El SGBD MariaDB

Introducció i posada en marxa

MariaDB (https://mariadb.org) és un sistema gestor de base de dades molt utilitzat com a alternativa "clònica" (però lliure) al SGBD MySQL (https://www.mysql.com). Per instal·lar-lo només cal fer *sudo apt install mariadb-server* (a Ubuntu) o *sudo dnf install mariadb-server* (a Fedora)

> NOTA: Aquesta comanda instal.larà com a dependències, a més del servidor pròpiament dit, altres programes addicionals, com ara el client genèric oficial (la comanda *mariadb*) i d'altres de més especialitzades (les comandes *mariadb-dump*, *mariadb-admin*, *mariadb-upgrade*...), les qual anirem estudiant poc a poc. Si volguéssim instal·lar només (totes) aquestes aplicacions clients, caldria instal·lar llavors el paquet "mariadb-client" (a Ubuntu) o "mariadb" (a Fedora)

> NOTA: En realitat, el servidor MySQL també té una versió lliure, anomenada "Community Edition", que igualment es troba als repositoris oficials tant d'Ubuntu (amb el nom de "mysql-server") com de Fedora (amb el nom de "community-mysql-server") però no l'estudiarem

> NOTA: Per conéixer les diferències concretes entre MariaDB i MySQL, es pot consultar aquesta taula comparativa: https://mariadb.com/kb/en/mariadb-vs-mysql-features

Un cop instal·lat, posar en marxa el servidor MariaDB (tant a Ubuntu com a Fedora) és tan fàcil com amb qualsevol altre servei Systemd: *sudo systemctl start mariadb* Òbviament, també disposem de l'opció d'aturar-lo (amb *sudo systemctl stop mariadb*) així com d'habilitar-lo (amb *sudo systemctl enable mariadb)* o deshabilitar-lo (amb *sudo systemctl disable mariadb*) i, en tot moment, d'observar el seu estat (amb *systemctl status mariadb*), entre moltes altres opcions.

> NOTA: Per tenir més informació sobre com configurar Systemd per a què inicïi el servei MariaDB d'una manera més personalitzada (tot i que no ens caldrà filar tan prim), es pot consultar https://mariadb.com/kb/en/systemd

---

Tot i que no els modificarem gaire perquè ja estan prou ben establerts els valors de les directives que allà hi apareixen, cal conéixer quins són els fitxers de configuració tant del servidor com dels clients MariaDB i quina estructura tenen. A continuació els mencionarem (tot i que podeu conéixer més detalls a https://mariadb.com/kb/en/configuring-mariadb-with-option-files) :

> *A Ubuntu hi ha l'arxiu "**/etc/mysql/my.cnf**" (per configuracions globals) i "**~/.my.cnf**" (per configuracions particulars d'un usuari, caldria crear-lo en tot cas). El primer, però, no és més que un enllaç a "**/etc/mysql/mariadb.cnf**", el qual inclou al seu torn -i en aquest ordre- els arxius "**/etc/mysql/conf.d/*.cnf**" i "**/etc/mysql/mariadb.conf.d/*.cnf**" En el cas que hi hagi alguna directiva repetida en algun dels fitxers anteriors, es tindrà en compte només la darrera.
>
> *A Fedora hi ha l'arxiu "**/etc/my.cnf**" (per configuracions globals) i "**~/.my.cnf**" (per configuracions particulars d'un usuari, caldria crear-lo en tot cas). El contingut del primer, però, només serveix per incloure tots els arxius "**/etc/my.cnf.d/*.cnf**" (que sí que tenen contingut efectiu). En el cas que hi hagi alguna directiva repetida en algun dels fitxers anteriors, es tindrà en compte només la darrera.

En tot cas, als fitxers anteriors hi haurà diferents seccions que serveixen per classificar les directives indicades segons l'àmbit on afectin: si bé només al servidor (secció "[***server***]", tot i que també pot haver la secció "***[mysqld]***" o també la secció "***[mariadb]***"), només als clients (una secció per cada client, anomenada com ell i entre claudàtors o bé la secció genèrica "[***client***]", llegida per tots els clients) o a ambdós (secció "[***client-server***]"). Algunes de les directives més importants que hi podem trobar, a la secció de "només configuració del servidor" (és a dir, *"[server]"* o *"[mysqld]"* o *"[mariadb]"*, tant és), són:

* ***datadir*** : Ruta de la carpeta on s'emmagatzemaran físicament les bases de dades (cadascuna en una subcarpeta diferent). Per defecte, tant a Ubuntu com a Fedora, és "**/var/lib/mysql**"

* ***bind-address*** : Adreça IP per on escoltarà les peticions realitzades pels clients. Per defecte, el seu valor (i si no està indicat explícitament també) és "0.0.0.0" (o "*"), el que vol dir que per defecte el servidor MariaDB escoltarà per totes les adreces IP que tingui disponibles. No obstant, a la configuració del paquet oferit per Ubuntu aquesta directiva té assignada el valor "127.0.0.1", permetent, doncs, només connexions TCP/IP des de la pròpia màquina servidora. Si es vol deshabilitar completament l'accéss a la xarxa per part dels clients (fins i tot locals...veieu l'opció *socket* més avall) es pot indicar l'opció ***skip-networking*** (que no té cap valor associat)

* ***port*** : Port on escolta el servidor MariaDB les peticions fetes via TCP/IP. Per defecte, el seu valor (i si no està indicat explícitament també) és **3306**

* ***socket*** : El servidor MariaDB ofereix un mètode alternatiu als clients locals (és a dir, que siguin executats a la mateixa màquina on s'està executant el servidor) per connectar-s'hi diferent del canal TCP/IP. Aquest altre mètode es basa en un "socket" local que fa de nus d'intercomunicació entre els processos servidor i clients del sistema. Aquesta directiva indica la ruta d'aquest socket (que serà, doncs, la ruta que hauran de fer servir els clients per comunicar-se amb el servidor local). A Ubuntu el seu valor per defecte és "**/run/mysqld/mysqld.sock**" i a Fedora és "**/var/run/mysqld/mysql.sock**"

* ***log-error*** (o ***log_error***) : Ruta del fitxer on s'escriuran els missatges d'error generats perl servidor. Per defecte, a Ubuntu aquesta directiva està comentada (això farà que aquests missatges vagin en realitat directament al Journald), però, en tot cas, la ruta indicada és "**/var/log/mysql/error.log**"; a Fedora, en canvi, sí que està operativa i la ruta indicada és "**/var/log/mariadb/mariadb.log**" Consulta https://mariadb.com/kb/en/error-log per tenir més informació

> NOTA: Per tenir un llistat complet de totes les directives de configuració possibles, es pot consultar https://mariadb.com/kb/en/server-system-variables . Per saber el valor concret de cadascuna d'aquestes directives que el servidor MariaDB està implementant en un moment donat (més enllà de les explícitament definides als arxius de configuració, ja sigui perquè són els valors per defecte o bé han sigut modificat via paràmetres o variables d'entorn), executeu *mariadbd --verbose –help* (o també *sudo mariadb-admin variables*).

Durant la instal·lació estàndard del servidor MariaDB es creen dos usuaris interns del SGBD que seran els que d'entrada podrem fer servir des dels clients per connectar-nos-hi i començar a treballar (a partir d'ells podrem, si així ho volem, crear-ne d'altres usuaris -amb altres permisos diferents designats a la carta-). Aquests dos usuaris "per defecte" són ***root@localhost*** i ***mysql@localhost*** El primer és important perquè és l'usuari administrador del SGBD: amb ell es podran fer totes les tasques que necessitems sense cap restricció (crear/eliminar base de dades, afegir-hi/modificar-hi/eliminar-hi taules i llur contingut, etc). Cap d'ells, però, no té contrasenya vàlida per defecte perquè estan pensats per fer-se servir en el propi sistema local del servidor (utilitzant el sistema "socket") només des de l'usuari homònim del sistema ("root" o "mysql", respectivament). És a dir, si executem a la màquina local del servidor qualsevol client (la comanda *mariadb*, per exemple) com a "root" (o via *sudo*, que seria equivalent), podrem accedir al SGBD com a "root@localhost" (si així ho indiquem) directament sense haver d'especificar cap contrasenya. Igualment, una aplicació funcionant amb l'usuari del sistema anomenat "mysql" (creat durant la instal·lació estàndard del servidor i propietari de la carpeta "/var/lib/mysql") podrà connectar-se al SGBD directament (si així ho indiquem) com a "mysql@localhost"

> NOTA: La cua "@xxx" representa la màquina client des d'on es connecta l'usuari en qüestió. D'aquesta manera, es podrien assignar diferents permisos al mateix usuari depenent des d'on es connecti (en aquest cas en utilitzar la connexió via "socket" en lloc de TCP/IP no tindria gaire sentit indicar un altre valor diferent de "localhost", però en connexions per xarxa aquesta cua sí que pot ser interessant d'utilitzar

Si no volem utilitzar l'usuari "root@localhost" de MariaDB essent l'usuari "root" del sistema (és a dir, si no volem haver d'usar *sudo* amb els clients -o scripts PHP!- que utilitzem per contactar amb el SGBD), es pot assignar una contrasenya a aquest usuari "root@localhost". D'aquesta manera, especificant-la ja es podrà accedir al SGBD, independentment de l'usuari del sistema que siguem (i que hagi executat el client o script PHP indicat). La manera més senzilla d'especificar una contrasenya a l'usuari "root@localhost" és executar (amb el servidor MariaDB encès) l'script *sudo mariadb-secure-installation* En realitat, aquest script realitza una sèrie de preguntes interactives que serveixen per acabar de polir diferents detalls de la instal.lació del servidor i que, per tant, és molt recomanable executar el més aviat possible. Precisament, un d'aquests detalls és la possibilitat, responent que **NO** a la pregunta "Switch to unix_socket authentication?", d'assignar una nova contrasenya (o canviar-la, si ja en tingués una d'algun cop anterior) a l'usuari "root@localhost"

NOTA: Una altra opció que ofereix l'script anterior en forma de preguntes i que és molt important és la possibilitat de permetre o no l'accés de forma remota a l'usuari "root@localhost" (un cop se li ha assignat una contrasenya perquè si no en té, no en tindria ni la possibilitat); aquesta opció bàsicament consisteix en modificar l'opció *bind_address* del servidor. D'altra banda, altres possibilitats que ofereix l'script són la d'esborrar una base de dades anomenada "test" i també un usuari anònim de prova, el quals es creen per defecte en la instal.lació del servidor i que no farem servir per res. Al final de l'execució de l'script no caldrà reiniciar el servidor MariaDB...els canvis s'apliquen immediatament

NOTA: A continuació es desenvolupa una explicació més tècnica sobre els dos modes d'autenticació de l'usuari "root@localhost" (i també "mysql@localhost"): el mode sense contrasenya (també anomenat "unix_socket") i el mode amb contrasenya (també anomenat "mysql_native_password"). En resum, en crear-se durant la instal·lació del servidor MariaDB l'usuari "root@localhost" (i també "mysql@localhost"), mitjançant les següents ordres SQL internes...

