es más fácil de leer posteriormente, aunque a MySQL le resulta indiferente.

Es también recomendable usar un editor de texto plano como gedit (Editor de textos en Ubuntu) o notepad ++ en Windows, y guardar ambos con extensión .sql. Es más sencillo corregir errores en los editores que en el cliente.

3.1. Restricciones de columna: null, default, primary key y unique

Tal y como se ha explicado, las restricciones pueden estar en una misma columna o ser especificadas individualmente. Pero solo algunas pueden ser especificadas en la propia columna. Son las siguientes:

- Null y not null.
- Default
- Primary key
- Unique

References no tiene funcionalidad aunque no de errores de sintaxis, por motivos de compatibilidad.

Null

Null indica que la columna que tenga esta propiedad puede no tener un valor, esto es, que al seleccionarse se mostrará null. No es necesario indicar null en una columna, ya que es la opción por defecto.

Sin embargo, indicar not null, obligará a que esa columna tenga un valor. No podrá usarse null. No es necesario indicarlo en una columna que tenga la restricción primary key, ya que esta restricción incluye not null.

```
nombre varchar(15) not null

apellido1 varchar(15) not null

 apellido2 varchar(15) null
```

Al ver valores de una tabla como esta podríamos encontrar:

nombre	apellido1	apellido2
Pedro	González	Suárez
Juan	García	Pérez
Jhon	Smith	null

No puede haber una fila con nombre o apellido1 con null. Pero sí es posible otras filas con apellido2 en null.

Hay que notar que una columna con null significa que no tiene valor. Por ejemplo, con números, 0 y null no son lo mismo. Y también con varchar, donde la longitud de la cadena es 0. De hecho es diferente el hecho de que sea null a la cadena "null", ya que la primera tiene longitud 0, y la segunda 4, estando formadas por los caracteres n, u, l y l.

Default

Esta restricción de columna hace que si no se indica un valor a una columna al insertar una fila (indicando el valor de otras columnas pero no de la que tiene default), esta tomará un valor por defecto que se indica:

```
fechaMatriculacion date default "2000-01-01"

autor varchar(20) default "Anónimo"

goles int default 0
```

Las formas de no indicar un valor en una columna de una fila son no indicarlo entre las columnas que se insertarán, o indicar la palabra default. Esto se verá en detalle cuando se vea la sentencia insert. Un resultado sería el siguiente al insertar en una tabla que tenga estos tres campos, y uno más a modo de identificador sin valor por defecto:

- ("2020-02-02","Juan",5,2)
- (default,default,3,1)
- ("2019-04-30","Ana",default,1)

fechamatricula	autor	goles	id
2020-02-02	Juan	5	2
2000-01-01	Anónimo	3	1
2019-04-30	Ana	0	1

Primary key

La restricción de clave primaria indica que una columna es clave principal o primaria. Esto significará que un valor concreto de esta columna identifica a una fila de forma unívoca. Se creará un índice implícitamente, lo cual facilita las búsquedas de datos en esta tabla usando esta columna. Por supuesto, solo puede haber una única clave primaria en cada tabla.

Por otro lado, a nivel de columna solo se puede indicar que una columna es la clave primaria. Si se quiere que una clave primaria esté formada por varias columnas, ha de especificarse como una restricción individual.

Ejemplos:

```
create table persona(
dni varchar(9) primary key
);
```

Sin enbargo, si se quiere que dos campos sean clave primaria, por ejemplo dni y idCompra, no sería correcto lo siguiente:

```
create table realizaCompra(
dni varchar(9) primary key
idCompra int primary key
); --> INCORRECTO
```

La forma correcta será la siguiente:

```
create table realizaCompra(
dni varchar(9),
idCompra int,
primary key (dni,idCompra)
);
```

En este caso, si tenemos las siguientes filas en la tabla realizaCompra:

dni	idCompra
123	1
123	2

Podremos insertar el valor ("121",2) y el valor (123,3), porque no se repite la clave primaria al completo. Recordemos que una clave primaria compuesta por dos campos, no significa que cada campo sea clave primaria. Los valores que no podríamos insertar son ("123",2) y ("123",1).

