Tema 7 – Lenguaje SQL: DCL. Funciones y Administración

1º DAW – Bases de Datos

J. Carlos Moreno

Curso 2021/2022

Contenido

7 Lenguaje SQL–DCL. Tratamiento de Datos y Funciones.

7.1 DCL – Lenguaje de Control de Datos.

En inglés DATA CONTROL LANGUAGE, es el lenguaje de control de datos, que incluye una serie de comandos que permiten al administrador controlar el acceso a los datos contenidos en la base de datos, otorgando y denegando privilegios sobre dichos datos.

Otras de las funciones básicas del DCL es la gestión de transacciones, teniendo como finalidad garantizar algo tan importante como la integridad y validez de los datos. Hay que indicar que no todos los sistemas gestores de bases de datos incluyen comando para la gestión de transacciones, sí MySQL.

7.2 Acceso y Seguridad de los Datos

El usuario que crea un objeto en la base de datos es dueño del mismo, pero MySQL contiene comandos para poder compartir dicho recurso y gestionar su acceso. Para ello proporciona los siguientes comandos DCL:

- CREATE USER
- DROP USER
- GRANT y REVOKE
- SHOW GRANTS
- RENAME USER
- SET PASSWORD

7.2.1 Creación de Usuarios. CREATE USER.

Permite la creación de una nueva cuentade acceso MySQL sin ningún tipo de privilegios.

Para usarlas, debe tener el permiso global CREATE USER o el permiso INSERT para la base de datos MySQL. CREATE USER crea un nuevo registro en la tabla mysql.user sin ningún tipo de privilegios.Puede ocurrir que el usuario no tenga privilegios CREATE USER y sí INSERT, en este caso podrá INSERTAR directamente un registro en la tabla mysql.user creando así un nuevo usuario sin usar el comando CREATE USER.

Sintaxis

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password'] [, user [IDENTIFIED BY
[PASSWORD] 'password']] ...;
```

Observaciones:

- Podemos establecer opcionalmente una contraseña mediante la cláusula IDENTIFID BY
- Normalmente en el nombre de usuario se especifica el nombre del servidor desde el que se puede acceder: user@servidor, por ejemplo *jcarlos@localhost*

En cualquier momento podremos acceder a los datos de la tabla mysql.user y comprobar la ejecución del CREATE USER, para ello usaremos el siguiente comando SQL:

```
SELECT * FROM mysql.user
```

Veamos los siguientes ejemplos

```
-- CREATE USER

-- EJEMPLO-01
-- Crear el usuario Pedro de forma que sólo pueda acceder desde el
localhost
-- asignar contraseña 123456

CREATE USER 'Pedro'@'localhost' IDENTIFIED BY '123456';

-- EJEMPLO-02
-- Crear el usuario Maria de forma que se pueda conectar desde
cualquier máquina o servidor
-- No asignar contraseña.

CREATE USER 'Maria';
```

Ambos usuarios creados en los ejemplos anteriores podrán acceder a la base de datos MySQL pero no tendrán asignados ningún tipo de privilegio. Sólo podrán usar y gestionar la base de datos de prueba TEST.

Veamos el siguiente ejemplo donde creamos un usuario con la contraseña emcriptada.

```
-- CREATE USER anonimo con contraseña 123456 pero encriptada

SELECT PASSWORD('123456');

-- Devuelve: *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9

CREATE USER anonimo@localhost IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

7.2.2 Borrar Usuarios. DROP USER.

El comando DROP USER borra una o más cuentas MySQL. Para usarlo, debe tener el permiso global CREATE USER o el permiso DELETE para la base de datos MySQL.

Sintaxis

```
DROP USER user [, user] ...
```

Ejemplos

```
-- DROP USER

-- EJEMPLO-03
-- Eliminar el usuario Pedro y Maria creados en los ejemplos anteriore

DROP USER Pedro@localhost, Maria;
```

7.2.3 AsignarPrivilegios. GRANT.

En el punto anterior aprendimos cómo crear usuarios pero sin asignación de ningún tipo de privilegios, por lo tanto el usuario podrá acceder a MySQL con la contraseña establecida en su caso y sólo podrá usar la base de datos de prueba TEST.

GRANT permite conceder permisos y privilegios personalizados a usuarios bien existentes o no existentes ya que si no existen procede a su creación.