CREATE USER root@localhost IDENTIFIED VIA unix_socket OR mysql_native_password USING 'invalid';
CREATE USER mysql@localhost IDENTIFIED VIA unix_socket OR mysql_native_password USING 'invalid';

... es dona la possibilitat a ambdós usuaris d'utilitzar dos "plugins" d'autenticació: el "unix_socket" i el "mysql_native_password", tot i que aquest darrer d'entrada no funciona perquè s'indica un hash de contrasenya no correcte (en concret la cadena 'invalid').És per això que s'ha d'indicar una contrasenya concreta a l'usuari "root@localhost" abans de poder utilitzar aquest mètode d'autenticació, el qual, cal tenir clar, ja estava disponible des del principi, gràcies a la partícula OR, com a "pla B" del mètode "unix_socket" si aquest no funcionés (perquè no s'executés el client com a usuari "root" del sistema o perquè directament el plugin "unix_socket" s'hagués deshabilitat per alguna raó -això es pot fer afegint la directiva ***unix_socket=OFF*** (o alternativament, ***disable_unix_socket***) sota la secció de servidor de l'arxiu de configuració del MariaDB-, etc)

NOTA: Si en un moment donat volguéssim canviar els mètodes d'autenticació de "root@localhost" per a què només s'usés un dels dos que coneixem (o d'altres que n'hi ha) i/o per directament canviar la contrasenya assignada a "root@localhost" usada concretament pel plugin "mysql_native_password", podríem emprar l'ordre SQL ALTER USER, així (un cop loguejats internament dins del SGBD com a "root@localhost", ja que és l'usuari amb permisos per fer-ho)...:

ALTER USER root@localhost IDENTIFIED VIA mysql_native_password;
SET PASSWORD FOR root@localhost = PASSWORD("contra");

...o tot en un: *ALTER USER root@localhost IDENTIFIED VIA mysql_native_password USING PASSWORD("contra");*

NOTA: Més enllà de l'usuari "root@localhost", si es vol crear un altre usuari intern del SGBD que s'autentiqui amb algun dels dos plugins anteriors (o amb qualsevol dels dos si un d'ells fallés, com passa amb "root@localhost"), caldria realitzar les següents ordres SQL internes (un cop loguejats internament dins del SGBD com a "root@localhost", ja que és l'usuari amb permisos per fer-ho). En el cas d'usar el plugin "unix_socket", cal tenir en compte que l'únic usuari del sistema que podria indicar llavors "usuari@localhost" en qualsevol client que executés per accedir al SGBD s'hauria d'anomenar igual; en aquest cas, "usuari".

*Per usar "unix_socket": *CREATE USER usuari@localhost IDENTIFIED VIA unix_socket;*
*Per usar contrasenya: *CREATE USER usuari@localhost IDENTIFIED VIA mysql_native_password;*
SET PASSWORD FOR usuari@localhost = PASSWORD("contra");

NOTA: Teniu més informació sobre els diferents plugins d'autenticació de MariaDB a la documentació oficial (https://mariadb.com/kb/en/authentication-plugins) i, concretament, sobre el "plugin" "unix_socket" aquí (https://mariadb.com/kb/en/authentication-plugin-unix-socket) i sobre el plugin "mysql_native_password" aquí (https://mariadb.com/kb/en/authentication-plugin-mysql_native_password). En tot cas, cal tenir clar que una cosa és el procés d'autenticació (és a dir, validar la teva identitat contra el SGBD) i una altra cosa (que en parlarem més endavant) és el mecanisme d'autorització (és a dir, comprovar que l'usuari autenticat tingui els permisos necessaris per fer una tasca).

Suposant que tenim un servidor MariaDB en marxa al qual li hem aplicat l'script *mariadb-secure-installation* per assignar una contrasenya a l'usuari "root@localhost", ja podrem accedir-hi per començar a treballar. Concretament, per connectar-nos al servidor MariaDB el client oficial que farem servir més sovint (ja sigui en la pròpia màquina servidora o bé, si així ho hem permès, des de alguna màquina remota) és la comanda *mariadb*, que és el client genèric que ens permetrà llençar de forma interactiva qualsevol sentència SQL contra el servidor per tal de crear/eliminar base de dades, crear/modificar/eliminar taules i llur contingut, etc. Així doncs, provarem d'executar la següent comanda: *mariadb -h IP.Serv.MariaDB -u root -pcontrasenya*, on:

* Si el client es troba a la mateixa màquina que el servidor, el paràmetre -h no caldrà especificar-lo

*El paràmetre *-u* serveix per indicar el nom de l'usuari intern amb el que ens autenticarem contra el SGBD. En el moment que creem altres usuaris més enllà de l'usuari "root@localhost", en aquest paràmetre llavors podrem indicar aquests altres noms d'usuaris. Com ja s'ha dit, però, depenent dels permisos que tingui assignats cada usuari emprat, en l'interior del SGBD es podran realitzar més o menys tasques

* Fixeu-vos que no hi ha espai entre el *-p* i la contrasenya. Es podria no escriure la contrasenya (és a dir, indicar el paràmetre *-p* sol); en aquest cas, la contrasenya es preguntaria de forma interactiva

Un cop loguejat l'usuari indicat contra el SGBD, apareixerà un "prompt" particular indicant que ja hi som "dins del servidor" i, per tant, podem començar a treballar amb ell executant les ordres SQL i comandes internes desitjades.

NOTA: Altres paràmetres interessants del client *mariadb* són:
-D nomBaseDades : Indica ja d'entrada la base de dades amb la qual es començarà a treballar. Si no s'indiqués, l'entorn interactiu no entra en cap base de dades, així que caldria fer-ho explícitament amb la comanda interna *USE nomBaseDades*
-e "ordre interna; una altra;" : Executa les ordres internes (pròpies o SQL) directament, sense haver d'entrar en la consola interactiva. Per exemple la comanda, *mariadb -u root -p -e "select @@version; select @@socket; select 1+1;"* retorna tres resultats: els valors de les variables internes del servidor "version" i "socket" (les quals representen la versió i ruta del "socket" local del servidor, respectivament) i el valor de la suma "1+1"
-P nº : Indica el port del servidor al que es conectarà (si el servidor escolta en un altre diferent de 3306)
--prompt="format": Estableix el texte que es veura`com a "prompt" del client en comptes del per defecte. A més de cadenes de caràcters estàndar, altres valors possibles (que es poden combinar entre sí) del format poden ser : \v (mostra la versió del servidor al que ens estem conectant), \d (la base de dades actualment utilitzada), \h (la ip del servidor al que ens estem conectats), \u (l'usuari amb qui hem conectat amb el servidor), \R (l'hora actual), \D (la data actual), \c (un contador per cada comanda que s'executa dins el client),etc.
-H : Fa que els resultats de les comandes s'escriguin dins de codi HTML (per poder-lo fer servir directament a pàgines web, normalment en forma de taules *<table>*). També és interessant el paràmetre *-s* , que fa que els resultats de les comandes s'escriguin tabulats, línia a línia (per poder-los usar a arxius CSV)
--protocol={TCP|SOCKET}: Tria explícitament el mètode de connexió amb el servidor. No obstant, en el cas d'emprar el paràmetre *-h* (sempre que no el seu valor no sigui "localhost"), el client triarà automàticament el mètode TCP sense que calgui indicar-ho. D'altra banda, en el cas d'emprar el mètode SOCKET, el paràmetre *-S /ruta/socket* pot ser necessari per indicar la ruta del "socket" a utilitzar per comunicar-se amb el servidor
En tot cas, teniu molta més informació a https://mariadb.com/kb/en/mysql-command-line-client

NOTA: L'historial de les comandes realitzades (SQL o no) es guarda per defecte a l'arxiu "~/.mysql_history".

Comandes bàsiques per començar

Algunes d'aquestes comandes (que provarem a l'entorn interactiu proporcionat per la comanda *mariadb*, ja sigui connectant-nos a un servidor MariaDB local o remot) formen part de l'estàndar SQL internacional (que actualment està en la seva versió 2016: https://en.wikipedia.org/wiki/SQL:2016) o bé de les ampliacions particulars afegides sobre aquest estàndard que MariaDB implementa (podeu veure la documentació més rellevant a https://mariadb.com/kb/en/sql-statements, https://mariadb.com/kb/en/sql-language-structure o també https://mariadb.com/kb/en/built-in-functions) mentre que altres són pròpies de l'entorn interactiu oferit per la comanda *mariadb* (https://mariadb.com/kb/en/mysql-command-line-client). D'entre aquestes darreres (les quals no cal que acabin amb el símbol ";", com sí passa amb les ordres SQL), en podem destacar les següents:

HELP: Mostra la llista de les comandes pròpies internes del shell (no SQL) que es poden utilitzar

HELP contents: Mostra un menú on es podrà accedir a l'ajuda sobre tota comanda SQL reconeguda
Si se sap ja d'entrada quina és la comanda SQL de la qual es vol obtenir informació, es pot escriure *HELP comanda SQL* (per exemple, així: *HELP show databases*)

STATUS : Mostra informació sobre la conexió actual (temps que porta oberta, nº de comandes realitzades, base de dades actualment activada, usuari actual utilitzat, versió del servidor, protocol de connexió emprat -TCL o "socket"-, "character set" emprat, si s'està utilitzant SSL o no, etc, etc).

USE nomBaseDades Aquesta comanda és imprescindible per començar a treballar amb la base de dades indicada (per consultant-hi o modificant-hi informació). En altres paraules, "activa" la BD. Òbviament, la base de dades en qüestió cal que existeixi; per crear-ne una nova veieu més endavant la comanda SQL *CREATE DATABASE nomBaseDades;*

EXIT o *QUIT*: Surt del shell del client *mariadb* i torna al terminal del sistema

NOTA: Altres comandes pròpies de l'entorn interactiu interessants són, per exemple:
PROMPT format (equivalent al paràmetre *–prompt="format"* de la comanda *mariadb*)
SOURCE /ruta/fitxer.sql (per interpretar el codi SQL escrit en el fitxer indicat en lloc de fer-ho interactivament)
SYSTEM comanda (per traspassar al shell */bin/sh* del client la comanda indicada i rebreń el resultat -i mostrar-lo-)
TEE /ruta/fitxer.txt (per guardar al fitxer indicat totes les comandes executades i els resultats mostrats a pantalla)

D'entre les comandes SQL (ja siguin pròpes de l'estàndard o particulars del producte MariaDB), en podem destacar les següents (totes aquestes comandes cal que acabin amb el símbol ";"):

SHOW DATABASES; : Mostra una llista de totes les bases de dades existents

NOTA:Veureu que, a més de les que hàgiu creat explícitament amb la comanda *CREATE DATABASE*, sempre hi apareixen a més tres bases de dades predefinides (¡i que no s'han d'esborrar mai perquè són fonamentals pel funcionament intern del servidor!). Concretament, són les següents (podeu obtenir més informació consultant https://mariadb.com/kb/en/system-tables): "**mysql**" (conté informació sobre els usuaris interns del SGBD i els seus permisos), "**performance_schema**"(conté informació sobre el rendiment del propi servidor, per monitoritzar-lo) i "**information_schema**" (conté informació variada sobre l'estructura de les bases de dades, taules, columnes, etc, a més de sobre les variables globals del servidor i les de sessió, sobre les característiques i funcionament del motor d'emmagatzematge utilitzat (per defecte, "InnoDB"), etc, etc; en general qualsevol comanda de tipus "show" obtindrà la informació requerida d'aquesta base de dades).

CREATE DATABASE nomBaseDades; : Crea una nova base de dades.