Unique

La restricción Unique es similar a primary key, pero en este caso más de una puede ser única, y también estar formadas por varios campos y haber varias de este tipo.

```
create table estudiante(
numeroMatricula int primary key,
dni varchar(9) not null unique,
nss varchar(15) unique
)
```

Una tabla como la anterior no podrá tener dos estudiantes con el mismo número de matrícula, ni con el mismo dni o número de la seguridad social. Pero a diferencia de la matrícula y el dni, podría no tener valor en el campo nss.

Al igual que con la clave primaria, una clave única compuesta por dos campos no implica que cada campo por su parte sea clave única. Si una tabla con dos campos, y ambos son una sola clave única, los valores (1,1) y (1,2) pueden estar ya que no se repite la clave única por completo, sino solo su primer campo, y por ello puede insertarse.

3.2. Restricción de clave primaria

La restricción de clave primaria indicada como restricción de tabla y no de columna, se indica de forma individual.

Tal y como se explicó en la restricción de columna **primary key**, no pueden repetirse valores de una clave primaria, tanto si está compuesta por una columna, como si está compuesta por varias. Pero en todo caso solo puede haber una clave primaria.

Un ejemplo para una clave primaria simple es:

```
create table persona(
dni varchar(9),
nombre varchar(20),
primary key (dni)
);
```

Si se quiere que dos campos sean clave primaria, por ejemplo dni y idCompra, no sería correcto lo siguiente:

```
create table realizaCompra(
dni varchar(9) primary key
idCompra int primary key
); --> INCORRECTO
```

La forma correcta será la siguiente:

```
create table realizaCompra(
dni varchar(9),
idCompra int,
primary key (dni,idCompra)
);
```

Ejemplos con datos en Restricciones de integridad de columna

3.3. Restricción de clave ajena

Una restricción de clave ajena obliga a que uno o varios campos tengan valores incluidos entre los campos de otra tabla, con las siguientes restricciones:

- Esos campos de la otra tabla han de ser clave primaria.
- Los campos referenciados han de ser de tipos idénticos. Un varchar(7) no podría referenciar a un varchar(8).

El caso más sencillo puede ser entre dos tablas:

```
create table coche(
  matricula varchar(8) primary key,
  caballos int
);
```

Y otra tabla que hace referencia:

```
create table clasificado(
  matricula varchar(8),
  foreign key (matricula) references coche(matricula)
);
```

Si tenemos las siguientes filas en coche:

matricula	caballos
0001ABC	150
2001ABC	180
2004ABC	200

En clasificado podremos insertar una de las matrículas que aparecen, pero no se podría insertar el valor ("5555ABC").

Una clave ajena puede ser compuesta también, y ello significa que en la tabla referenciada hay una clave primaria compuesta. Por ejemplo:

```
create table persona(
  dni varchar(9) primary key
);
```

Y la tabla matrimonio:

```
create table matrimonio(
  dni1 varchar(9),
  dni2 varchar(9),
  foreign key (dni1) references persona(dni),
  foreign key (dni2) references persona(dni),
  primary key (dni1,dni2)
);
```

Y tenemos otra tabla que le hace referencia:

```
create table divorcio(
  conyuge1 varchar(9),
  conyuge2 varchar(9),
  foreign key (conyuge1,conyuge2) references matrimonio (dni1,dni2),
  primary key (conyuge1,conyuge2)
```

```
);
```

Como se ve en este ejemplo se puede deducir que:

- Una clave ajena compuesta ha de referenciar a una clave primaria compuesta, de los mismos tipos de datos.
- Los nombres de las columnas que referencian no tienen por qué llamarse igual que las columnas referenciadas. De hecho es suficiente con que se indiquen en el orden correcto.
- Una clave ajena puede ser a su vez una clave primaria, o única.