Sintaxis

```
GRANT
Tipo_Privilegios [(lista_columnas)]
 [, Tipo_Privilegios [(lista_columnas)]] ...
    ON Nivel_Privilegio
TO usuario [IDENTIFIED BY 'password'][, usuario …]
    [WITH {GRANT OPTION | resource_option} ...]

Resource_option
    MAX_QUERIES_PER_HOUR count |
    MAX_UPDATES_PER_HOUR count |
    MAX_CONNECTIONS_PER_HOUR count |
MAX_USER_CONNECTIONS count]
```

Los tipos de permisos y privilegios(**Tipos_Privilegios**) a nivel básico que se pueden otorgar son los siguientes:

- **ALL PRIVILEGES**: permite a un usuario de MySQL acceder a todas las bases de datos asignadas en el sistema.
- **CREATE**: permite crear nuevas tablas o bases de datos.
- **DROP**: permite eliminar tablas o bases de datos.
- **DELETE**: permite eliminar registros de tablas.
- **INSERT**: permite insertar registros en tablas.
- **SELECT**: permite leer registros en las tablas.
- **UPDATE**: permite actualizar registros seleccionados en tablas.
- **GRANT OPTION**: permite remover privilegios de usuarios.

En la siguiente tabla se muestran todos los tipos de privilegios.

Privilegio	Descripción
ALL [PRIVILEGES]	Da todos los permisos simples excepto GRANT OPTION
ALTER	Permite el uso de ALTER TABLE
ALTER ROUTINE	Modifica o borra rutinas almacenadas
CREATE	Permite el uso de CREATE TABLE

CREATE ROUTINE	Crea rutinas almacenadas
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLE
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER, y REVOKE ALL PRIVILEGES.
CREATE VIEW	Permite el uso de CREATE VIEW
DELETE	Permite el uso de DELETE
DROP	Permite el uso de DROP TABLE
EXECUTE	Permite al usuario ejecutar rutinas almacenadas
FILE	Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE
INDEX	Permite el uso de CREATE INDEX y DROP INDEX
INSERT	Permite el uso de INSERT
LOCK TABLES	Permite el uso de LOCK TABLES en tablas para las que tenga el permiso SELECT
PROCESS	Permite el uso de SHOW FULL PROCESSLIST
REFERENCES	No implementado
RELOAD	Permite el uso de FLUSH
REPLICATION CLIENT	Permite al usuario preguntar dónde están los servidores maestro o esclavo
REPLICATION SLAVE	Necesario para los esclavos de replicación (para leer eventos del log binario desde el maestro)
SELECT	Permite el uso de SELECT
SHOW DATABASES	SHOW DATABASES muestra todas las bases de datos
SHOW VIEW	Permite el uso de SHOW CREATE VIEW
SHUTDOWN	Permite el uso de mysqladminshutdown
SUPER	Permite el uso de comandos CHANGE MASTER, KILL, PURGE MASTER LOGS, and SET GLOBAL , el comando mysqladmindebug le permite conectar (una vez) incluso si se llega a max_connections
UPDATE	Permite el uso de UPDATE
USAGE	Sinónimo de “no privileges”
GRANT OPTION	Permite dar permisos
REFERENCES	No implementado

RELOAD	Permite el uso de FLUSH
REPLICATION CLIENT	Permite al usuario preguntar dónde están los servidores maestro o esclavo
REPLICATION SLAVE	Necesario para los esclavos de replicación (para leer eventos del log binario desde el maestro)
SELECT	Permite el uso de SELECT
SHOW DATABASES	SHOW DATABASES muestra todas las bases de datos
SHOW VIEW	Permite el uso de SHOW CREATE VIEW
SHUTDOWN	Permite el uso de mysqladminshutdown
SUPER	Permite el uso de comandos CHANGE MASTER, KILL, PURGE MASTER LOGS, and SET GLOBAL , el comando mysqladmindebug le permite conectar (una vez) incluso si se llega a max_connections
UPDATE	Permite el uso de UPDATE
USAGE	Sinónimo de “no privileges”
GRANT OPTION	Permite dar permisos

Los privilegios se pueden otorgar en varios niveles, dichos niveles se especifican en la cláusula **Nivel_Privilegio** de la sintáxisGRANT:

