Introducción a MariaDB

fanta <fanta@elbinario.net>

Fanta <fanta@elbinario.net>
19 de Febrero del 2015

0. Introducción a las bases de datos

0.0 Bases de datos

Las bases de datos también conocidas como **bancos de datos no son necesariamente digitales**. **Una biblioteca es una base de datos** compuesta por documentos y textos impresos almacenados en estanterías e indexados y catalogados para su consulta.

Actualmente estas bases de datos pueden estar digitalizadas completamente. Los documentos y libros pueden digitalizarse y almacenarse en discos duros en algún servidor y es posible desarrollar software que permita a los usuarios de forma presencial en una biblioteca o de forma remota desde sus casas (por ejemplo) realizar consultas a los documentos o libros que quiere.

La tecnología permite esto pero por el contrario las leyes de los diferentes países en materia de copyright no permiten que esto pueda hacerse de forma legal sin miles de trabas y mucho trabajo.

En el caso de una Hemeroteca donde consultar prensa esta muchas veces es parte de una biblioteca pero muchas otras veces este archivo le pertenece en exclusiva al periódico que publica.

Hoy en día estas Hemerotecas están digitalizándose y es posible acceder a ellas desde casa mediante una conexión a internet.

Estas bases de datos por tanto tienden a digitalizarse ya que esto permite ahorrar costes y muchísima comodidad para buscar lo que se quiere.

Tipos de bases de datos

Las bases de datos se pueden separar por tipos y estas separaciones dependen de las características de las bases de datos.

Nos encontramos por ejemplo con **bases de datos estáticas** y con **bases de datos dinámicas** y esta separación se realiza analizando la variabilidad de los datos.
Una b**ase de datos estática** es la que simplemente almacena datos que no van a variar y que permiten ser consultados pero no tiene sentido actualizarlos. Un ejemplo puede ser una biblioteca. Existen diferentes libros y en caso de una nueva edición de uno en especial este aparecerá como versión de 1932 o versión de 1988. Ambos serian datos estáticos y el libro en realidad seria diferente (contendría cambios).

En el caso de b**ases de datos dinámicas** son las que sus datos son muy cambiantes permitiendo operaciones como borrado, edición, etc.

Un ejemplo pueden ser los productos de un supermercado en venta. Productos que cambian de

nombre, cambian de precio, se dejan de vender, etc.

Estas bases de datos sirven para consulta pero al mismo tiempo son bases de datos dinámicas que podemos ir alterando. Los precios de los vuelos por ejemplo lo mismo.

También es posible encontrar diferentes tipos si nos centramos en la característica de la forma en la que se almacenan los datos. A esto se le llama "**modelos de bases de datos**".

Estos modelos pueden ser:

Bases de datos jerárquicas
Base de datos de red
Bases de datos transaccionales
Bases de datos relacionales
Bases de datos multidimensionales
Bases de datos orientadas a objetos
Bases de datos documentales
Bases de datos deductivas

Las bases de datos jerárquicas pueden ser por ejemplo cuando organizamos por directorios nuestros documentos en un sistema de archivos.
Podemos crear un directorio llamado libros, dentro de libros crear muchos directorios llamados por ejemplo "misterio, romántica, ciencia, idiomas, química, humor, ..." y dentro de estos otros sub-directorios llamados por ejemplo "español, inglés, chino, francés, ...".

Esta forma de organizar estos datos es bastante problemática ya que es posible que tendremos el problema de la redundancia de datos. Un libro por ejemplo que sea de romántico y de misterio estará 2 veces en diferentes niveles de la jerarquía.

El resto de modelos no nos interesan salvo el de **bases de datos relacionales**. Estas son las bases de datos que vamos a utilizar.

0.1 Bases de datos relacionales

Existe un buen articulo sobre esto en wikipedia que podéis consultar si queréis:

https://es.wikipedia.org/wiki/Base_de_datos_relacional

Aquí simplemente vamos a hablar un poco en forma de resumen de que tiene de especial este modelo de bases de datos y los motivos por los que es el modelo más usado.

Para entender el modelo relacional vamos a partir de un ejemplo en el que previamente analizamos las relaciones entre los datos para poder diseñar previamente una estructura.

En el caso de los libros de una biblioteca usando el modelo jerárquico tenemos problemas con la redundancia de datos. En el caso de analizar las relaciones vamos a ver que esto no ocurre si no queremos.

Partimos de la base de que todo en la biblioteca son **textos** pero estos pueden ser de por ejemplo estos **tipos:** periódicos, revistas y libros.

Apuntamos en una hoja la palabra **tipos** y seguimos analizando que estos pueden estar en diferentes **idiomas** (independientemente del tipo) de modo que añadimos en la hoja la palabra **idiomas**.
Estos idiomas podrían ser por ejemplo: Inglés, esperanto, francés, castellano, gallego, catalán, …

Nos encontramos con que todos los textos dando igual el tipo o el idioma se pueden ordenar en **categorías**. Apuntamos por tanto la palabra **categorías** en nuestra hoja.

Esas categorías podrían ser por ejemplo: Terror, comedia, teatro, ensayo, poesía, misterio, literatura técnica, manuales, ...

Ahora nos fijamos en que en la biblioteca los libros viene gente a llevárselos de modo que tenemos que fichar a la gente. Esto se hace haciendo carnets de la biblioteca y por tanto anotaremos **usuarios** a nuestra hoja.
Existen muchas más relaciones pero simplemente vamos a abstraernos y anotaremos una última llamada **estados**. Los libros pueden tener varios **estados** en la biblioteca: perdido, disponible, no-disponible.