NOTA: Normalment, a aquesta comanda s'afegeix una indicació del "character set" a associar a la base de dades en qüestió (és a dir, del joc de caràcters que es vol utilitzar per codificar les dades de tipus cadena que s'hi emmagatzemin en totes les seves taules; el més comú és indicar el joc "*utf8mb4*" perquè és el més flexible i universal a l'hora de reconèixer caràcters d'altres idiomes diferents dels de l'anglès, ja que el per defecte, el joc "latin1", és més limitat), així com també una indicació del "collation" vinculat a aquest "character set" (és a dir, del sistema d'ordenació entre caràcters -majúscules vs minúscules, símbols, etc- que el servidor farà servir per comparar cadenes emmagatzemades en aquella base de dades; el més comú és indicar, pel joc "utf8mb4", el collation "*utf8mb4_general_ci*", tot i que no caldria perquè aquest ja és el "collation" per defecte associat a aquest joc concret). En tot cas, per indicar tant el joc com el "collation" desitjats es faria així: *CREATE DATABASE nomBaseDades* ***CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;*** En tot cas, per tenir més informació sobre aquests conceptes consulta https://mariadb.com/kb/en/character-sets i per implementar-los a la pràctica, https://mariadb.com/kb/en/setting-character-sets-and-collations

SHOW CREATE DATABASE nomBaseDades; : Mostra la comanda *CREATE DATABASE* exacta que al seu moment es va escriure corresponent a la base de dades indicada.

DROP DATABASE nomBaseDades; : Esborra la base de dades indicada

Creació de taules

Un cop haguem creat una base de dades, aquesta estarà buida, però. Per poder començar a introduir-hi dades, el que haurem de fer (¡després d'"ubicar-nos" a dins amb la sentència *USE nomBaseDades* !) és començar a crear les taules que necessitem, amb l'estructura i característiques desitjades, per allotjar les nostres dades. Per crear una taula, utilitzarem la sentència SQL *CREATE TABLE* de la següent forma:

```
CREATE TABLE nomTaula (nomCol1 tipusDada1 modifs,
                       nomCol2 tipusDada2 modifs,
                       ...
                       CONSTRAINT nomConst1 tipusConst,
                       CONSTRAINT nomConst2 tipusConst,
                       ...);
```

A la sentència anterior es pot veure com entre parèntesis, i separades per comes, s'ha d'indicar el nom de cada columna de la taula, el tipus de dada que aquesta contindrà i (de forma opcional) algun/s modificador/s per aquella columna; un cop indicades les característiques de totes les columnes, es pot veure també que s'han d'especificar, també separades per comes, les eventuals CONSTRAINTS que té la taula, indicant en aquest cas el nom que volguem donar a cadascuna i el seu tipus.

Tipus de dades més habituals (https://mariadb.com/kb/en/data-types)

Números sencers:

TINYINT : Ocupa 1 byte. Els valors possibles van de -2^7 a 2^7-1 (amb signe) o de 0 a 2^8-1 (sense signe)
SMALLINT : Ocupa 2 bytes.Els valors possibles van de -2^{15} a $2^{15}-1$ (amb signe) o de 0 a $2^{16}-1$ (sense signe)
MEDIUMINT: Ocupa 3 " . Els valors possibles van de -2^{23} a $2^{23}-1$ (amb signe) o de 0 a $2^{24}-1$ (sense signe)
INT : Ocupa 4 bytes. Els valors possibles van de -2^{31} a $2^{31}-1$ (amb signe) o de 0 a $2^{32}-1$ (sense signe)
BIGINT: Ocupa 8 bytes.Els valors possibles van de -2^{63} a $2^{63}-1$ (amb signe) o de 0 a $2^{64}-1$ (sense signe)

Números decimals:

DECIMAL(x,y) : 'x' indica el número de xifres totals que pot tenir el número i 'y' indica el número de xifres només decimals. Així, DECIMAL(5,2) pot contenir números entre -999.99 i 999.99.
FLOAT : Ocupa 4 bytes, on el número de xifres decimals pot variar.
DOUBLE : Ocupa 8 bytes, on el número de xifres decimals pot variar.

Cadenes:

CHAR(nº) : El nº indica la quantitat fixa de caràcters que s'emmagatzemaran (si el valor en té menys, es guardarà justificat a la dreta i la resta de caracters es guardaran en forma d'espais en blanc). El valor màxim de nº és 255; si es emmagatzemar més caràcters, s'ha d'utilitzar algun tipus de dades de la família TEXT.
VARCHAR(nº): El nº indica la quantitat màxima de caràcters que s'emmagatzemaran (però es poden emmagatzemar menys si el valor no arriba a aquest màxim). El valor màxim de nº, de totes maneres, és 255; si es emmagatzemar més caràcters, s'ha d'utilitzar algun tipus de dades de la família TEXT.

NOTA: La codificació de caracters associada als tipus de dades anteriors és la indicada en crear la base de dades corresponent

Temps:

DATE : Emmagatzema any, el mes i dia vàlid (l'entrada ha de ser una cadena com 'yyyy-mm-dd' o 'yyyymmdd' o un número com yyyymmdd). Un valor invàlid es guardarà com '0000-00-00'
TIME : Emmagatzema hora, minut i segon vàlid (l'entrada ha de ser una cadena com 'hh:mm:ss' -o 'hh:mm:ss.fraq' si s'usen fraccions de segon- ,'hhmmss' o un número com hhmmss). Un valor invàlid es guardarà com '00:00:00'. Opcionalment es pot indicar entre parentesis el número de decimals que es volen guardar com a fraccions de segon (1 seria només dècimes de segon, 2 centèsimes, etc fins a 6); si no s'indica, és equivalent a 0.

DATETIME : Emmagatzema data i hora vàlids (l'entrada ha de ser la cadena com 'yyyy-mm-dd hh:mm:ss' -o 'yyy-mm-dd hh:mm:ss.fraq' si s'usen fraccions de segon-, 'yyyymmddhhmmss' o un número com yyyymmddhhmmss). Un valor invàlid es guardarà com '0000-00-00 00:00:00'. Si no s'indica una part del valor (o la data o l'hora), aquesta part es guardarà també com zeros. Opcionalment es pot indicar entre parentesis el número de decimals que es volen guardar com a fraccions de segon (1 seria només dècimes de segon, 2 centèsimes, etc fins a 6); si no s'indica, és equivalent a 0.
TIMESTAMP : Emmagatzema una marca de temps en una cadena amb el mateix format que DATETIME. Una marca de temps és un valor que marca la data i hora actual; normalment aquests valors s'insereixen automàticament en accions de tipus INSERT o UPDATE a les columnes d'aquest tipus per tal de registrar el moment de la inserció o de l'actualització. Per tant, com que ja s'encarrega el servidor d'això, no cal indicar aquest valor manualment -i si es fa, cal indicar un valor NULL-. Cal tenir en compte, però, que per això funcioni aquest tipus de dades haurà de venir acompanyat del modificador **DEFAULT CURRENT_TIME** seguit de, opcionalment, **ON UPDATE CURRENT_TIME** si també es vol actualitzar el seu valor en les actualitzacions d'alguna columna del registre en qüestió. Opcionalment també es pot indicar entre parentesis el número de decimals que es volen guardar com a fraccions de segon (1 seria només dècimes de segon, 2 centèsimes, etc fins a 6); si no s'indica, és equivalent a 0.

NOTA: Aquests tipus de dades permet fer servir funcions del MariaDB especialitzades (com ara *DATA_ADD()* o *DATE_SUB()*, entre moltes altres) útils per manipular dates molt fàcilment.

Dades binàries:

TINYBLOB : Conté una dada binària (una foto, per exemple) que pot ocupar un màxim de 2^8-1 bytes
BLOB : Conté una dada binària que pot ocupar un màxim de 2^{16}-1 bytes
MEDIUMBLOB : Conté una dada binària que pot ocupar un màxim de 2^{24}-1 bytes
LONGBLOB : Conté una dada binària que pot ocupar un màxim de 2^{32}-1 bytes

Texts llargs:

TINYTEXT: Conté un text de com a màxim 2^8-1 caràcters (suposant que cada caràcter ocupa un byte)
TEXT : Conté un text de com a màxim 2^{16}-1 caràcters (suposant que cada caràcter ocupa un byte)
MEDIUMTEXT: Conté un text de com a màxim 2^{24}-1 caràcters (suposant que cada caràcter ocupa 1 byte)
LONGTEXT : Conté un text de com a màxim 2^{32}-1 caràcters (suposant que cada caràcter ocupa un byte)

NOTA: La codificació de caracters associada als tipus de dades anteriors és la indicada en crear la base de dades corresponent

Altres:

BIT: Emmagatzema un bit. Si volem emmagatzemar més (fins 64), entre parèntesis es pot indicar opcionalment la quantitat de bits desitjada, així BIT(nº). Els valors a guardar s'han d'indicar amb la notació b'xxx' (per exemple, b'10001' emmagatzemarà 5 bits amb el valor 1, 0,0,0 i 1 respectivament.
ENUM('valor1','valor2',...): Indica que la columna associada només podrà tenir un dels valors especificats a la llista (que s'indicaran com a cadenes, encara que a la llista també es pot especificar NULL i la cadena buida, ''). A més de pel seu propi valor, els elements de la llista es poden identificar també pel seu índex (1,2,3...); per exemple, es podria fer SELECT * FROM taula WHERE columnaEnum=2, on "2" indica el segon valor de la llista..
JSON : Emmagatzema dades de tipus JSON. Per saber com utilitzar amb més detall aquest tipus de dada complex, pots consultar https://mariadb.com/kb/en/json-data-type i https://mariadb.com/resources/blog/using-json-in-mariadb

Opcionalment, tal com ja hem dit, les columnes poden tenir determinades característiques (anomenats "modificadors") que, si s'indiquen, s'hauran d'especificar després del seu tipus -un rera l'altre i separats per espais en blanc si n'hi ha més d'un- i afectaran a totes les seves cel.les. Els més habituals són:

Modificadors de columnes

UNSIGNED: *Només afecta a columnes de tipus numèric (sencer o decimal)* **.** Serveix per indicar que el tipus de dades numèric és sense signe (per defecte és amb signe).

ZEROFILL: *Només afecta a columnes de tipus numèric (sencer o decimal)* **.** Serveix per indicar que es vol omplir de zeros (per defecte no es fa) totes les posicions per l'esquerra del número fins arribar al límit visual de representació de xifres establert. En el cas dels valors sencers, aquest límit s"estableix indicant entre parèntesis el número de xifres màximes que es vol representar així: INT(4), BIGINT(7), etc. i en el cas dels valors *float* i *double*, s'estableix indicant entre parèntesis el número de xifres màximes total i el número de xifres màximes decimals, així: FLOAT(7,4), per exemple.

AUTO_INCREMENT: *Només afecta a columnes de tipus numèric sencer.* Indica que la columna en qüestió conté un valor que s'assignarà automàticament a cada sentència INSERT (i, que, per tant, no caldrà especificar-ho a dita sentència o bé, si s'especifica, caldrà fer-ho amb el valor NULL) de forma seqüencial. Això vol dir que el primer INSERT que es realitzi sobre la taula que tingui una columna d'aquest tipus guardarà automàticament el valor 1 en ella, el segon INSERT guardarà el valor 2, el tercer el valor 3, etc. Aquest tipus de columnes serveixen per assignar un identificador únic a cada registre sense que calgui fer res explícitament. Es pot començar a partir d'un altre número diferent de 1 si s'indica així: AUTO_INCREMENT=5 -en aquest cas, començaria pel 5-. També es pot fer que l'autoincrement no vagi d'un en un sinó que sigui una altra quantitat: per fer això hem de manipular la variable del servidor *auto_increment_increment*, tal com explica el quadre següent

Per obtenir la llista completa de variables generals del servidor (i els seus valors actuals) simplement cal fer: **SHOW VARIABLES;**. Si es vol acotar a algunes variables indicant part del seu nom, es pot fer *SHOW VARIABLES LIKE '%partnom%';* on el símbol % indica "qualssevol caràcters". En el cas concret de la gestió del comportament de l'autoincrement, hi ha dues variables: *auto_increment_offset* (que serveix per indicar el valor inicial -per defecte és 1-) i *auto_increment_increment* (que serveix per indicar el "salt" entre valor i valor -per defecte és 1-). Així, fent **SHOW VARIABLES LIKE 'auto_inc%';** ja podríem veure els seus valors actuals. Per canviar-los simplement caldria executar la comanda *SET @nomvariable=nouvalor* (és a dir, en aquest cas, **SET @auto_increment_increment=5;** per exemple).