Si ponemos datos a este ejemplo:

La tabla persona:

dni
0001
0002
0003
0004
0005
0006

La tabla matrimonio:

dni1	dni2
0001	0004
0002	0003
0001	0005

Finalmente la tabla divorcio:

conyuge1	conyuge2
0001	0004

Sería posible insertar en divorcio el valor (0002,0003), pero no sería posible insertar (0001,0003) porque no existe una fila en matrimonio con esos dos valores y en ese orden.

Tampoco puede insertarse en matrimonio los valores (0003,0009) ya que no existe 0009 en persona.

Acciones de borrado y actualización

Si una fila de una tabla que es referenciada por otra tabla va a ser borrada o actualizada, se ha de especificar qué sucederá con la fila que la referencia.

Por ejemplo, si en el caso anterior se borra la persona con dni 0001, ¿qué sucederá con el matrimonio con 0004, y a su vez, qué sucederá con el divorcio? Hay tres opciones:

- RESTRICT o NO ACTION: es la opción por defecto, ya que si no se indica nada es lo que sucederá.
  - En caso de intentar borrar el matrimonio (0001,0004) no se permitiría el borrado, ya que existe un divorcio que le referencia. Para ello habría que borrar antes el divorcio.
  - Si es una actualización, por ejemplo, cambiar el matrimonio (0001,0004) por (0006,0004), tampoco lo permitiría.
- CASCADE: en esta opción, todo cambio que afecte a la fila referenciada, afectará a las filas que les referencian.
  - Borrar el matrimonio (0001,0004) borrará el divorcio (0001,0004).
  - Actualizar el matrimonio (0001,0004) por (0006,0004), hará que el divorcio también cambie a (0006,0004).
- SET NULL: en el caso del borrado, y si la columna admite nulos, se podrá borrar. La fila que hace referencia tomará valor nulo.
  - Borrando la persona (0002), en matrimonio la segunda fila tomará los valores (null,0003).

Esto habrá que especificarlo para actualizar y borrar por separado. Véanse los siguientes ejemplos.

```
create table persona2(

  dni varchar(9) primary key

);
```

Y la tabla matrimonio:

```
create table matrimonio2(

  dni1 varchar(9),

  dni2 varchar(9),

  foreign key (dni1) references persona2(dni) on delete restrict on update cascade,
```

```
  foreign key (dni2) references persona2(dni) on update cascade,
  primary key (dni1,dni2)
);
```

Y tenemos otra tabla que le hace referencia:

```
create table divorcio2(
  conyuge1 varchar(9),
  conyuge2 varchar(9),
  foreign key (conyuge1,conyuge2) references matrimonio2 (dni1,dni2) on delete set null on update cascade
);
```

Usando los datos de los ejemplos anteriores:

```
insert into persona2 values ("1"),("2"),("3"),("4"),("5"),("6");
insert into matrimonio2 values ("1","4"),("2","3"),("1","5");
insert into divorcio2 values ("1","4");
```

Se explica de la siguiente forma:

- Cambiar en persona2 el dni 0004 por 0014, hará que en matrimonio cambie la fila (0001,0004) por (0001,0014). Y a su vez en divorcio también (0001,0014). En ambos hay actualización en cascada.
- No se podrá borrar en persona2 la fila con dni 0001.
- Tampoco se podrá borrar en persona2 la fila con dni 0003, por el hecho de que dni2 forma parte de la clave primaria.
- Si se actualiza la primera fila de matrimonio (0001,0004) por (0006,0004), en divorcio se tomará también (0006,0004).

3.4. Nombrado de restricciones

Es posible que se pida indicar un nombre a una restricción. No modifica en nada el comportamiento de cualquier restricción, pero sí añade código previo:

```
create table TB(
 a int,
 b int,
 constraint PK_TB primary key (a),
 constraint FK_TB_TC foreign key (b) references TC(x),
 constraint UQ_TB_B unique (b)
);
```

Es equivalente a:

```
create table TB(
 a int,
 b int,
 primary key (a),
 foreign key (b) references TC(x),
 unique (b)
);
```

Pero hay una ventaja en indicar el nombre de las restricciones: una vez se quiere modificar la tabla podría eliminarse, cambiarse el código y volver a crearla. Pero si la tabla tiene datos y está en producción esto no sería recomendable. Se necesitaría usar un "Alter table" que se ve en el epígrafe 5 de estos apuntes, y lo más sencillo es utilizar los nombres de las restricciones.