Esto nos lleva a tener esto anotado en nuestra hoja de papel:

- Textos
- Tipos
- Idiomas
- Categorías
- usuarios
- estados

Y ahora podemos ir desarrollando que contenidos hemos de añadir en cada una de estas relaciones y lo vamos a hacer como si fuese una tabla con columnas.

Tabla estados:

id_estado	**nombre_estado**
1	Perdido
2	Disponible
3	No-Disponible

Tabla idiomas:

id_idioma	**nombre_idioma**
1	Castellano
2	Inglés
3	Francés
4	Alemán

Tabla tipos:

id_tipo	**Nombre_tipo**
1	Revista
2	Libro
3	Periódico

Tabla Categorías:

id_categoría	nombre_categoría
1	Química
2	Diseño
3	Misterio
4	Informática
5	Matemáticas
6	Aventura
7	Romance
8	Erótica

Tabla Usuarios:

id_usuario	Nombre_usuario	Primer_apellido	DNI	Fecha_registro
1	Benito	Camelas	60493219N	12/10/1999
2	Juanita	García	54787821A	01/07/2001
3	Roberto	Gómez	47854752F	05/04/2011
4	Matilda	Gatos	47514575B	07/07/2014

Tabla Textos:

id_texto	nombre	tipo	idioma	categoría	fecha	autor	edición	editorial	Estado
1	Babilonia	2	1	6	1982	Boby	1	bobylos	2
2	PHP simple	2	1	4	2012	Bea García	4	freakdev	3
3	Kamasutra	2	2	8	1992	breasdx	54	kama	2
4	Pc Mania 33	1	1	4	1996	Pcmania	1	Pcmania	2

Si nos fijamos sobre todo en esta última tabla nos damos cuenta que en ella vamos añadiendo el ID de lo que hemos definido en otras tablas. Por ejemplo el libro babilonia sabemos que es un libro por que en tipo tiene un 2 y si miramos en la tabla de tipos veremos que el 2 es para libros.

En el caso del mismo libro nos fijamos que vamos añadiendo el ID de otras tablas para la categoría, el estado, el idioma, etc.

Esto nos permite por ejemplo hacer consultas y de esta forma obtener información como por ejemplo si el libro esta disponible o no lo esta. Podemos buscar solamente por libros que estén en castellano o solamente en la categoría aventura.

Estas relaciones y esta forma de organizar en tablas la información nos permite no duplicar datos constantemente. Si deseamos el día de mañana añadir una nueva categoría podemos hacerlo sin problemas. Se añadiría a esa tabla una nueva categoría y los libros que entren en esa categoría se empezarían a añadir a la base de datos seleccionando esa categoría.

Este tipo de relaciones y lo que hemos realizado a boli primero sobre un papel son la forma de diseñar como será nuestra base de datos relacional. Hemos estado analizando la forma en la que queremos organizar los datos basándonos en las posibles relaciones pero se nos ha olvidado que lo mismo queremos obtener también quien tiene un libro en un momento dado.

Esto seria crear una nueva tabla llamada por ejemplo "**prestados**".

En esa tabla añadiríamos algo así como esto:

id_prestado	fecha_préstamo	texto_prestado	usuario_que_lo_tiene
1	12/01/2015	2	3

En este caso vemos que el texto prestado es el 2 y podríamos mirar en la tabla de textos para ver a que categoría pertenece (mirando a su vez en la tabla categorías) y ver el nombre del libro.

Cada vez que alguien se lleva un libro de la biblioteca una persona indicaría quien se lo lleva y este pasaría a la tabla de prestados y en la tabla de textos se cambiaría el estado del libro.

De esta forma podemos hacer una consulta a la tabla textos para saber que libros/revistas/periódicos están prestados actualmente. Podríamos imprimir ese resultado y tendríamos los libros que han sido prestados pero no a quien se ha prestado.

No importa ya que es posible realizar consultas cruzadas e ir sacando más datos de modo que si que podríamos saber e incluso sacar un listado con los libros prestados y a su vez a quien y cuando se han prestado. Si en la base de datos de usuarios tuviésemos un campo con el teléfono podríamos llamar a esas personas para avisarles de que han de entregar los libros o de lo contrario se formalizará automáticamente una multa con su nombre, apellidos, dni, …

Lo importante de este modelo es que no importa donde almacenemos la información o como ya que la vamos a recuperar mediante consultas.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como **normalización de una base de datos.**

Durante los años 80 la aparición de dBASE produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no utilizaba SQL como lenguaje base para su gestión.

Actualmente un software para gestión de bases de datos muy utilizado es MariaDB. Mucha gente conocerá esto por el nombre de Mysql server pero hemos de añadir que MariaDB es el fork libre realizado por su autor y que actualmente es lo que se usa en las comunidades de software libre.

Las diferencias frente a mysql no son muchas y al final de cuentas utiliza SQL como lenguaje para realizar las consultas.

Esto va a ser lo que vamos a ver. Como montar un servidor de bases de datos (que nos permita gestionarlas) y como empezar a administrar estas y realizar consultas.

0.2 Normalización de bases de datos

El proceso de normalización de una base de datos es un proceso en el que aplicamos una serie de reglas para entre otras cosas:

- Evitar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

Estamos diseñando las tablas y para esto es bueno pensar en que deseamos obtener. Podemos crear tablas con las relaciones como hemos realizado y en el proceso de normalización tener en cuenta que datos deseamos obtener.