DEFAULT valor: Valor per defecte que tindrà la columna en qüestió si no s'indica cap valor explícitament en una sentència INSERT. Si el valor és de tipus cadena, ha d'indicar-se entre cometes i si es de tipus data ha de seguir els formats indicats al quadre anterior. A columnes de tipus BLOB, TEXT i JSON no es pot indicar valor per defecte.

UNIQUE: Indica que la columna ha de tenir obligatòriament un valor únic en cadascun dels registres existents a la taula. Es permet, però, el valor NULL (que pot estar repetit). Si es vol inserir, però, un registre no nul amb un valor repetit, l'operació donarà error.

NOT NULL: Indica que la columna no pot tenir cap valor NULL. Si es vol inserir un valor d'aquest tipus, l'operació donarà error.

Les "constraints" (o traduït, "restriccions") són característiques de la taula en sí que afecten al comportament de determinades columnes. S'indiquen, com ja hem vist, amb la sintaxi *CONSTRAINT nomConst tipusConst* després d'haver especificat totes les columnes de la taula a crear. Només estudiarem tres tipus de "constraints": PRIMARY KEY, FOREIGN KEY i CHECK, essent la primera la única obligatòria:

"Constraints" (restriccions)

CONSTRAINT nomPK PRIMARY KEY(nomCol): Combinació d'UNIQUE i NOT NULL; és a dir: indica que la columna (o el conjunt de columnes en global, si s'especifiqués més d'una entre els parèntesis separades per comes) ha de tenir obligatòriament un valor únic en cadascun dels registres existents a la taula i no pot tenir tampoc cap valor NULL. Per tant, si es vol inserir un registre amb un valor repetit o nul, l'operació donarà error. Aquesta "constraint" serveix per distingir sense equívoc un determinat registre de la taula respecte la resta de registres existents.

CONSTRAINT nomFK FOREIGN KEY(nomCol) REFERENCES unaAltraTaula(lasevaPK): Indica que la columna anomenada "nomCol" només podrà contenir valors que estiguin presents dins la columna que sigui la clau primària (amb nom "lasevaPK") d'una altra taula creada prèviament anomenada "unaAltraTaula". Aquesta "constraint" serveix per assegurar-se que la columna "nomCol" no pugui tenir valors incoherents respecte les dades que hi ha en altres taules (per exemple, evitant que la columna-clauforània "idclient" de la taula "Comandes" pugui tenir un id de client que no existeixi dins la columna-clauprimària "idclient" de la taula "Clients", per posar un exemple).

Una altra característica que podem controlar amb les claus foràmies és decidir què passa quan s'esborra o s'actualitza un registre d'"unaAltraTaula" que està vinculats a un o més registres de la taula que conté la clau foràmia. Per defecte, no es permet ni l'esborrament ni l'actualització (si s'intenta dóna error). Però hi ha una altra possibilitat escrivint (tot seguit després de *...references unaAltraTaula(lasevaPK)*) la següent clàusula: **ON DELETE CASCADE ON UPDATE CASCADE**; això farà que es pugui esborrar i actualitzar el registre d'"unaAltraTaula" esborrant/actualitzant al seu torn tots els registres vinculats de la taula que conté la clau foràmia. També existeix la línia **ON DELETE SET NULL ON UPDATE SET NULL**, la qual igualment esborra i actualitza el registre d'"unaAltraTaula" però estableix a NULL el valor de la columna que fa de clau foràmia en els registres vinculats (si no és de tipus NOT NULL: en aquest cas donaria error). En canvi, la línia **ON DELETE RESTRICT ON UPDATE RESTRICT** (o *ON DELETE NO ACTION ON UPDATE NO ACTION*, és el mateix) seria el valor que indica el comportament per defecte, ja descrit. Es pot indicar un comportament diferent per *ON DELETE* i per *ON UPDATE*, però no sol ser habitual.

CONSTRAINT nomCK CHECK (condicio) : Indica una determinada condició que ha de complir els valors de la columna o columnes que s'indiquin. Si s'executa un INSERT o un UPDATE que no compleixi alguna de les condicions definides, es generarà un error. Les condicions que es poden escriure poden ser bàsicament aquestes:

Si les columnes són numèriques	Si les columnes són de text
nomCol > nº **nomCol < nº** **nomCol = nº** **nomCol != nº** (diferent) **nomCol <> nº** (una altra manera de dir "diferent") **nomCol >= nº** **nomCol <= nº** **nomCol between nº and nº** **nomCol not between nº and nº** **nomCol in (nº, nº, nº...)** : el valor ha d'estar a la llista **nomCol not in (nº, nº, nº...)**	**nomCol like 'cadena'** Amb *like* es poden utilitzar els següents comodins: % per indicar qualssevol caràcters (incloent cap) _ per indicar un caràcter qualsevol [xyz] per indicar un caràcter de la llista [!yz] per indicar un caràcter diferent dels de la llista **nomCol rlike 'patró'** (o **nomCol regexp 'patró'**) Amb *rlike/regexp* es poden emprar expr. regulars. NOTA: Per saber més sobre com usar expressions regulars en sentències SQL de MariaDB, pots consultar https://mariadb.com/kb/en/regular-expressions-overview i, més general,https://mariadb.com/kb/en/regular-expressions-functions i https://mariadb.com/kb/en/pcre També existeixen les condicions **not like** i **not rlike/not regexp**. En qualsevol cas, totes aquestes condicions es gestionen de forma "case-insensitive" (és a dir, és igual si són minúscules o majúscules).

Es poden combinar diferents condicions amb els connectors **AND** i **OR** (i **XOR**). Per exemple, per fer que el nom d'un client només pugui començar per 'a' o per 'b' es podria fer una restricció CHECK d'aquest tipus: *CONSTRAINT inicialNom CHECK (nomClient like 'a%' or nom like 'b%')*

A continuació es presenta un exemple de comanda *CREATE TABLE* útil com a plantilla per altres que es puguin necessitar.

```
CREATE TABLE Client (
        id_client smallint unsigned auto_increment,
        data_alta timestamp,
        tipus_client enum("Premium","Standard","Poor"),
        nom_client varchar(20) not null,
        cp_client char(5) not null default "08921",
        constraint clientPK primary key(id_client)
);

CREATE TABLE Comanda (
        id_comanda smallint unsigned auto_increment,
        nom_producte1 varchar(20) not null,
        nom_producte2 varchar(20) default "cosa",
        data_comanda datetime,
        preu_comanda float unsigned,
        idclient smallint unsigned,
        constraint comandaPK primary key(id_comanda),
        constraint comandaFK foreign key(idclient) references Client(id_client),
        constraint comandaCK check (nom_producte1 like "%supreme%")
);
```

Altres sentències SQL relacionades amb les taules que convé conéixer són:

SHOW CREATE TABLE nomTaula; : Mostra la comanda *CREATE TABLE* exacta que al seu moment es va escriure per crear la taula indicada.

SHOW TABLES; : Llista les taules existents a la base de dades actual

SHOW COLUMNS FROM nomTaula; : Mostra les característiques (nom, tipus, modificadors i "constraints") de les columnes de la taula indicada. Es pot afinar més la informació escrivint *SHOW COLUMNS FROM nomTaula LIKE 'nomColumna';* . Un sinònim d'aquesta comanda és *DESCRIBE nomTaula;*

DROP TABLE nomTaula; : Esborra la taula indicada

ALTER TABLE nomTaula ... : Modifica l'estructura de la taula indicada (la qual ha d'estar creada ja) de formes diverses. A continuació es mostren els casos més comuns:

ALTER TABLE nomTaula ADD nomCol tipusDada modifs;	Afegeix una nova columna a la taula NOTA: Per defecte la columna s'afegeix al final de les que hi ha però després dels modificadors podem escriure FIRST si volem afegir la columna la primera de totes o AFTER nomAltraCol si volem afegir-la just rera la columna indicada.
ALTER TABLE nomTaula ADD CONSTRAINT nomConstr PRIMARY KEY(nomCol);	Afegeix una nova clau primària. En principi només pot haver una clau primària a cada taula.

ALTER TABLE nomTaula ADD CONSTRAINT nomConstr FOREIGN KEY(nomCol) REFERENCES altraTaula(saclau) ...;	Afegeix una nova clau forània
ALTER TABLE nomTaula ALTER nomCol SET DEFAULT valor;	Canvia el valor per defecte de la columna indicada. Per treure el valor per defecte seria ... *ALTER nomCol DROP DEFAULT*
ALTER TABLE nomTaula CHANGE nomColActual nomColNou tipusDadaNou;	Canvia el nom i el tipus de la columna (incloent modificadors) indicada
ALTER TABLE nomTaula CHANGE nomColActual nomColActual tipusDadaNou;	Canvia sols el tipus (i modif.) de la columna indicada
ALTER TABLE nomTaula CHANGE nomColActual nomColNou tipusDadaActual;	Canvia sols el nom de la columna indicada NOTA: Per defecte la columna no canvia de posició però si al final de la comanda CHANGE escrivim FIRST, es recolocarà la primera de totes i si escrivim AFTER *nomAltraCol* es recolocarà just rera la columna indicada.
ALTER TABLE nomTaula MODIFY nomCol1 tipusDadaNou;	Canvia el tipus de la columna indicada. En aquest sentit, és sinònim d'un dels possibles "modes" de la comanda CHANGE anterior. NOTA: Per defecte la columna no canvia de posició però si al final de la comanda MODIFY escrivim FIRST, es recolocarà la primera de totes i si escrivim AFTER nomAltraCol es recolocarà just rera la columna indicada.
ALTER TABLE nomTaula DROP nomCol;	Elimina la columna indicada de la taula
ALTER TABLE nomTaula DROP PRIMARY KEY;	Elimina la clau primària de la taula
ALTER TABLE nomTaula DROP FOREIGN KEY nomConst;	Elimina la clau forània indicada
ALTER TABLE nomTaula RENAME TO nouNomTaula;	Renombra la taula. Una comanda equivalent és *RENAME TABLE nomTaula TO nouNomTaula;*

Inserció de dades

Un cop creades les taules, el primer que hem de fer és introduir les dades en elles. La manera clàssica de fer-ho és amb la comanda SQL *INSERT*:

INSERT INTO nomTaula(nomCol1,nomCol2) VALUES (--,--), (--,--), (--,--); **:** Insereix a "nomTaula" els registres indicats (en aquest exemple serien tres registres formats per dues columnes). Els valors de cadena (així com els de tipus data si s'escau) aniran escrits entre cometes -dobles o simples, és igual- i els de tipus numèric no; el separador decimal és el punt (.) i el valor NULL és un valor vàlid. No caldria indicar el nom de les columnes (rera el nom de la taula) si s'insereixen explícitament valors a totes les columnes existents a la taula però en el cas de tenir alguna columna de tipus AUTO_INCREMENT o TIMESTAMP, o voler inserir els valors per defecte que estiguin predefinits, llavors sí que caldrà indicar el nom de les columnes a les què sí inserirem valors explícitament).