En el caso de la clave primaria no, ya que al haber solo una es suficiente con indicar "primary key" en el alter table. Pero en los otros casos sí es necesario utilizar "FK_TB_TC" y "UQ_TB_B".

4. Tipos de datos

Cada columna de una tabla deberá tener siempre uno de los tipos de datos que permite el SGBD que se use.

En general los tipos de datos son:

- Alfanuméricos: utilizando UTF-8, unicode, ASCII, etc, representan todos los caracteres posibles. Sobre ellos es posible realizar operaciones que funcionan sobre texto.
- Numéricos: ya sean naturales (binario puro), enteros (complemento a 2) o reales (coma flotante), tienen una codificación concreta que les limita en rango o precisión. Pero es el tipo de datos más adecuado para realizar operaciones matemáticas.
- Fechas: no se almacenan como un texto tal cual, las diferentes codificaciones tienen diferentes rangos (fecha mínima y máxima) y precisión (días, segundos, milisegundos). Puede operarse mediante funciones de fecha fácilmente.
- Binarios: pueden almacenar objetos binarios, ya sea una imágen, contenido multimedia, etc.

4.1. Tipos Alfanuméricos sencillos

Hay dos tipos alfanuméricos sencillos, varchar y char.

Char almacena caracteres de igual forma que lo haría varchar. Siempre hay que indicar la longitud máxima de la cadena, con el límite de 255. Si no se indica, la longitud por defecto es 1. Pero cualquier valor que se almacene ocupará siempre la longitud indicada:

```
nombre char(15)
```

Cuando se asigna un valor debe ser entre comillas dobles o simples:

```
"Pepe"

'Ana'
```

En ambos casos ocupará 15 caracteres.

El tipo varchar es el más usado, se utiliza de la misma forma (empleando comillas simples o dobles) y se define de la misma forma:

```
nombre varchar(15)
```

Pero a diferencia de char, las cadenas "Pepe" y 'Ana' ocupan solo 4 y 3 caracteres.

4.2. Tipos numéricos enteros o naturales

Los tipos numéricos enteros o naturales son muy utilizados, no solo por su significado literal, sino también porque suelen usarse para identificadores de tuplas.

Los números enteros son aquellos que no tienen decimales, abarcando los positivos y los negativos. Los naturales en cambio abarcan únicamente los positivos y el 0. Esto afecta a todos los tipos siguientes, cuya única diferencia son los rangos de valores que aceptan. Por defecto todos los tipos son enteros, excepto si se expresa antes del tipo la palabra reservada **unsigned**, que los convierte en naturales.

El tamaño que ocupa cada tipo de dato es el siguiente:

- tinyint: 1 byte.
- smallint: 2 bytes.
- mediumint: 3 bytes.
- int: 4 bytes.
- bigint: 8 bytes.

De esta forma, los rangos aplicables para cada tipo son los siguientes:

tipo	valor mínimo	valor exp	valor máximo	valor exp
tinyint	-128	-2^7	127	-2^7-1
tinyint unsigned	0	0	255	2^8-1
smallint	-32768	-2^{15}	32767	$2^{15}-1$
smallint unsigned	0	0	65535	$2^{16}-1$
mediumint	-8388608	-2^{23}	8388607	$-2^{23}-1$
mediumint unsigned	0	0	16777215	$2^{24}-1$
int	-2147483648	-2^{31}	2147483647	$2^{31}-1$
int unsigned	0	0	4294967295	$2^{32}-1$
bigint	$-9{,}223372037\times10^{18}$	-2^{63}	$9{,}223372037\times10^{18}$	$2^{63}-1$
bigint unsigned	0	2^{64}	$1{,}844674407\times10^{19}$	$2^{64}-1$

Se puede indicar una longitud a estos tipos, entre paréntesis:

```
stock int(5)
```

Este modificador no impide que se almacenen números con más de 5 dígitos. La única funcionalidad que tiene es usarlo en conjunción con **zerofill**, explicado más adelante, pero que será obsoleto en futuras versiones de MySQL.