Es un momento en el que tenemos que pensar en consultas posibles que nos ofrezcan los datos que deseamos y comprobar si con las tablas que tenemos esto es posible.

En el proceso de normalización vamos a buscar sobre todo errores y duplicidad de datos.

Las tablas han de tener todas una columna clave. Normalmente para que esto suceda en todas las tablas se suele poner (no necesariamente) una columna llamada id para identificar los datos.
Esa columna clave no necesariamente tienen que ser números correlativos pero es muy normal que esto se haga así para evitar problemas.

En el supuesto caso de una tabla donde almacenamos los usuarios de la biblioteca que se han registrado podríamos usar la columna DNI en vez de crear una columna llamada id_usuario (por ejemplo). Sabemos que no pueden existir 2 personas con el mismo número y letra de DNI pero lo cierto es que quizás el DNI no lo solicitamos obligatoriamente y por tanto esa columna no valdría ya que algunos usuarios podrían no tener DNI.

Puede darse el caso de una tabla de usuarios donde tengamos un campo llamado edad y otro llamado fecha de nacimiento.
En el proceso de normalización de la base de datos esto es un error. Con saber la fecha de nacimiento podemos sacar en cualquier momento la edad de modo que es un dato que no necesitamos.

Analizaremos por tanto todas las tablas para ver que datos son necesarios, que datos podemos deducirlos a partir de otros y sobre todo evitar que existan tablas con datos duplicados.

Las claves primarias no necesariamente son una columna. Es posible usar varias columnas para generar una clave primaria que nos sirva de identificador.

Un ejemplo es usar en una tabla de usuarios las columnas “nombre, apellido1, apellido2, fecha de nacimiento, localidad”. La probabilidad de que existan en nuestra tabla personas que se llaman igual, tienen los mismos apellidos, han nacido el mismo día y viven en la misma localidad es muy poca. Este riesgo de duplicidad podría asumirse en casos donde por ejemplo el listado de personas será muy reducido y la probabilidad de que ocurriese una duplicidad es improbable.

Usaríamos todas esas columnas como identificativo sin precisar por tanto de una columna llamada id.

En el proceso de normalización vamos a tener en cuenta estas cosas y a analizarlas. **Es muy importante que el acceso a los datos este garantizado en nuestras búsquedas.**

Cuando alguien (ya sea humano o sea software) realice una consulta a la base de datos partiendo de un nombre de tabla, el valor de la clave primaria y el nombre de la columna requerida tendría que obtener solamente un valor. Si se encuentra con más de un valor es que la cosa ha ido mal y esto es un error al diseñar la clave primaria seguramente.

Si por ejemplo tomásemos como clave primaria el DNI y por error alguien metiese el mismo que el de otra persona la consulta retornaria 2 datos en vez de 1. Esto supondría un error.

Las consultas tendrían que poder leer, escribir, eliminar y agregar registros (SELECT, UPDATE, DELETE e INSERT en SQL). En el proceso de normalización de la base de datos analizaremos en que casos se precisan todas las que sean distintas a select para ir diseñando que **tipos de usuarios se van a necesitar.**

Ningún componente de una clave primaria puede tener valores en blanco o nulos (ésta es la norma básica de integridad).

Si se diese el caso de que usásemos el DNI como clave primaria en una tabla es posible que si un usuario no añade el DNI o la aplicación que introduce los datos estuviese mal programada (permitiendo por ejemplo no meter el DNI como requerimiento) tuviésemos un campo null. Esto no sucedería si se usase un identificativo o clave primaria que no dependiese de los campos que rellena un usuario.

1. Primeros pasos con MariaDB

1.0 Instalar MariaDB en Debian GNU/Linux 7

```
apt-get install python-software-properties
apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
add-apt-repository 'deb http://tedeco.fi.upm.es/mirror/mariadb/repo/10.0/debian wheezy main'
apt-get update
apt-get install mariadb-server
```

Nos pedirá la clave de acceso como usuario root y entonces es cuando le metemos una clave.
Dejar sin clave el usuario root en la base de datos es un error grave.

No hemos de confundir el usuario root del sistema con el usuario root en la base de datos y por tanto **NO usaremos la misma contraseña para el usuario root en el sistema y para el usuario root en la base de datos.**

En Debian el archivo por defecto para configurar y afinar MariaDB estará en /etc/mysql/my.cnf y puede ser alterado en caso de que lo precisemos no obstante para una instalación en un ordenador de escritorio esto no es necesario.

En un servidor dedicado y dependiendo de la estimación de accesos simultáneos, visitas, peticiones, … se tocarán los valores de ese fichero. En nuestro caso no vamos a entrar en retoques personales.

1.1 Acceder por primera vez a la consola de MariaDB

Desde nuestro emulador de terminal, en el interprete de línea de comandos vamos a escribir lo siguiente para acceder por primera vez a la consola de MariaDB con el usuario root:

```
mysql -u root -p
```

Nos solicitará la clave de acceso una vez hemos pulsado intro y entonces si la clave es correcta accederemos a la consola de MariaDB.

Hemos de fijarnos que hemos indicado el argumento -u con root como usuario y luego el argumento -p para indicar que deseamos introducir la clave de ese usuario. Esto será igual para otros usuarios en la base de datos, es decir, si tenemos 3 usuarios más llamados: **benito, juanito, maria** los comandos para acceder a la consola serán iguales pero cambiando el nombre de usuario:

```
mysql -u benito-p
mysql -u juanito -p
mysql -u maria -p
```

Evidentemente aún no existen esos usuarios en la base de datos de modo que no vamos a poder acceder con ellos ni conocer su password.