NOTA: En el cas dels valors per defecte, una sintaxi alternativa és no indicar el nom de les columnes però llavors escriure la paraula clau *DEFAULT* en el lloc del valor a inserir de la columna en qüestió

NOTA: La comanda *INSERT* té una altra sintaxi alternativa que permet seleccionar registres formats per columnes concretes d'una taula i inserir-los en una altra, així: *INSERT INTO nomTaula(nomCol1,nomCol2) SELECT unaCol,unaAltraCol FROM unaAltraTaula* Molt útil per copiar una taula en una altra, per exemple

Cal saber també que existeix una altra ordre diferent d'INSERT anomenada *REPLACE* la qual funciona igual (fins i tot admet les dues sintaxis, la clàssica i la mencionada al paràgraf anterior) excepte en què si un registre ja existent a la taula té el mateix valor de clau primària que el registre que s'hi vol afegir, en comptes d'ocòrrer un error (com passaria amb l'INSERT) el registre antic s'esborra i el nou el substitueix.

La comanda *INSERT* no és gaire pràctica si es vol realitzar importacions massive de dades, més si aquestes estan prèviament guardades en algun fitxer (cas habitual quan aquest fitxer és el resultat d'algun procés d'obtenció previ; de fet, és molt freqüent tenir un arxiu CSV amb dades obtingudes a partir d'algun determinat programa com ara un full de càlcul que voldrem importar dins d'una taula de la nostra base de dades). En aquest cas, és més cómode, en lloc de la sentència *INSERT*, fer servir la sentència SQL següent:

LOAD DATA LOCAL INFILE "/ruta/arxiu.csv" INTO TABLE nomTaula;

NOTA: Atenció, aquesta sentència, tot i que comú, no és estàndard; per exemple, a PostgreSQL no existeix (la seva alternativa seria la sentència *COPY*)

La comanda anterior llegeix l'arxiu indicat (ubicat, ¡atenció!, a una carpeta de la màquina client; si no haguéssim especificat la paraula "LOCAL" la ruta llavors sí que s'interpretaria des de la màquina servidor, però per poder fer això sense problemes l'usuari amb què ens connectem al servidor caldria que tingués el permís especial "FILE") tenint en compte que cada línia representa un registre de la taula, a cada línia les dades estan escrites segons l'ordre de les columnes especificat al CREATE TABLE pertinent i estan separades per tabulador (els valors nuls s'indiquen amb el símbol "\N", sense cometes).

Si el caràcter separador és diferent del tabulador (per exemple suposarem que és la coma) es pot indicar afegint després del *...INTO TABLE nomTaula* la clàusula *FIELDS TERMINATED BY ","*. També es pot indicar allà mateix si els valors estan envoltats d'algun caràcter concret (per exemple suposarem les cometes simples) amb *FIELDS ENCLOSED BY "'"*. Igualment, també podem indicar el caràcter separador de les línies (això és important en el cas de fitxers de text creats a Windows, on aquest caràcter és "\r\n") amb la clàusula *LINES TERMINATED BY "\r\n"* i un eventual caràcter de començament de línia amb *LINES STARTING BY "'"*. Si no s'indica res, seria equivalent a escriure *FIELDS TERMINATED BY '\t' ENCLOSED BY '' LINES TERMINATED BY '\n' STARTING BY ''*

Abans del *... INTO TABLE nomTaula* es pot afegir la paraula *REPLACE* per tal de sobrescriure els possibles registres que tinguessin el mateix valor de clau primària. Per defecte, si passa això els valors llegits del fitxer s'ignoren (i per tant, no es realitza cap modificació en aquell registre de la taula).

També es pot afegir (al final de tot de la comanda) la clàusula *IGNORE nº LINES* per obviar el número de línies indicat al principi del fitxer (per exemple, es podria escriure *IGNORE 1 LINES* per saltar-se la primera línia que se suposa que podria contenir el títol de les columnes).

Després de la sentència anterior, encara es pot afegir una darrera clàusula que consistiria simplement en una llista (entre parèntesis i separades per comes) de les columnes concretes de la taula en qüestió a les que s'insertarien les dades del fitxer (si no s'especifica cap llista, s'insertaran a totes les columnes de la taula). Si una línia del fitxer tingués més camps que columnes té la taula, els camps sobrants s'ignoren; si passés al contrari, les columnes que no tinguessin camp associat s'establirien al seu valor per defecte.

D'altra banda, amb el servidor MariaDB es distribueix un executable anomenat *mariadb-import* que realitza la mateixa tasca que LOAD DATA però directament des del terminal del sistema operatiu. La seva referència es pot trobar a https://mariadb.com/kb/en/mysqlimport però bàsicament cal indicar com a paràmetres l'usuari (*-u*), la contrasenya (*-p*), els possibles indicadors de format necessaris (*--fields-terminated-by*, *--fields-enclosed-by*, *--replace*, *--ignore-lines*, etc) i, a continuació, el nom de la base de dades de treball seguit finalment de la ruta del fitxer a importar

NOTA: Teniu més informació sobre les diferents maneres de realitzar processos d'importació de dades a l'article https://mariadb.com/kb/en/importing-data-into-mariadb

Consulta de dades

La comanda SELECT és la que es fa servir per realitzar consultes. És una comanda molt versàtil, flexible i a vegades complexa: es necessitaria un trimestre sencer per conèixer totes les seves possibilitats. A continuació només es veuran els casos més habituals d'ús:

SELECT nomCol1,nomCol2 FROM nomTaula; : Mostra, per tots els registres, els valors de les columnes especificades de la taula indicada. Es pot utilitzar l'asterisc (*) per indicar "totes les columnes"; així, la comanda *SELECT * FROM nomTaula;* mostrarà, per tots els registres, els valors de totes les columnes de la taula indicada.

Si no es volen obtenir tots els registres, es poden fer filtres amb la clàusula *WHERE condicio*. D'aquesta manera, no es mostraran totes les "files" sinò només les que compleixin una determinada condició. Per exemple, ***SELECT * FROM mascotes WHERE numPotes > 3;*** mostrarà els valors de totes les columnes de la taula "mascotes" però només en el cas dels registres que tinguin el valor de la columna "numPotes" més gran que 3. En aquest sentit, els camps de tipus data es poden manipular com camps numèrics; per exemple, la consulta ***SELECT * FROM mascotes WHERE data_naix > "2015-1-1";*** mostrarà totes les dades de les mascotes que hagin nascut després de l'1 de gener del 2015. També es poden combinar diferents condicions dins de la -única- clàusula WHERE; per exemple, ***SELECT nommascota FROM mascotes WHERE especie LIKE "gos" AND color NOT LIKE "marró";*** mostraria el nom de només les mascotes que siguin gossos de color diferent al marró.

De fet, les condicions que es poden escriure a la clàusula *WHERE* són les mateixes que les admeses a la "constraint" *CHECK* (explicada en un apartat anterior: *<,>,=,<>, BETWEEN, IN, LIKE, RLIKE*, etc).Per exemple, si es vol obtenir totes els noms de mascotes que comencin per "b" (o "B") es podria escriure ***SELECT * FROM mascotes WHERE nommascota LIKE "b%";*** si a més volguéssim que aquest nom només pugui tenir quatre lletres, podríem fer llavors ***SELECT * FROM mascotes WHERE nommascota LIKE "b___";***. Si, en canvi, el que es vol és obtenir les mascotes que no tinguin ni el color marró, ni el negre ni el blanc, es podria fer ***SELECT * FROM mascotes WHERE color NOT IN ("marró","negre","blanc");***

NOTA: Hi ha, no obstant, una condició exclusiva de la clàusula WHERE, que és: *nomCol IS NULL* (o també *nomCol IS NOT NULL*), ja que el valor NULL no es pot comparar amb els operadors comuns =,<,>,<>,etc.

Altres clàusules interessants que es poden afegir en diferents llocs dins d'una consulta SELECT són les següents:

DISTINCT : Només mostra un únic resultat de tots els que puguin haver repetits. Per exemple: ***SELECT DISTINCT nompropietari FROM mascotes;***

AS nomNou: Serveix per indicar un altre títol de columna a mostrar, en comptes del títol real de la columna. Per exemple, la sentència ***SELECT nompropietari AS amo FROM mascotes;*** mostraria el títol "amo" en comptes del real ("nompropietari"). Aquesta clàusula és útil sobretot quan el resultat a mostrar no directament el contingut en una columna sinò s'ha obtingut a partir d'un determinat càlcul (en veurem exemples més endavant).

ORDER BY nomCol1, nomCol2...: Ordena un resultat pels valors d'una columna (o més). Per exemple: ***SELECT nom, data_naix FROM mascotes ORDER BY data_naix;*** Per ordenar de forma descendent -o de Z a A en el cas de camps de text (per defecte l'ordenació sempre és ascendent i de A a Z) s'ha d'especificar la paraula *DESC*, així*:* ***SELECT nom, data_naix FROM mascotes ORDER BY data_naix DESC;*** L'exemple següent ordena primer (de manera ascencent) per espècies, i seguidament, (i de manera descendent) per dates: ***SELECT nom, especie, data_naix FROM mascotes ORDER BY especie, data_naix DESC;.***

COUNT(*) : Mostra (com si fos el resultat d'una columna més) el número de registres que compleixen la condició indicada a la clàusula *WHERE*. Per exemple, ***SELECT COUNT(*) FROM mascotes WHERE color="marró";*** mostrarà el número de mascotes de color marró. Si no hi ha clàusula *WHERE*, el recompte es fa de tots els registres ("files") de la taula.

GROUP BY nomCol1, nomCol2...: Agrupa els registres segons el valor d'una determinada columna. D'aquesta manera, es poden obtenir dades "estadístiques" diferenciades de cada grup. Per exemple, si es vol comptar el número de mascotes que té cada propietari, es podria fer ***SELECT COUNT(*), nompropietari FROM mascotes GROUP BY nompropietari;*** : el que es mostrarà serà el recompte de files per cada grup de registres existent (els quals estan definits segons cada propietari diferent). Si només volem comptar el número de mascotes de cada espècie per separat que tinguin el color marró, hauríem de fer ***SELECT COUNT(*), especie FROM mascotes WHERE color="marró" GROUP BY especie;***

NOTA: A la clàusula *GROUP BY* es poden indicar més d'una columna per tal de fer grups més específics. Per exemple, *GROUP BY nompropietari,especie;* faria grups diferenciats per mascotes de la mateixa espècie que fossin del mateix propietari.