4.3. Tipos numéricos de coma fija

Estos tipos de datos permiten que los valores numéricos tengan una cantidad fija de decimales, los cuales se indican al declarar el tipo de dato.

Aunque tienen nombres diferentes, realmente **numeric** y **decimal** son el mismo tipo de dato. Se indican de la siguiente forma:

```
dinero numeric(7,2)

edad numeric(3)

peso decimal(4,1)
```

Al definir cualquiera de los dos se indican dos números:

- El primero es el número de dígitos en total, contando enteros y decimales.
- El segundo es cuántos de los dígitos anteriores son decimales.

Por ello, los rangos de los valores indicados en el ejemplo serían:

- dinero: -99999.99 a 99999.99
- edad: -999 a 999
- peso: -999.9 a 999.9

Todo valor que exceda la precisión será redondeado. Por ejemplo, si se asigna un peso de 78.384 el valor almacenado será 78.4.

El mayor número de dígitos que se permite indicar es 65, aunque en la práctica puede estar restringido, dependiendo finalmente del sistema operativo.

4.4. Tipos numéricos de coma flotante

La coma flotante tiene una precisión limitada, pero su rango de valores es mucho mayor que cualquier otro tipo de dato numérico. Aunque en su versión de mayor precisión la precisión suele ser suficiente para la mayor parte de las medidas.

La mayor ventaja es que internamente se utiliza la codificación IEEE754, que es la que internamente emplean los microprocesadores (y otros elementos computacionales como tarjetas gráficas), por lo que una gran carga de trabajo operando estos tipos de datos será más eficiente que con coma fija.

Los dos tipos son los siguientes:

- float: 32 bits.
- double: 64 bits.

La diferencia, además de representar números absurdamente grandes como 2^{500}, es que la precisión si es superada truncará el valor. Por ejemplo:

```
a float

b double
```

Si insertamos los valores: 9000000.008, 9000000.8, 9000001 y 9000010, en un campo float (a) y en uno double (b), el resultado será el siguiente:

a	b
9000000	9000000,008
9000000	9000000,8
9000000	9000001
9000010	9000010

Se observa como el campo pierde precisión, truncando el valor insertado por aquel más cercano que pueda representar. Sin embargo, double no tiene tal problema.

De hecho, en el campo double se puede insertar el valor 9000000.00000008 sin perder precisión. Y en el caso de añadir un 0 más, el valor 9000000.000000008 se almacenará como 9000000.000000007, lo cual es muy cercano.

4.5. Tipos de fecha

Almacenar fechas no es algo sencillo, es algo más que almacenar el número del año, del mes y del día. Para poder operar con fechas (como por ejemplo sumar 304 días a una fecha concreta) se ha de tener en cuenta:

- Los días que tiene cada mes.
- Los años bisiestos.

Por ello los tipos de datos usados podrán manejar todas estas circunstancias para ofrecer un dato con el que operar de forma fiable. Cualquier ejemplo de un tipo de datos se ha de entrecomillar igual que las cadenas de caracteres, ya sea comillas dobles o comillas simples.

Según las necesidades están disponibles:

Tipo	Descripción	Formato	Límite inferior	Límite superior
date	Fecha, sin hora	YYYY-MM-DD	1000-01-01	9999-12-31
datetime	Fecha con hora	YYYY-MM-DD hh:mm:ss	1000-01-01 00:00:00	9999-12-31 23:59:59
timestamp	Fecha, con hora	YYYY-MM-DD hh:mm:ss	1970-01-01 00:00:01	2038-01-19 03:14:97

Los formatos significan:

- YYYY: el año con 4 dígitos. YY sería solo dos.
- MM: mes.
- DD: día.
- hh: hora.
- mm: minuto.
- ss: segundo. Puede tener decimales hasta la millonésima de segundo.
  - ss.xxxxxx, por ejemplo 12:03:35.023582

Las diferencias entre timestamp y datetime no se reducen al rango de fechas válidas:

- timestamp está basado en el formato de fecha unix, el cual admite zonas horarias que pueden actualizarse automáticamente.
- timestamp emplea 4 bytes, mientras que datetime emplea 8.
  - Las búsquedas mediante índice (primary key, unique e index) son más rápidas con timestamp.