Antes de aprender como crear usuarios desde la consola de MariaDB vamos a ver como es posible **salir de la consola.** Esto es muy sencillo y se hace **escribiendo quit** en la consola o también escribiendo **exit.**

```
MariaDB [(none)]> exit
```

Ya sabemos acceder a la consola y salir de la consola. Se empieza por algo.

1.2 Algunos comandos de utilidad

Aparte de saber entrar y salir de la consola hemos de saber muchas más cosas. Antes de meternos en faena y ver como añadir más usuarios que puedan acceder a la consola hemos de saber que **el usuario root** con el que estamos accediendo **es un usuario creado para administrar el servidor de bases de datos** y que por tanto no será nunca un usuario al que el resto de gente tenga acceso.

Con MariaDB podemos gestionar muchas bases de datos diferentes para muchos proyectos que no tienen por que guardar relación alguna entre ellos.

Supongamos que vamos a tener 3 clientes que quieren tener una base de datos en nuestro servidor de bases de datos. Nosotros/as N**O crearemos un solo usuario** y les daremos acceso a los clientes, nosotros/as crearemos un usuario por cada cliente o incluso un usuario por cada proyecto.

Antes de ver unos cuantos comandos de utilidad hemos de dejar esto claro ya que es un error de base que se comete a diario.

- **NO** dejaremos al usuario root en MariaDB sin clave de acceso. Siempre se le pone clave.
- La clave de root en MariaDB **NO** es la misma que la del usuario root en el sistema operativo.
- **NO** le daremos acceso a root a nadie.
- Crearemos diferentes usuarios a ser posible para cada proyecto o mínimo para cada cliente.

Es un error de seguridad que 2 proyectos diferentes de una misma empresa compartan el mismo usuario para acceder a una base de datos o varias. Es por esto que es mejor ir separando y limitando el acceso y cada base de datos tendrá su propio usuario.

En caso de que alguien consiga acceso a la base de datos sin autorización entrará con un usuario sin privilegios de administración y no verá todas las bases de datos, solamente verá las bases de datos sobre las que el usuario con el que esta accediendo tiene privilegios.

De todos modos antes de empezar a crear usuarios es bueno ver una serie de comandos de utilidad:

system

Con system vamos a poder indicar comandos del sistema y ver su salida. Ejemplo:

```
MariaDB [(none)]> system uptime
```

Esto nos permite estar en la consola de MariaDB pero ejecutar comandos del sistema operativo sin tener que salir de la consola de MariaDB.
En este ejemplo hemos usado uptime pero se puede usar “ping elbinario.net”, ls, top o cualquier comando de sistema que necesitemos.

Aparte de escribiendo system es posible usar el comando abreviado y obtener el mismo resultado:

```
MariaDB [(none)]> \! uptime
```

help

En caso de no recordar un comando podemos utilizar help o /? para obtener un poco de ayuda.

status

Con status vamos a poder ver un poco de información sobre el servidor de bases de datos y nuestra conexión. Si olvidamos que versión de mariadb estamos usando, con que usuario hemos accedido, … es posible usar status y ver esta información.

```
MariaDB [(none)]> status
--------------
mysql  Ver 15.1 Distrib 10.0.16-MariaDB, for debian-linux-gnu (x86_64) using readline 5.2

Connection id:          45
Current database:
Current user:           root@localhost
SSL:                    Not in use
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server:                 MariaDB
Server version:         10.0.16-MariaDB-1~wheezy mariadb.org binary distribution
Protocol version:       10
Connection:             Localhost via UNIX socket
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
UNIX socket:            /var/run/mysqld/mysqld.sock
Uptime:                 53 min 7 sec

Threads: 1  Questions: 713  Slow queries: 0  Opens: 575  Flush tables: 2  Open tables: 30  Queries per second avg: 0.223
--------------

MariaDB [(none)]>
```

prompt

El prompt es eso que vemos antes de escribir. Por defecto en MariaDB saldrá el siguiente:

```
MariaDB [(none)]>
```

Y se queda a la espera de que escribamos algo. Podemos cambiar el texto que sale a la izquierda por algo que no sea MariaDB[(none)]> y que de paso sea de utilidad.

```
MariaDB [(none)]> prompt \N [\d]
```

No se ve pero he añadido un espacio al final de modo que el prompt se quedará igual que como esta.
En caso de querer cambiarlo por otra cosa podemos usar estos ejemplos:

```
MariaDB [(none)]> prompt \D [\d]
Mon Feb 16 10:03:11 2015 [(none)]
```

Se mostrará la fecha en vez de MariaDB.

```
Mon Feb 16 10:03:11 2015 [(none)]  prompt \O [\d]
Feb [(none)]
```

Eso mostrará el Mes actual.

```
Feb [(none)] prompt \U [\d]
root@localhost [(none)]
```

Esto mostrará el usuario y la maquina en la que estamos conectados y con la \u pero minúscula se mostrará solamente el usuario sin @localhost o la maquina a la que estemos conectados.

Cada cual que use el prompt que más le guste.

1.3 Crear bases de datos y agregar usuarios

Ambas cosas van de la mano o han de ir de la mano. **Cuando creamos una base de datos es el momento perfecto para pensar en que va a necesitar un usuario que acceda a ella** (uno o varios).

Supongamos por un momento que una empresa necesita que todos sus empleados tengan acceso a un listado de números de teléfonos. Es una empresa grande y tienen 200 teléfonos internos controlados por una centralita. Estos teléfonos tienen cada uno una extensión y son usados por 200 personas.