Treballar amb grups

A més de comptar registres, amb els grups definits per *GROUP BY* podem fer altres tasques estadístiques interessants (també es podrien fer per tots els registres d'una taula directament -sense fer grups, per tant- però no és tan habitual). Per exemple:

AVG(nomCol) : Calcula la mitjana dels valors -numèrics, s'entén- de la columna indicada (per cada grup per separat si a la sentència *SELECT* hi ha la clàusula *GROUP BY*)
SUM(nomCol) : Calcula la suma dels valors -numèrics, s'entén- de la columna indicada (per cada grup per separat si a la sentència *SELECT* hi ha la clàusula *GROUP BY*)
MIN(nomCol) : Calcula el mínim dels valors -numèrics, s'entén- de la columna indicada (per cada grup per separat si a la sentència *SELECT* hi ha la clàusula *GROUP BY*)
MAX(nomCol): Calcula el màxim dels valors -numèrics, s'entén- de la columna indicada (per cada grup per separat si a la sentència *SELECT* hi ha la clàusula *GROUP BY*)

HAVING condicio: Aquesta clàusula sempre acompanya a *GROUP BY* per descartar, un cop fet els grups, els que no compleixin una determinada condició (la qual segueix les mateixes normes que les d'una clàusula *WHERE* però a més poden fer servir les funcions *COUNT(), AVG(),SUM(), MIN()* o MAX(), cosa que a *WHERE* no es pot fer). Per exemple, ***SELECT COUNT(*), nompropietari FROM mascotes GROUP BY nompropietari HAVING especie="gos";*** només mostra el recompte de mascotes pels propietaris que tinguin gossos.

LIMIT nº: Indica el número de files (màxim) que es volen mostrar, encara que la consulta *SELECT* n'obtingui més. Per exemple ***SELECT * FROM mascotes LIMIT 3;*** només mostrarà tres registres encara que a la taula hi hagin més. Aquesta clàusula és útil combinada amb *ORDER BY* per obtenir els "x primers" registres ordenats segons un determinat criteri.

NOTA: Encara que no molt utilitzada, existeix també la clàusula *LIMIT nº OFFSET nº* per mostrar el número de registres indicats però començant a partir d'una posició concreta en comptes de pel primer registre obtingut.

INTO OUTFILE '/ruta/Fitxer' : Permet exportar les dades obtingudes de la consulta a un fitxer (que no ha d'existir prèviament, per seguretat) que serà guardat a la màquina servidor (per tant, per a què aquesta sentència funcioni caldrà que l'usuari executor tingui el permís "FILE"). Per defecte cada registre es guardarà en una línia i els camps estaran separats per tabuladors. Si es vol canviar això, es poden fer servir les clàusules *LINES TERMINATED BY* i *FIELDS TERMINATED BY* i/o *ENCLOSED BY*, entre d'altres (consultar quadre anterior sobre la comanda LOAD FILE per més informació).

Moltes vegades en una consulta intervenen camps que pertanyen a diferents taules (relacionades entre sí mitjançant claus forànies). Per exemple, si suposem que tenim una taula anomenada "propietaris" amb el camp "nompropietari" que conté un identificador fent de clau primària (el qual aquí representa que és el seu nom complet però seria millor que fos, per exemple, el seu DNI) i el camp "direcció" que conté la seva adreça, i una altra taula anomenada "mascotes" amb el camp "nompropietari" fent de clau foràni del camp homònim de la taula anterior i el camp "nommascota" que conté el nom de la mascota, per obtenir els noms de les mascotes del propietari que visqui a una adreça donada (per exemple, "C/Verdi") només caldria realitzar la següent consulta: ***SELECT mascotes.nommascota FROM mascotes, propietaris WHERE mascotes.nompropietari=propietaris.nompropietari AND propietaris.direccio LIKE "C/Verdi";***. Fixem-nos que a més d'indicar rera el FROM totes les taules implicades i com a prefixe de cada columna la taula a la què pertany, el fet important radica en la condició *mascotes.nompropietari=propietaris.nompropietari* de la clàusula *WHERE*, ja que és la responsable de vincular la clau foràni d'una taula amb la clau primària de l'altra de manera que poguem accedir als camps d'ambdues taules (en aquest cas, per obtenir "mascotes.nom" i per utilitzar com a filtre "propietaris.direccio") mantenint la integritat i coherència de les dades. Aquest tipus de consulta on s'utilitzen camps de diferents taules vinculades és el que tècnicament es diu "JOIN" (o més específicament, "INNER JOIN", perquè n'hi ha d'altres tipus de JOINs dels quals no en parlarem).

Una altra possibilitat que no estudiarem a fons és la de realitzar subconsultes. És a dir: aprofitar el resultat d'una consulta (escrita entre parèntesis) fent-lo servir com filtre per realitzar una altra consulta. Per exemple, ***SELECT nommascota FROM mascotes WHERE nompropietari IN (SELECT nompropietari FROM propietaris WHERE direccio="C/Verdi");*** serviria, en aquest cas, per obtenir les mateixes dades que el "JOIN" descrit al paràgraf anterior. Un altre exemple (sense gaire sentit) és el següent: ***SELECT nommascota FROM mascotes WHERE numPotes > (SELECT COUNT(*) FROM propietaris);*** , el qual obté el nom de les mascotes que tenen més potes que el número total de propietaris que tinguem a la base de dades.

D'altra banda, el servidor MySQL ofereix determinades funcions que permeten processar el contingut de columnes amb un determinat tipus de dades per tal de generar, a partir d'aquest processament, un resultat diferent a l'originalment obtingut i/o també per tal de modificar la seva presentació. A continuació presentem diferents casos:

Treballar amb dates

Els tipus de dada DATE, TIME, DATETIME i TIMESTAMP permeten ser manipulats amb funcions específiques (generalment sempre en sentències SELECT) que realitzen càlculs molt flexibles i versàtils. Alguns exemples (s'entén que els valors indicats com a paràmetre es poden haver escrit directament o bé provenir d'alguna columna d'alguna taula) són:

YEAR("data") : Extreu només l'any de la data indicada, en forma de número
MONTH("data") : Extreu només el mes de la data indicada, en forma de número
DAY("data"):Extreu només el dia de la data indicada, en forma de número
HOUR("data"): Extreu només l'hora de la data indicada, en forma de número
MINUTE("data"): Extreu només el minut de la data indicada, en forma de número
SECOND("data"): Extreu només el segon de la data indicada, en forma de número
DATE("data"): Extreu només la part de la data d'una dada DATETIME o TIMESTAMP. També està **TIME("data")**, que extreu només la part de l'hora d'una dada DATETIME o TIMESTAMP.

NOW() o **CURRENT_TIMESTAMP()** o **CURRENT_TIMESTAMP:** Retorna la data i hora actual (en forma de cadena). També està **CURDATE()** o **CURRENT_DATE()** o **CURRENT_DATE**, que retorna només la data actual (igualment en forma de cadena) i **CURTIME()** o **CURRENT_TIME()** o **CURRENT_TIME**, que retorna només l'hora actual, (també en forma de cadena)

DATEDIFF("data1","data2") : Retorna el número de dies entre la primera i la segona data. També està **TIMEDIFF("data1","data2")**, que retorna les hores,minuts i segons entre la 1ª i la 2ª data.

ADDDATE("data", INTERVAL nº unitat) : Afegeix a la data indicada com a primer paràmetre el nº de la unitat de temps (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND...) indicada com a 2n paràmetre (INTERVAL és una paraula clau). Existeixen més opcions per indicar la unitat de temps: per més informació pots consultar la pàgina http://www.w3resource.com/mysql/date-and-time-functions/mysql-adddate-function.php . També està **ADDTIME("data","hora")** , que afegeix a la data indicada com a primer paràmetre l'hora indicada com a segon (en format "hh:mm:ss" o inclús "d hh:mm:ss") i retorna la data resultant

SUBDATE("data", INTERVAL nº unitat): Resta a la data indicada com a primer paràmetre el nº de la unitat de temps (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND...) indicada com a 2n paràmetre. Existeixen més opcions a l'hora d'indicar la unitat de temps: per més informació consultar http://www.w3resource.com/mysql/date-and-time-functions/mysql-adddate-function.php També està **SUBTIME("data","hora")** , que resta a la data indicada com a primer paràmetre l'hora indicada com a segon (en format "hh:mm:ss" o inclús "d hh:mm:ss") i retorna la data resultant

DATE_FORMAT("data", "cadena") : Retorna la data escrita dins d'una cadena que pot contenir els següents símbols especials %Y (any), %m (mes), %d (dia de mes), %H (hora), %i (minuts), %s (segons), entre molts d'altres. També existeix TIME_FORMAT(), la qual funciona igual però només per dades de tipus TIME. Per veure'n tots, consultar http://www.w3resource.com/mysql/date-and-time-functions/mysql-date_format-function.php .

Un exemple: per saber les mascotes que tenen el seu aniversari el més que ve, la sentència que podríem escriure és: *SELECT nom, data_naix FROM mascotes WHERE MONTH(data_naix) = MONTH(DATE_ADD(CURDATE(), INTERVAL 1 MONTH));*

Treballar amb cadenes

Algunes de les funcions més habituals relacionades amb el tractament de cadenes són:

UCASE("cadena"): Retorna la mateixa cadena indicada com a paràmetre però transformada en tot majúscules. També està **LCASE("cadena")**, que retorna la mateixa cadena indicada com a paràmetre però transformada en tot minúscules.

CHAR_LENGTH("cadena"): Retorna el nº de caracters de la cadena indicada. Depenent de la codificació empreda, un caracter podrà ocupar un byte o més...per saber quant ocupa (en bytes) una determinada cadena cal executar **BIT_LENGTH("cadena").**

RPAD("cadena",nº,"caracter") : Retorna una cadena formada per la cadena indicada com a primer paràmetre més un número determinat de repeticions (escrites a la dreta d'aquesta) del caràcter indicat com a tercer paràmetre, fins arribar a un número total de caràcters indicat com a segon paràmetre. També està **LPAD("cadena, nº, "caracter"),** que fa el mateix però el caràcter repetit s'escriu a l'esquerra de "cadena".

CONCAT("cadena1","cadena2",...): Retorna una cadena formada per les cadenes indicades com a paràmetre concatenades una rera l'altra. Existeix també la funció **CONCAT_WS()**, la qual fa el mateix però el seu primer paràmetre ha de ser un caràcter separador que volguem definir per afegir a la cadena-resultat entre cadena i cadena.

TRIM("cadena") : Retorna la mateixa cadena indicada com a paràmetre però sense els possibles espais en blanc i/o tabuladors que pogués tenir tant al seu començament com al seu final. També està **RTRIM("cadena"),** la qual només elimina els espais/tabuladors que hi ha al final de la cadena i **LTRIM("cadena")** que només elimina els espais/tabuladors que hi ha al començament.

RIGHT("cadena",nº): Retorna una subcadena amb el nº de caracters indicat com a segon paràmetre de la cadena indicada com a primer paràmetre, començant per la dreta. També està **LEFT("cadena", nº)**, que retorna una subcadena amb el nº de caracters indicat com a segon paràmetre de la cadena indicada com a primer paràmetre, començant per l'esquerra.

LOCATE("subcadena", "cadena"): Retorna la posició on es troba per primera vegada el primer caràcter de "subcadena" dins de "cadena".Opcionalment, aquesta funció pot tenir un tercer paràmetre que indica la posició a partir de la qual es vol començar a fer la búsqueda (per defecte és igual a 1).

MID("cadena", nº pos, nº carac): Retorna una subcadena amb un número de caràcters indicat com a tercer paràmetre obtinguda a partir de la posició indicada com a segon paràmetre dins de la cadena passada com a primer paràmetre. Per exemple, *MID("barret",2,3);* retornaria "arr". Sinònims són *SUBSTRING()* i *SUBSTR().*

REPLACE("cadena","subcadenaABuscar","subcadenaAEscriure"): Retorna una cadena similar a "cadena" però on s'ha substituït totes les ocurrència que hi poguessin haver de "subcadenaABuscar" per "subcadenaAEscriure".