4.6. Tipo enum

Este tipo es una forma de tener una serie de valores predefinidos, un tipo de dato personalizado.

Para ello no solo se indica que una columna es enum, sino que se indican los valores válidos.

```
color enum( 'blanco' , 'rojo' , 'azul' , 'amarillo' , 'verde')

talla enum( 'XS' , 'S' , 'M' , 'L' , 'XL', 'XXL')
```

Al asignar un valor a color, este solo podrá tener uno de los valores indicados. Por ello es útil para características que nunca van a variar.

En otros casos es posible que no sea recomendable. Por ejemplo, si se hace un enum de marcas de coches, posiblemente en el futuro aparezca una nueva marca y no podrá ser añadido fácilmente, de hecho será problemático. Sin embargo si se crea una tabla con la lista de marcas será trivial añadirla.

4.7. Otros tipos

MySQL admite otra serie de tipos de datos que aportan muchas utilidades diferentes:

- spatial: para almacenar coordenadas espaciales.
- binary: almacena binario, mostrándose también como binario.
- blob (tinyblob, blob, mediumblob, longblob): almacenan grandes objetos binarios, como pueden ser archivos multimedia (imágenes, vídeos, pdf).
- text (tinytext, text, mediumtext, longtext): almacenan más allá de los 255 caracteres de longitud de varchar y char.
- set: se define de forma similar a enum, pero una celda puede almacenar más de un valor a la vez. Simula conjuntos. Realmente se puede considerar que no cumpliría la 1FN, pero usado correctamente no tiene por qué dar problemas.

4.8. Modificador zerofill

Zerofill es un modificador que en las futuras versiones de MySQL quedará obsoleto, aunque sigue siendo compatible. Se podrá sustituir usando la función LPAD() a la hora de mostrar los datos. En todo caso se explica:

A cualquier tipo numérico se puede modificar con la palabra reservada **zerofill** para que a nivel de presentación aparezca con ceros a la izquierda de forma que aparezca siempre con la misma longitud. La restricción es que **no pueden ser valores negativos**.

Además de por motivos de presentación, puede tener cierta utilidad. Ordenar datos numéricos con zerofill tiene el mismo resultado que si el orden es alfabético.

Se indica después del tipo numérico pero antes de cualquier restricción de columna:

```
a double zerofill

b int(5) zerofill
```

Véanse los siguientes ejemplos de ambos campos, si insertamos los valores:

- Campo a:
  - 1.0002
  - 1000000000.0002
  - 1000000000.0001234412
  - 14411000000000.0001234412 (sobrepasa la precisión de double)

a
000000000000000001.0002
00000001000000000.0002
000001000000000.0001234
0000000014411000000000

- Campo b:
  - 5
  - 1415
  - 141545
  - 3545

b
00005
01415
141545
03545

Se puede observar cómo al insertar un número que precisa más de 5 dígitos (estando especificado el tipo como int(5)) el número se inserta realmente, pero su longitud será mayor que 5.

4.9. Auto incremento

Una columna que sea clave (primary key o unique) puede llevar un auto incremento. Esto es, si no se indica el valor en una inserción, se numerará automáticamente.

```
create table TA(
  id int auto_increment primary key,
 b int );
```

Si se realizan inserciones de forma que no se indique un valor a la columna id (como se verá detalladamente en la unidad 5), aparecerán valores automáticamente en dicha columna:

```
insert into TA (b) values (5),(3);

insert into TA values (null, 7), (null, 0);
```

El resultado sería:

TA

id	b
1	5
2	3
3	7
4	0

5. Modificación de tablas

Las tablas, una vez se han creado, pueden ser modificadas además de ser borradas. Para ello se podrá utilizar la sentencia alter table con muchas posibilidades.