El informático de esta empresa ha pensado que es buena idea crear una base de datos donde se mantenga esta información al día. Para ello ha desarrollado una aplicación que accede a una base de datos y la cual no nos importa lo más mínimo. Simplemente hemos de saber que tendrán a un becario/a todo el día revisando que no despidan a nadie y en caso de sustituciones o cambios estará encargado de cambiar esos datos.

Este informático programador necesitará 2 tipos de usuarios. Uno que solamente va a leer datos de la base de datos y otro que va a poder agregar nuevos, editar los ya existentes e incluso borrar.
A nosotros nos importa muy poco como desarrolle su aplicación pero si que nos importa limitar los permisos de los usuarios ajustando estos a lo necesario.

El tipo que ha programado esto es un tipo inteligente. Por un lado ha diseñado su aplicación para montarla en un servidor web y por otro lado ha pensado que **la base de datos es mejor que este en otro servidor de bases de datos.** Ambos servidores pueden estar en la misma maquina pero NO es

lo recomendable.

El servidor web en una maquina y el servidor de bases de datos en otra son buena cosa. Colocar ambos en la misma maquina no es lo más recomendable.

Lo que nos pide el programador es: Una base de datos a la que accedan 2 usuarios. Uno puede alterarla y otro será el usuario que solamente va a realizar consultas.
Esto traducido para nosotros supone que hemos de crear una base de datos con 2 usuarios que puedan acceder a ella siendo solamente uno de ellos el que va a poder alterar las tablas de esta base de datos.

Ver que bases de datos tenemos actualmente.

Antes de crear una base de datos vamos a ver cuales tenemos. Y con el comando status podemos ver con que usuario estamos accediendo. En nuestro caso como root ya que somos los/las administradores/as de este servidor de bases de datos.

```
MariaDB [(none)]> show databases;
```

Con show databases; (no olvidemos poner punto y como al final) se mostrarán las bases de datos actuales en nuestro servidor.

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
```

El resultado no será que no tenemos ninguna base de datos. En realidad tenemos 3 por defecto y mejor no tocarlas ni saber nada de ellas ahora mismo.

Cuando tengamos creada la base de datos “teléfonos” repetiremos este comando y se verá.

Crear una base de datos

Crear una base de datos es sencillo. La estamos creando como usuario root para posteriormente crear los usuarios y darle los permisos de acceso.

```
MariaDB [(none)]> create database empresa_telefonos;
```

El tema de darle nombre a las bases de datos es importante. Supongamos que la gente suele pedirte mucho este tipo de bases de datos. No pueden existir 2 bases de datos con el mismo nombre de modo que es buena cosa buscar una forma de nombrarlas. Si son clientes puedes usar el nombre de la empresa cliente (o el particular) para identificar rápidamente y que no tengan el mismo nombre en un futuro 2 bases de datos. Usaremos _ en vez de un espacio.
No se pueden añadir espacios al nombre de la base de datos.

Si nos confundimos al crear una base de datos podemos borrar esta base de datos rápidamente usando drop database y el nombre:

```
MariaDB [(none)]> drop database empresa_telefonos;
```

Crear usuarios

Es importante a la hora de crear usuarios especificar el contexto del usuario para evitar que este se conecte desde otras maquinas. Indicaremos la maquina local.

Esto se ve claro en el comando de creación de usuarios:

```
MariaDB [(none)]> CREATE USER 'nombreusuario'@'localhost' IDENTIFIED BY 'clavebuena'
```

Por otro lado hemos de indicar una clave robusta más o menos. Cuidado con esto ya que si ponemos claves malas la seguridad se comprometerá antes.

En el caso de los telefonos vamos a crear 2 usuarios primero y luego les daremos los permisos.

```
MariaDB [(none)]> CREATE USER 'picateclas'@'localhost' IDENTIFIED BY 'pica5password.';
MariaDB [(none)]> CREATE USER 'luser'@'localhost' IDENTIFIED BY 'luser5password.';
```

Hemos creado con eso 2 usuarios. Uno llamado picateclas y otro llamado luser. El picateclas va a poder realizar más cosas que el luser. El luser simplemente realizará consultas cada vez que quiera llamar a alguien de la empresa.

No nos importa como esta realizada la aplicación pero suponemos que posiblemente el usuario luser buscará en una cajita y filtrará por departamento o cosas así.

Vamos a crear por tanto los permisos del usuario luser y los permisos del usuario picateclas para que ambos puedan acceder a la misma base de datos pero con diferentes permisos sobre esta:

```
MariaDB [(none)]> GRANT SELECT ON empresa_telefonos.* TO 'luser'@'localhost'
IDENTIFIED BY 'luser5password.';

MariaDB [(none)]> GRANT SELECT,INSERT,DROP,CREATE ON empresa_telefonos.* TO
'picateclas'@'localhost' IDENTIFIED BY 'pica5password.';
```

Esto es importante.

picateclas = SELECT,INSERT,DROP,CREATE
luser = SELECT

Estamos indicando los permisos. Luser será el usuario que accederá para realizar consultas de lectura mientras que picateclas (el becario/a) será el que se encargará de acceder para añadir nuevos registros a la base de datos, alterarlos, etc.

Ambos usuarios solamente van a acceder a la base de datos empresa_telefonos de modo que simplemente se crean diferentes usuarios para limitar más el acceso y por tanto reforzar la seguridad.

Si un usuario no va a necesitar realizar operaciones de escritura mejor que no tenga permisos

para esto.

Salimos con **quit** o con **exit** y entramos de nuevo con el usuario picateclas. Si probamos a ver con ese usuario las bases de datos veremos que solamente nos sale una.