INSERT("cadena", nº pos, nº carac elim, "cadenaAEscriure"): Retorna una nova cadena formada per l'eliminació dins de "cadena" del número de caràcters indicats com a tercer paràmetre a partir de la posició indicada com a segon paràmetre (les posicions comencen per 1) i la posterior inserció a partir d'aquesta mateixa posició de la cadena "cadenaAEscriure". Per exemple, *INSERT("w3resource",3,2,"web")* retornaria "w3websource".

REVERSE("cadena"): Retorna la cadena indicada com a paràmetre al revés.

REPEAT("cadena", nº): Retorna una cadena formada per "cadena" repetida nº cops.

SPACE(nº): Genera una cadena formada per la quantitat indicada com a paràmetre d'espais en blanc.

Treballar amb números

La sentència SELECT es pot fer servir per realitzar càlculs matemàtics, com ara *SELECT (4+1)*5;* o bé *SELECT (nomCol1/nomCol2)-5 FROM nomTaula;* (suposant que ambdues columnes són de tipus numèric). A més a més, podem fer servir diferents funcions matemàtiques com per exemple les següents (entre moltes més):

SIN(nº): Retorna el sinus del número indicat (que representa un valor en radians). També està **COS(nº)**, que retorna el cosinus o **TAN(nº)**, que retorna la tangent, entre d'altres funcions trigonomètriques

EXP(nº): Retorna el número $e^{nº}$ També està **LOG(nº)**, que retorna el logaritme en base *e* (és a dir, el logaritme "natural") de nº (un sinònim és **LN(nº)**). També està **LOG10(nº),** que retorna el logaritme en base 10 i **LOG2(nº),** en base 2.

POW(x,y) : Retorna el número x^y .També està **SQRT(nº)**, que retorna l'arrel quadrada del número indicat

MOD(x,y): Retorna el remanent ("el mòdul") de dividir *x* entre *y* . També es podria utilitzar l'operador % per indicar aquest resultat, així: *x % y.*

ABS(nº): Retorna el valor absolut del número indicat com a paràmetre. També està **SIGN(nº)**, la qual retorna 1 si el número indicat com a paràmetre és positiu, -1 si és negatiu i 0 si és 0.

PI(): Retorna el número PI.

RAND() : Retorna un número decimal (pseudo)aleatori entre 0 i 1. Opcionalment, es pot indicar un número sencer com a paràmetre;en aquest cas, aquest valor s'utilitzaria com a "llavor" per generar el número aleatori (una mateixa llavor genera la mateixa seqüència de números aleatoris)

GREATEST(nº,nº,nº...) : Retorna el número més gran dels de la llista indicada. També està **LEAST(nº,nº,nº...)**, que retorna el número més petit dels de la llista indicada

ROUND(nº, nºdec): Retorna el mateix número (decimal) indicat com a primer paràmetre però arrodonit amb el número de decimals indicat com a segon paràmetre. També està **FORMAT(nº, nºdec)**, que **r**etorna una *cadena* que representa el número (decimal) indicat com a primer paràmetre amb el número de decimals indicat com a segon paràmetre (arrodonint si cal). També està **TRUNCATE(nº, nº dec),** que retorna el número truncat en comptes d'arrodonit. D'altra banda, també trobem **CEIL(nº)**, la qual retorna el número sencer més petit major que el número (decimal) indicat com a paràmetre i **FLOOR(nº)**, la qual retorna el número sencer més gran no major que el número (decimal) indicat com a paràmetre.

Treballar amb metadades del servidor

USER() : Retorna l'usuari actualment utilitzat
DATABASE(): Retorna la base de dades actualment activa
VERSION(): Retorna la versió del servidor al que s'està connectat

Altres funcions molt interessant són *IFNULL(nomCol, valor)*, la qual retorna el valor de "nomCol" sempre i quan aquest no sigui NULL (i si ho és, llavors retorna el valor indicat com a segon paràmetre); *COALESCE(valor1,valor2,...)*, la qual retorna el primer valor de la llista indicada que no és NULL (o NULL si tots ho són) i *IF(expr,valor1,valor2)*, la qual comprova si "expr" és certa, diferent de 0 o diferent de NULL: en el cas que sigui així retorna valor1 i si no, el segon valor2 (els dos valors poden ser cadenes, números, etc; per exemple: ***SELECT IF(1>3;"hola","adéu");*** Finalment, comentarem un parell de possibilitats més que ofereixen les sentències *SELECT*:

Treballar amb variables d'usuari

Les variables d'usuari són variables que qualsevol usuari pot crear per "recordar" resultats obtinguts en consultes anteriors. Aquests variables "s'incrusten" dins de la sentència SELECT escollida per tal de recollir el valor pertinent i s'identifiquen perquè el seu nom (que pot ser qualsevol) sempre ha de començar, però, amb una arroba. Per exemple, per recordar el preu mínim i màxim dels articles guardats a una taula "articles" podríem fer *SELECT @preumin :=MIN(preu), @preumax :=MAX(preu) FROM articles;* (on "preu" és una columna de la taula); fixeu-vos en el símbol ":=", utilitzat per assignar el valor obtingut de la consulta a una determinada variable. Un cop "recordat" aquest valor, el podríem utilitzar en qualsevol altre sentència SQL que volguéssim executar a posteriori, dins de la mateixa sessió d'usuari, així, per exemple: *SELECT * FROM articles WHERE preu=@preumin OR preu=@preumax;* Un altre exemple més útil és la sentència *SELECT @ultim:=LAST_INSERT_ID();* , on es fa ús de la funció **LAST_INSERT_ID()**, la qual sempre retorna l'últim valor autonumèric inserit a la sessió (i, per tant, en aquest cas, estaríem assignant aquest valor a la variable @ultim).

Treballar amb vistes

Una vista és una "taula virtual" formada a partir d'una consulta *SELECT*. La idea és fer servir les vistes com taules estàndar, però amb la diferència de que les dades no es troben a la vista en sí mateixa sinó que es troben a les taules reals a les què la vista fa referència. Treballar amb vistes permet realitzar tasques habituals d'una forma més senzilla, ocultant l'estructura real de la base de dades (que a vegades no interessa tenir en compte).

Per crear una vista simplement cal fer **CREATE VIEW nomVista AS SELECT ....** i per esborrar-la **DROP VIEW nomVista** . Per exemple, *CREATE VIEW colorsmascotes AS SELECT nommascota,color FROM mascotes* crearà una vista de dos columnes, totes dues obtingudes de la taula "mascotes". A partir d'aquí, podriem tractar la vista com si fos una taula més, realitzant per exemple consultes directament a ella (*SELECT * FROM colorsmascotes*) o fins i tot insercions i/o actualitzacions i/o eliminacions (per exemple, així: *INSERT INTO colorsmascotes VALUES ("chuky","negre");*). Cal tenir en compte, com ja hem dit, que les modificacions de les dades es realitzaran sobre les taules reals: això implica que totes les columnes que no estiguin presents a la vista però sí a la taula vinculada es modificaran també amb el seu valor per defecte i/o NULL i/o autonumèric. Per veure les vistes definides en una base de dades cal executar la sentència **SHOW FULL TABLES IN nomBD WHERE TABLE_TYPE LIKE 'VIEW';**

Actualització de dades

Per actualitzar el valor d'una o més columnes d'una taula es fa servir aquesta sentència: *UPDATE nomTaula SET nomCol1=nouValor1,nomCol2=nouValor2,... WHERE condició;* , on la condició del *WHERE* és fonamental per filtrar quins registres seran els què efectivament s'actualitzin: si no s'indiqués, tots els registres sense distinció s'actualitzarien als mateixos valors indicats.

Eliminació de dades

Per esborrar registres complets d'una taula es fa servir simplement aquesta sentència: *DELETE FROM nomTaula WHERE condició;* , on la condició del *WHERE* és fonamental per filtrar quins registres seran els què efectivament s'esborraran: si no s'indiqués, tots els registres sense distinció s'esborrarien.

Scripts SQL

Si es vol executar una comanda SQL directament des del terminal contra el servidor, es pot utilitzar el client *mariadb* de forma no interactiva per fer-ho: per això serveixen els paràmetres *-D* i *-e*:

-D nomBD : serveix per seleccionar directament la base de dades amb la que es treballarà; això evita haver d'utilitzar la sentència *use nomBD;*

-e "comanda SQL a executar" : aquesta comanda pot ser qualsevol (*INSERT, SELECT*, etc). En el cas de què retorni alguna dada, en comptes de veure aquest resultat per pantalla es podria redireccionar a un fitxer. Així, podríem guardar els resultats d'una consulta en un fitxer. Un exemple: *mariadb -u root -p1234 -D HospitalVeterinari -e "SELECT * FROM mascotes" > fitxerResultats.txt*

D'altra banda, el client *mariadb* també permet rebre les instruccions SQL a executar des d'un fitxer on s'hagin prèviament escrit, així: *mariadb -u root -p1234 < fitxerSentencies.txt*. D'aquesta manera, podríem tenir escrites en un fitxer les sentències *CREATE DATABASE, CREATE TABLE* i *INSERT* necessàries per "posar a punt" una base de dades i utilitzar aquest fitxer cada cop que volguem començar de zero.

NOTA: A la comanda anterior es podria afegir també el paràmetre *-f* per forçar a la comanda *mariadb* a continuar encara que trobi algun error al fitxer.
NOTA: Dins d'aquest fitxer d'instruccions es poden escriure comentaris d'una línia (si comencen amb el símbol #) o de vàries línies si comencen per **/*** i acaben per ***/**.

Si volguéssim executar el fitxer d'instruccions SQL des de dins del shell del propi client *mariadb*, s'ha d'utilitzar la comanda ***source /ruta/fitxerSentencies.txt***

Gestió d'usuaris i permisos

No és recomanable conectar amb el servidor MariaDB sempre com a usuari "root": és millor crear usuaris amb diferents permisos i fer servir cadascun per las tasques adequades segons els seus privilegis. Més tenint en compte que moltes vegades no accedirem al servidor MariaDB "manualment" des del client *mariadb* com estem fent al llarg d'aquest document sinó que ho farem a través d'scripts PHP o similars accessibles des de qualsevol navegador web del món (i molt sovint executats a la mateixa màquina on està funcionant el servidor MariaDB), la qual cosa fa que l'exposició pública a qualsevol vulnerabilitat que es pugui trobar i aplicar entrant com a "root" en la base de dades tingui uns efectes demolidors.