- Añadir o eliminar columnas.
- Añadir o eliminar restricciones.
- Renombrar columnas, añadir valores por defecto o restricciones de nulo.

También serán indispensables si entre dos tablas hay referencias cruzadas. Esto es, que la tabla A tenga una FK a la tabla B a la vez que la tabla B tiene otra FK a la tabla A.

En todos los casos el sintaxis será el siguiente:

```
alter table <nombre_tabla> [opcion_alter]
```

Las opciones del alter son muy variadas, y se pueden ver en el enlace.

5.1. Añadir y eliminar columnas

A la hora de añadir o eliminar columnas tendremos que usar <add column> mientras que para eliminar una será <drop column>, según los siguientes ejemplos:

```
alter table A add column b varchar(10) null;
```

Se ha de indicar el nombre de la tabla, que en este caso es A. Tras añadir <add column> viene la definición de la columna, que **será igual que en la definición de una columna en un create table**.

```
alter table A drop column b;
```

En el caso de la eliminación, solo es necesario indicar el nombre de la tabla y de la columna. Esto, además, eliminará las restricciones que tuviese asociadas (como primary key, unique o foreign key).

Por supuesto, al añadir una columna se puede añadir una restricción asociada a esa columna únicamente. Por ejemplo, si no hay una clave primaria en una tabla, se puede añadir un campo nuevo que sea clave primaria:

```
alter table A10 add column x int primary key;
```

Sin embargo esto no podrá funcionar si la tabla ya tenía filas, ya que no se indica qué valor tendrán. De hecho mysql tratará de poner un 0 en cada. Si hay más de una no lo permitirá. Esto se soluciona con un auto_increment:

```
alter table A10 add column x int auto_increment primary key;
```

Esto numerará automáticamente todas las filas en su nueva columna desde el 1 hasta la última fila.

5.2. Añadir y eliminar restricciones

Al igual que con las columnas se puede añadir y eliminar restricciones. De la misma forma también, se ha de indicar el nombre de la tabla y después la definición de la restricción al igual que se indica en el create table.

```
alter table A add primary key(a);

alter table B add primary key(a,b);
```

Como se ve, es posible añadir claves primarias simples y compuestas. Pero a diferencia del **alter table add column** en este caso las columnas deben existir. Adicionalmente, si la tabla ya tenía filas, debe cumplirse que no haya repeticiones en la clave primaria, ni nulos.

Por otro lado, añadir **cualquier otro tipo de restricción será igual que en un create table**:

```
alter table A add foreign key(b) references B(id);

alter table A add constraint A_fk_B foreign key(b) references B(id);
```

Estas dos versiones son iguales, tal y como se vio con el create table. Pero para eliminar posteriormente la restricción es más útil la segunda dado que tenemos su nombre "A_fk_B":

```
alter table A drop constraint A_fk_B;
```

Funcionará de la misma forma con unique.

```
alter table A add constraint A_uq_1 unique (b);

alter table A drop constraint A_uq_1;
```

Y en este caso de restricción unique, al igual que con la primary, no debe haber filas con el valor en b repetido, o no se podrá crear la restricción.

5.3. Modificar columnas existentes

Si en vez de eliminar columnas para volverlas a crear se quiere mantener la existente (y los valores de sus filas) modificándola, se recurrirá a alter column, change o rename, dependiendo del alcance de los cambios.

El siguiente ejemplo tenemos la tabla con la columna original y el cambio.

```
create table A( a int );

alter table A change column a a varchar(5) not null;
```

Es posible hacer un cambio de int a varchar, haciéndose un cambio de datos automáticamente (siempre que quepa en 5 caracteres).

Esta forma de cambiar funciona de la siguiente forma: se indica el nombre de la columna que se cambiará, el nuevo nombre, el tipo y las restricciones. En este ejemplo no cambia el nombre de la columna, pero sí el tipo y que no pueda ser nulo.