```
mysql -u picateclas -p

show databases;
```

2. Instrucciones SQL por categorías

Las instrucciones SQL se pueden organizar por categorías en función de su utilidad.

2.0 Instrucciones DDL
2.1 Instrucciones DML
2.2 Instrucciones TCL
2.3 Instrucciones SCL
2.4 Instrucciones Embedded SQL

2.0 Instrucciones DDL

Las instrucciones DDL (Data Definition Language, lenguaje de definición de datos) permiten **gestionar las "estructuras" de los objetos.**

- Crear, modificar, eliminar.
- Autorizar o prohibir el acceso a los datos.
- Activar o desactivar la auditoría.
- Añadir comentarios al diccionario de datos.

Las instrucciones DDL son:

CREATE, ALTER, DROP, GRANT, REVOKE, AUDIT, NOAUDIT, ANALYZE, RENAME, TRUNCATE, COMMENT, FLASHBACK y PURGE.

2.1 Instrucciones DML

Las instrucciones DML (Data Manipulation Language, Lenguaje de manipulación de datos) permiten la **gestión de los datos contenidos en los objetos existentes.**

- Añadir, eliminar y modificar filas.
- Visualizar el contenido de las tablas.
- Bloqueo de tablas.

Las instrucciones DML son:

INSERT, UPDATE, DELETE, SELECT, EXPLAIN, PLAN, LOCK TABLE y MERGE.

2.2 Instrucciones TCL

Las instrucciones TCL (Transaction Control Languaje, Lenguaje de Control de Transacción) se usan para **gestionar las modificaciones realizadas** con las instrucciones DML**.**

- Características de las transacciones.
- Validación y anulación de modificaciones.

Las instrucciones TCL son:

COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION y SET CONSTRAINT.

2.3 Instrucciones SCL

Las instrucciones SCL (Session Control Language, Lenguaje de control de sesión) permiten la **gestión de una sesión de usuario**:

- Modificación de las características de sesión.
- Activación y desactivación de los privilegios de usuario.

Las instrucciones SCL son:

ALTER SESSION y SET ROLE

2.3 Instrucciones Embedded SQL

Estas instrucciones nos permiten **integrar las instrucciones DDL, DML y TCL en un lenguaje de programación.**

- Declaraciones de objetos e instrucciones.
- Ejecución de instrucciones.
- Gestión de variables y cursores.
- Tratamiento de errores.

Las instrucciones Embedded SQL son:

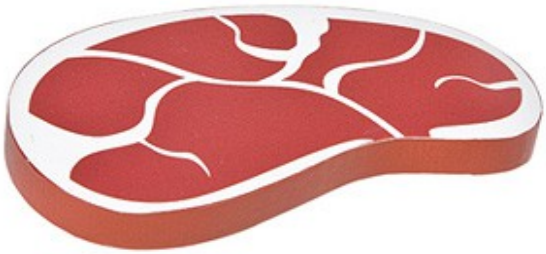
DECLARE, TYPE, DESCRIBE, VAR, CONNECT, PREPARE, EXECUTE, OPEN, FETCH, CLOSE y WHENEVER.

3. Tipos de datos

En este apartado vamos a ver tipos de datos pero no sin ver al mismo tiempo algunas instrucciones básicas que nos permitan un poco movernos en una base de datos.
Vamos a saber crear bases de datos, tablas, acceder a tablas, insertar datos en tablas, eliminar datos, renombrar tablas y realizar copias de seguridad de bases de datos.

3.0 Chuleta con las instrucciones básicas
3.1 Crear tablas y tipos de datos

3.0 Chuleta con las instrucciones básicas



Esta chuleta sirve un poco para moverse en bases de datos. Añadir registros y tablas son algo que no esta contemplado ya que requiere de algunos conocimientos más (no muchos más).

La idea es que simplemente con estos comandos podemos movernos y realizar consultas y gestiones básicas. Es evidente que para realizar consultas y gestiones más avanzadas se requiere de conocer unos cuantos más.

Mucha gente que no esta todo el día dándole a la tecla y gestionando bases de datos simplemente no retiene los comandos y les ocurre que cuando tienen que realizar una sencilla gestión tienen que regresar a mirarse algún manual.

Esto puede servir como chuletilla para la gente que simplemente usa las bases de datos y no las suele gestionar mucho.

Acceder desde la consola como un usuario determinado

Si quieres acceder como root el nombre de usuario ha de ser root. Es importante que el usuario root disponga de una buena contraseña y que las bases de datos que van a usarse con alguna aplicación web tengan su propio usuario y solamente se acceda a la base de datos con ese usuario.

```
mysql -u nombreusuario -p
```

Mostrar bases de datos actuales para el usuario con el que hemos accedido

```
SHOW databases;
```

Ver accediendo previamente como root los usuarios que pueden acceder a las diferentes bases de datos

```
SELECT user FROM mysql.user;
```

Ver el usuario actual con el que estamos accediendo

```
SELECT USER(),CURRENT_USER();
```

Crear bases de datos como root Con esto creamos una base de datos llamada shit. Solamente podrá acceder a ella de momento el usuario root. El usuario root solamente lo usaremos para gestionar y nunca lo usarán las aplicaciones que programemos o configuremos para que accedan a una determinada base de datos.