Un cop loguejats com a "root" dins del servidor MariaDB, per crear un usuari nou la manera més senzilla és executant la següent sentència SQL:

CREATE USER 'nomUsuari'@'xxx' IDENTIFIED BY "contrasenya";

Tal com es pot veure, després d'especificar el nom de l'usuari s'ha d'afegit el símbol @ i seguidament una cua. Aquesta cua representa la màquina client des d'on es connecta l'usuari. D'aquesta manera, es poden assignar diferents permisos al mateix usuari depenent des d'on es connecti. Valors possibles per aquesta cua són:

% : qualsevol client. Escriure *'nomUsuari'@'%'* seria equivalent a escriure només *'nomUsuari'*
localhost : només la màquina actual (equivalent a 127.0.0.1). Aquest valor és el que se sol fer servir si la manera de treballar és entrar prèviament a la màquina servidor via SSH per llavors, un cop dins, connectar al servidor MariaDB
nom.Maquina.Client : el nom d'una determinada màquina (cal, però, que el servidor en pugui fer la resolució de noms correctament, vé via arxiu "/etc/hosts", o DNS, etc)
xxx.yyy.www.zzz : la IP d'una determinada màquina
xxx.yyy.www.% : la IP d'una determinada xarxa de classe C (en general, el símbol % es pot utilitzar com a comodí en qualsevol lloc)

Si no es vol escriure la contrasenya en text pla a l'ordre *CREATE USER*, es pot escriure "hasheada" (fent servir l'algoritme propi que utilitza el servidor MariaDB). Per obtenir prèviament la contrasenya "hasheada" simplement cal executar *SELECT PASSWORD("contrasenya");* i, un cop obtinguda, llavors podrem escriure la contrasenya "hasheada" a l'ordre *CREATE USER* afegint la paraula PASSWORD, així:

*CREATE USER 'nomUsuari'@'xxx' IDENTIFIED BY **PASSWORD** "contrasenyaHasheada";*

D'altra banda, es pot canviar de contrasenya en qualsevol moment amb la comanda *SET PASSWORD = PASSWORD("contrasenya");* -per canviar-se-la un mateix- o *SET PASSWORD FOR 'nomUsuari'@'xxx' = PASSWORD("contrasenya");* -per canviar-la a un altre, si tenim permisos-.

NOTA: Ja vam veure al principi d'aquest document que els usuaris poden implementar diferents mecanismes d'autenticació (entre els quals hem vist el basat en contrasenya però també el basat en el "socket" local). En el cas d'haver d'explicitar que es vol fer servir una contrasenya concreta amb el mecanisme d'autenticació per contrasenya (perquè l'usuari en qüestió en tingui implementat més, de mecanismes d'autenticació), la sintaxis llavors seria, com ja vam veure: *CREATE USER 'nomUsuari'@'xxx' IDENTIFIED VIA mysql_native_password USING PASSWORD('contrasenya');* o bé, en dos pasos: *CREATE USER 'nomUsuari'@'xxx'IDENTIFIED VIA mysql_native_password;* i *SET PASSWORD FOR usuari@localhost = PASSWORD("contrasenya");*

NOTA: L'ordre *CREATE USER* té moltes altres opcions, com ara definir si es vol que l'usuari hagi d'usar obligatòriament TLS per connectar-se al servidor (o no), o les condicions de caducitat de les contrasenyes, o bloquejar/desbloquejar l'usuari, etc. Per més informació, consulteu https://mariadb.com/kb/en/create-user

També existeix la comanda *RENAME USER 'nomUsuariAntic'@'xxx' TO 'nomUsuariActual'@'xxx';* per canviar el nom d'un usuari. Finalment, per eliminar un usuari del servidor MySQL es pot utilitzar la comanda *DROP USER 'nomUsuari'@'xxx';*.

Un cop creat els usuaris, se'ls haurà de donar els permisos necessaris amb l'ordre *GRANT*, així:

GRANT permís1,permís2,... ON lloc TO 'nomUsuari'@'xxx';

NOTA: Fixeu-vos com hem d'indicar la "cua" darrera del @ en el nom de l'usuari a assignar els permisos, ja que segons el valor d'aquesta cua es podran assignar permisos diferents segons si l'usuari es connecta des d'un lloc o des d'un altre.

L'ordre anterior assigna a l'usuari indicat els permisos indicats (de seguida veurem quins poden ser) corresponents al "lloc" indicat. Poden haver tres "llocs" (en realitat poden ser més, però no els veurem), depenent dels quals els permisos disponibles per assignar podran ser uns o uns altres:

nomBD.nomTaula :Indica que els permisos assignats s'apliquen a una taula concreta de la BD
*nomBD.** : Indica que els permisos assignats s'apliquen a totes les taules d'una BD
. : Indica que els permisos assignats s'apliquen a totes les BD del servidor

La llista de permisos més habituals (¡no està completa!) que podem assignar a un usuari l'escrivim a continuació (la llista completa de permisos es pot obtenir executant la comanda *SHOW PRIVILEGES;*):

CREATE : Permís per crear base de dades o taules
CREATE VIEW : Permís per crear vistes
DROP : Permís per eliminar base de dades o taules
ALTER: Permís per modificar l'estructura d'una taula
INSERT: Permís per inserir dades en una taula
SELECT: Permís per realitzar consultes en una taula
UPDATE: Permís per modificar les dades existents en una taula
DELETE: Permís per eliminar dades existents en una taula
SHOW DATABASES : Permís per executar la comanda homònima (permís vàlid sols en *.*)
SHOW VIEW : Permís per executar la comanda homònima
CREATE USER: Permís per crear altres usuaris
GRANT OPTION : Permís per concedir permisos (els ha de tenir ell mateix) a altres usuaris
ALL : Tots els permisos
USAGE : Cap permís (per defecte és l'estat que tenen els usuaris creats per *CREATE USER*)
SHUTDOWN : Permís per apagar el servidor (permís vàlid sols en *.*)
FILE : Permís per llegir/crear fitxers emmagatzemats dins la màquina servidora (via LOAD DATA INFILE i/o SELECT ...INTO OUTFILE) amb els mateixos privilegis que té el servidor MariaDB (vàlid sols en *.*)

NOTA: A l'ordre anterior es poden afegir al final clàusules extra -la primera d'elles seguida de la paraula *WITH*- que permeten assignar permisos en l'àmbit de les conexions realitzades:

MAX_QUERIES_PER_HOUR nº : si nº és 0 (valor per defecte) no hi ha límit
MAX_UPDATES_PER_HOUR nº : si nº és 0 (valor per defecte) no hi ha límit
MAX_CONNECTIONS_PER_HOUR nº : si nº és 0 (valor per defecte) no hi ha límit
MAX_USER_CONNECTIONS nº : indica el nº de conexions concurrents d'un mateix usuari. Si nº és 0 (valor per defecte) el límit vindrà donat per la variable de sistema "max_user_connections".

NOTA: La comanda GRANT també es pot fer servir per crear usuaris i assignar-lis permisos "tot en un" (és a dir, no caldria fer servir l'ordre CREATE USER prèviament). Això és pot aconseguir simplement afegint a l'ordre *GRANT* "estàndard" la clàusula *IDENTIFIED BY "contrasenya";* (o *IDENTIFIED BY PASSWORD "contrasenyaHasheada";*, així: *GRANT permís1,permís2,... ON lloc TO 'nomUsuari'@'xxx' IDENTIFIED BY "contrasenya";*

En tot cas, després d'haver assignat els permisos adients (ja sigui el primer cop o en haver fet algun canvi respecte els eventuals permissos anteriors que tingués un determinat usuari), cal executar la següent ordre per tal que el servidor MariaDB se n'adoni i "refresqui" els privilegis als elements afectats:

FLUSH PRIVILEGES;

Per treure permisos prèviament assignats, s'utilitza la comanda *REVOKE*, així: *REVOKE permís1,permís2 ON lloc FROM 'nomUsuari'@'xxx' ;* Per veure quins permisos té l'usuari actualment utilitzat, cal fer: *SHOW GRANTS; ;* per veure els permisos d'un altre usuari (si es té el permís corresponent) cal fer *SHOW GRANTS FOR 'nomUsuari'@'xxx';*

NOTA: Tota la informació sobre els usuaris, contrasenyes, permisos, etc es guarda en una base de dades predefinida anomenada "**mysql**". Concretament, els permisos que afecten al "lloc" *.* s'hi guarden a la taula "**global_priv**"; els que afecten al "lloc" *nomBD.** s´hi guarden a la taula "**db**"; els que afecten al "lloc" *nomBD.nomTaula* es guarden a la taula "**tables_priv**". Per exemple, la comanda *SET PASSWORD FOR 'root'@'%' = PASSWORD("1234");* seria equivalent a executar *UPDATE mysql.global_priv SET PASSWORD=PASSWORD("1234") WHERE User="root" AND Host="%";* i seguidament *FLUSH PRIVILEGES;* Igualment, afegir un nou usuari anomenat 'pepe'@'%' amb contrasenya "1234" i tots els permisos seria equivalent a executar *INSERT INTO mysql.global_priv VALUES ('%', 'pepe', PASSWORD('1234'),'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');* i seguidament *FLUSH PRIVILEGES;*

Gestió de les còpies de seguretat

La comanda de terminal *mariadb-dump* (que s'instal.la per defecte juntament amb el client *mariadb*) serveix per crear (i restaurar) còpies de seguretat de les dades emmagatzemades en una determinada base de dades (o més d'una) "en calent" (és a dir, mentre el servidor està funcionant, sense alterar-lo). En realitat, aquestes còpies de seguretat no són més que fitxers de texte (anomenats "scripts SQL") que contenen totes les instruccions SQL necessàries per reproduïr l'estructura i les dades que es volen preservar, gràcies a la presència de sentències *CREATE DATABASE, CREATE TABLE i INSERT.*

* Si aquesta comanda s'invoca així: *mariadb-dump -u root -p [opcions] --all-databases > script.sql*, farà la còpia de totes les bases de dades existents al servidor (i la comanda *mariadb -u root -p < script.sql* les restaurarà)

* Si s'invoca així: *mariadb-dump -u root -p [opcions] --databases nomBD1 nomBD2 ... > script.sql* farà la còpia de les bases de dades indicades (i la comanda *mariadb -u root -p < script.sql* les restaurarà)

* Si s'invoca així: *mariadb-dump -u root -p [opcions] nomBD > script.sql* només farà la còpia de la base de dades indicada (opcionalment també es poden indicar nom de taules per fer només la còpia d'aquestes taules, però això no sol ser habitual) i la comanda *mariadb -u root -p -D nomBD < script.sql* la restaurarà (¡observeu com nomBD ha d'haver sigut creada prèviament!).

D'altra banad, les opcions més útils de la comanda *mariadb-dump* són:

*--add-drop-database :*Afegeix una sentència *DROP DATABASE* abans de cada *CREATE DATABASE*
-n : No afegeix la sentència *CREATE DATABASE*
--add-drop-table : Afegeix una sentència *DROP TABLE* abans de cada *CREATE TABLE* (per defecte)
-f : Continua encara que hi hagi errors
-d : No treu cap dada, només l'estructura
--single-transaction --skip-lock-tables: Assegura la coherència i integritat de les dades recollides "en calent" del servidor
-T /ruta/carpeta : Crea,per cada taula i a la carpeta indicada que es trobarà al servidor (per tant, aquesta opció només funciona si l'usuari emprat té el permís FILE), un arxiu anomenat "nomTaula.sql" que conté la sentència CREATE TABLE necessària per poder generar la taula en qüestió i un arxiu anomenat "nomTaula.txt" amb les dades contingudes en aquesta taula (allà cada registre apareix en una línia diferent i els camps estan separats entre sí per tabulacions; això es pot canviar amb els paràmetres *–fields-terminated-by, --fields-enclosed-by* i *–lines-terminated-by*).

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Queden més coses "en el tinter", però per no fer massa feixuc aquest document, ho deixarem per més endavant, quan ho necessitem. Concretament, m'estic referint a temes avançats d'administració com ara la realització de còpies incrementals, l'optimització del motor InnoDB, l'activació dels diferents modes del servidor, 'ús de les diferents variables de sistema i de servidor, la profundització en consultes de tipus JOIN, l'ús de l'eina MySQL Workbench, els mètodes possibles de recuperació de la contrasenya de l'usuari "root",etc.