Como en otros alter table, tras indicar la columna afectada, se describe la columna como en un create table.

Los siguientes dos ejemplos realizan otros dos cambios:

```
alter table A change column a a2 varchar(10) not null;

alter table A rename column a2 to a;
```

El primero renombra la columna a para que se denomine a2 en adelante, alargando la longitud a 10 y manteniendo la restricción de no nulo.

La segunda solo renombra de nuevo a2 para que vuelva a ser a. De esta forma (con rename) es más sencillo renombrar columnas, pero no puede realizar más cambios.

Por último , si se tiene una columna con un valor por defecto, este puede ser modificado:

```
create table C (c int default 0);

alter table C alter column c set default -1;
```

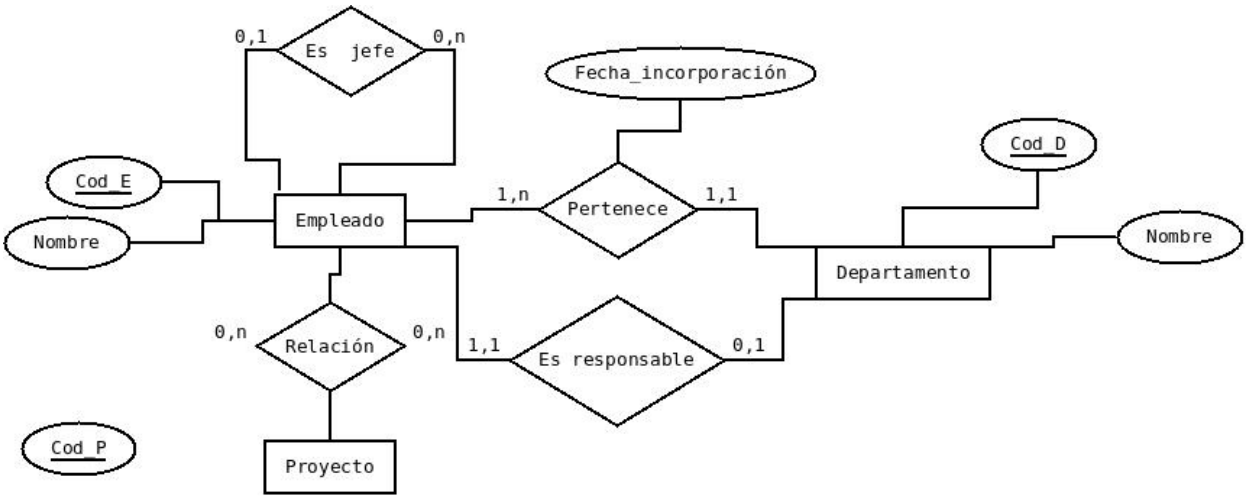
Tras el alter table, si se inserta un valor por defecto será -1 en vez de 0.

5.4. Tablas con referencias cruzadas

Aunque hay otros medios, como no evaluar las restricciones hasta que se indique, este es un método sencillo de realizar una referencia cruzada.

El ejercicio visto en la UT2 servirá de ejemplo, donde había dos entidades, Empleado y Departamento, que tenían dos relaciones entre si.

El resultado era tener dos tablas, cada una con una clave ajena a la otra.



Para crear estas dos tablas no será posible con dos create table sin más, dado que al crear la primera tabla no se puede hacer clave ajena a la segunda que aún no existe. Por ello se actuará de la siguiente forma:

```
create table Empleado(
   cod_e int primary key,
   nombre varchar(20),
   cod_d int);
create table Departamento(
   cod_d int primary key,
   nombre varchar(20),
   responsable int,
   foreign key (responsable) references Empleado(cod_e) );
alter table Empleado add foreign key (cod_d) references Departamento (cod_d);
```

La primera tabla es creada sin la FK. La segunda ya sí puede crearse con la FK referenciando a la primera. Tras esto, modificamos la primera tabla añadiendo la FK que faltaba.