```
CREATE DATABASE shit;
```

Crear usuarios Es importante especificar localhost ya que indicando que el contexto sea la máquina local esto impedirá que el usuario se conecte desde otras máquinas. Es importante que la clave sea buena, larga, etc.. Vamos que sea una clave robusta.

```
CREATE USER 'nombreusuario'@'localhost' IDENTIFIED BY 'laclave';
```

Dar acceso limitado a un usuario para que pueda gestionar una base de datos Esto lo realizamos accediendo como root en la consola mysql.

```
GRANT SELECT, INSERT ON shit.* TO 'suario'@'localhost' IDENTIFIED BY
'lacontraseña';
```

Esto permitirá al usuario nombreusuario acceder con la contraseña que indiquemos a la base de datos shit y poder hacer select e insert. No se le permitirá crear tablas, eliminar tablas, renombrar o alterar tablas, … Si va a poder obtener datos de los registros e insertar nuevos datos.

Acceder/conectar/usar una base de datos

```
USE nombrebasededatos;
CONNECT nombrebasededatos;
```

Mostrar las tablas de una base de datos Una vez hemos accedido a la base de datos es con el siguiente comando:

```
SHOW tables;
```

Este otro nos sirve igualmente sin tener que acceder primero a la base de datos:

```
SHOW tables FROM nombrebasededatos;
```

Mostrar contenido de una tabla Si hemos accedido a la base de datos el comando más corto puede ser este:

```
SELECT * FROM nombretabla;
```

Si no estamos usando ninguna base de datos el comando es este otro:

```
SELECT * FROM nombrebasededatos.nombretabla;
```

Con * estamos indicando que se muestren todas las columnas. Si deseáramos solamente mostrar los datos de la columna id y la columna nombres es así:

```
SELECT id,nombres FROM nombrebasededatos.nombretabla;
```

Mostrar las columnas de una tabla determinada Si no hemos accedido a la base de datos así:

```
DESCRIBE nombrebasededatos.nombretabla;
```

Ver la estructura de la tabla:

```
DESCRIBE nombretabla;
```

Eliminar una tabla y su contenido

```
DROP TABLE nombretabla;
```

Eliminar el contenido de la tabla (los registros) pero no su estructura

```
TRUNCATE TABLE nombretabla;
```

Renombrar una tabla

```
RENAME TABLE nombretabla TO nuevonombretabla;
```

Realizar copia de una base de datos determinada:

```
mysqldump -u nombreusuario -p nombrebasededatos > nombrebasededatos.sql
```

Realizar copia de una base de datos determinada permitiendo restaurarla sin tener que eliminar la vieja:

```
mysqldump --add-drop-table -u nombreusuario -p nombrebasededatos >
nombrebasededatos.sql
```

Realizar restauración de la copia:

```
mysql -u nombreusuario -p nombrebasededatos < nombrebasededatos.sql
```

Estos últimos comandos se realizan con mucho cuidadin, bajo tu responsabilidad y fuera de la shell mysql.

3.1 Crear tablas y tipos de datos

Es **el momento que todos y todas estábamos esperando.** Es el momento de crear tablas.
Primero creamos una base de datos llamada **mis_contactos.**

```
CREATE DATABASE mis_contactos;
```

Para esto vamos a pensar en un ejemplo que nos sea de utilidad, por ejemplo una tabla con contactos de gente.
Esta tabla se llamará originalmente: **contactos**.

```
SHOW databases;
USE mis_contactos;
CREATE TABLE contactos;
```

Si nos fijamos al usar CREATE TABLE llamada contactos y no especificar nada más nos va a soltar la consola un error.

ERROR 1113 (42000): A table must have at least 1 column

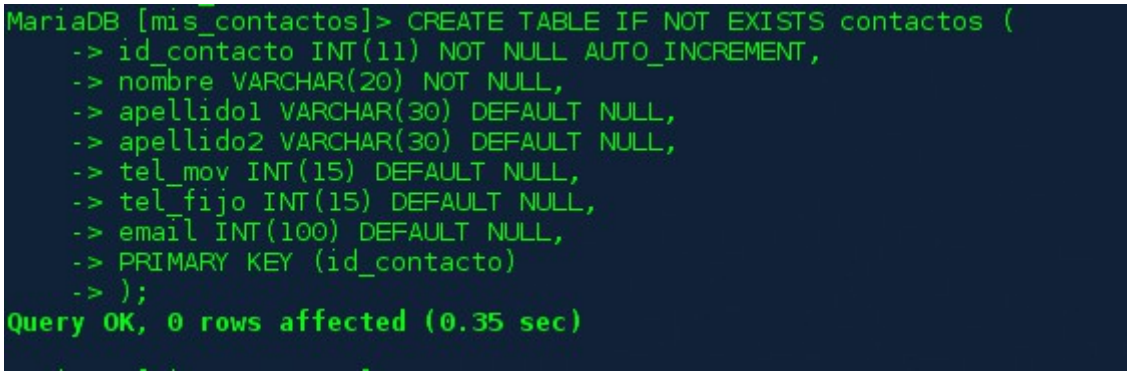
Esto significa que la tabla ha de tener al menos una columna. No podemos crear la tabla así como así de modo que vamos a crear una tabla pensando previamente que contenido vamos a meter. Pensamos un rato y al final obtenemos algo como esto:

id_contacto	nombre	apellido1	apellido2	tel_mov	tel_fijo	Email

Ahora vamos a crear la tabla con las columnas id_contacto, nombre, apellido1, apellido2, tel_mov, tel_fijo y email.

```
CREATE TABLE IF NOT EXISTS contactos (
      id_contacto INT(11) NOT NULL AUTO_INCREMENT,
      nombre VARCHAR(20) NOT NULL,
      apellido1 VARCHAR(30) DEFAULT NULL,
      apellido2 VARCHAR(30) DEFAULT NULL,
      tel_mov INT(15) DEFAULT NULL,
      tel_fijo INT(15) DEFAULT NULL,
      email VARCHAR(100) DEFAULT NULL,
      PRIMARY KEY (id_contacto)
);
```

Una vez hemos escrito todo ese chorro vamos a darle a intro:



Nos saldrá algo como en esa captura.

Posiblemente no entendemos nada ahora mismo o posiblemente si. No importa, vamos a explicarlo. Cuando vamos a crear la tabla podemos mirar si ya existe con la instrucción **SHOW TABLES;** no obstante podemos del mismo modo al crearla añadir IF NOT EXISTS para que se compruebe antes de crearla. **Si no existe entonces la va a crear.**

Ahora usando la tecla de los cursores hacia arriba (la fecha hacia arriba) podemos regresar a lo que acabamos de escribir y lo veremos todo en una sola linea. Si pulsamos ENTER va a intentar crear de nuevo la tabla pero nos mostrará un aviso (warning).

Podemos ver estos avisos con esta instrucción:

```
SHOW warnings;
```

Nos sacará algo así como esto:

```
MariaDB [mis_contactos]> show warnings;
+-------+------+-----------------------------------+
| Level | Code | Message                           |
+-------+------+-----------------------------------+
| Note  | 1050 | Table 'contactos' already exists |
+-------+------+-----------------------------------+
```

De modo que el aviso es que ya existe la tabla.

Ahora vamos a analizar las columnas que hemos creado y esas cosas raras que le hemos añadido para posteriormente analizar todos los tipos de datos.

Hemos de saber que podemos indicar a las columnas los tipos de datos que estas van a tener. **En el proceso de normalización de tablas no esta bien que una columna tenga diferentes tipos de datos**, es decir, en el caso de la columna id_contactos hemos indicado que sea del tipo INT(11) y esto significa que los datos que van a estar en esa columna van a ser numéricos sin decimales. Ejemplos de datos que se irán añadiendo a esa columna:

1
2
3
4

Esta columna no va a permitir que existan registros como por ejemplo estos:

uno
2.133
tres
4.22

Por estos motivos a las columnas se les especifica un tipo de dato. En el caso de **id_contactos** hemos de comentar que será la columna clave, será la columna que hace de indice y no se repetirán nunca los valores en sus registros.

Si fuésemos metiendo a mano el número identificativo de cada contacto posiblemente podríamos llegar a confundirnos y meter duplicados. Eso es un error que se evita especificando que esa columna será la que actuá como indice clave y que sus valores se añadirán automáticamente de forma incremental.

Para entender mejor como hemos creado la tabla y sus columnas y los motivos por los que lo hemos realizado así hemos de conocer los tipos de datos.

Hemos de pensar que definiendo los tipos de datos y ajustando la longitud de estos vamos a optimizar mucho mejor el tamaño de los datos de modo que es ahora cuando hemos de pensar un poco.

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

1. **Tipos numéricos (con coma flotante o no)**
2. **Tipos de Fecha**
3. **Tipos de Cadena**

1. Tipos numéricos:

Los tipos de datos numéricos se pueden dividir en dos grandes grupos, **los que están en coma flotante (con decimales) y los que no.**

TinyInt: es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

Bit ó Bool: un número entero que puede ser 0 ó 1

SmallInt: número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

MediumInt: número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.

Integer, Int: número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

BigInt: número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

Float: número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38. Existen algunos más pero no nos interesan. Quien quiera profundizar en esto puede buscar algún manual donde se indiquen todos.

TIPO DE DATO NÚMERICO	**TAMAÑO DE ALMACENAMIENTO**
TYNYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes

2. Tipos fecha

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

Date: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

DateTime: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

TimeStamp: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

Time: almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

Year: almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

TIPO DE DATO FECHA	**TAMAÑO DE ALMACENAMIENTO**
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

De todos modos la tabla no es correcta al 100%. En el caso de **TimeStamp** dependiendo del formato de la fecha (AñoMesDiaHoraMinutoSegundo) puede ocupar 14 bytes.

3. Tipos de cadena

Char(n): almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

VarChar(n): almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

TinyText y TinyBlob: Columna con una longitud máxima de 255 caracteres.

Blob y Text: un texto con un máximo de 65535 caracteres.

MediumBlob y MediumText: un texto con un máximo de 16.777.215 caracteres.

LongBlob y LongText: un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

Enum: campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

TIPO DE DATO FECHA	**TAMAÑO DE ALMACENAMIENTO**
CHAR(n)	n bytes
VARCHAR(n)	n + 1 bytes
TINYBLOB, TINYTEXT	Longitud + 1 bytes
BLOB, TEXT	Longitud + 2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud + 3 bytes
LONGBLOB, LONGTEXT	Longitud + 4 bytes
ENUM('valor1','valor2', …)	1 o 2 bytes dependiendo de los valores

"Aprendemos el 20% de lo que escuchamos, el 50% de lo que vemos, el 80% de lo que hacemos y el 95% de lo que eseñamos"

Desconocido/a

"Las manzanas no son peras, y una manzana y una pera no son dos manzanas, son dos peras"

Ana Botella