- **Nivel Global**. Permisos que se aplican a todas las bases de datos de un servidor dado, normalmente en nuestro caso localhost. Estos permisos se almacenan en la table*mysql.user*. En los ejemplos *GRANT ALL ON *.** y *REVOKE ALL ON *.** se otorgan y quitan permisos globales.
- **Nivel de Base de Datos**. Los permisos de base de datos se aplican a todos los objetos en una base de datos dada. Estos permisos se almacenan en las tablas *mysql.db* y *mysql.host* . *GRANT ALL ON db_name.** y *REVOKE ALL ON db_name.** otorgan y quitan permisos a nivel de base de datos.
- **Nivel de tabla**. Los permisos de tabla se aplican a todas las columnas en una tabla dada. Estos permisos se almacenan en la tabla *mysql.tables_priv*. *GRANT ALL ON db_name.tbl_name* y *REVOKE ALL ON db_name.tbl_name* otorgan y quitan permisos sólo de tabla.
- **Nivel de columna**. Los permisos de columna se aplican a columnas en una tabla dada. Estos permisos se almacenan en la tabla *mysql.columns_priv* . Usando REVOKE, debe especificar las mismas columnas que se otorgaron los permisos. En el siguiente ejemplo se otorgan privilegios de consulta a nivel de conlumna, *GRANT SELECT(ISBN,TITULO, AUTORES) onventaseditorial.libros to Marta@localhost;*

En una sentencia GRANT es preciso indicar el usuario al que se le otorgan los privilegios, en caso de no existir MySQL procede a su creación, así con la claúsula IDENTIFIED BY tenemos la opción de especificar su contraseña de acceso.

Veamos los siguientes ejemplos

```
-- GRANT

-- EJEMPLO-04
-- Crear un usuario sin privilegios usando GRANT con contraseña -- '123456'

SELECT PASSWORD('123456');
-- Devuelve: *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9

-- Ahora uso la cadena encriptada para crear el usuario, de esta forma --
si tengo que almacenar el scriptno desvelo la contraseña.

GRANT USAGE ON *.* TO anonimo@localhost IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';

-- El comando anterior es equivalente al siguiente

CREATE USER anonimo@localhost IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

```
-- GRANT
-- EJEMPLO-05
-- Usuario Carlos sólo podrá consultar la base de datos ventaseditorial
GRANT SELECT ON ventaseditorial.* TO Carlos;

-- EJEMPLO-06
-- Usuario Carlos tendrá todos los priviletios sobre la base de datos
ventaseditorial
GRANT ALL ON ventaseditorial.* TO Carlos;

-- EJEMPLO-07
-- Usuario Carlos sólo podrá consultar la tabla clientes de ventaseditorial
GRANT SELECT ON ventaseditorial.clientes TO Carlos;

-- EJEMPLO-08
-- Usuario Carlos sólo podrá consultar la tabla clientes y la tabla libros
-- de la base de datos ventaseditorial
GRANT SELECT ON ventaseditorial.clientes TO Carlos;
GRANT SELECT ON ventaseditorial.libros TO Carlos;

-- EJEMPLO-09
-- El usuario Carlos sólo podrá consultar las columnas ISBN, TITULO,
AUTORES Y PVPSINIVA
-- de la tabla libros de ventaseditorial
GRANT SELECT(ISBN, TITULO, AUTORES, PVPSINIVA) ON ventaseditorial.libros TO
Carlos;

-- EJEMPLO-10
-- Asignar todos los privilegios al usuario Carlos
GRANT ALL PRIVILEGES ON *.* TO Carlos WITH GRANT OPTION;
```

La cláusula **WITH GRANT OPTION** proporciona al usuario la oportunidad de dar a otros usuarios cualquier privilegio que que éste tenga en el nivel de privilegios especificado. Se debe ser cuidadoso con a quien se le da el privilegio GRANT OPTION, porque dos usuarios con privilegios diferentes pueden unirlos.

No se puede conceder a otro usuario un privilegio que no se posee; el privilegio GRANT OPTION permite dar sólo aquellos privilegios que se poseen.

Hay que saber que cuando se concede a un usuario el privilegio GRANT OPTION en un nivel particular, cualquier privilegio que posea el usuario (o que se le conceda en el fututo) en ese nivel, también se puede conceder por ese usuario. Supongamos que se concede a un usuario el privilegio INSERT en una base de datos. Si se concede a continuación el privilegio SELECT en la base de datos y se especifica WITH GRANT OPTION, el usuario puede dar a otros no sólo el privilegio SELECT, sino también el INSERT. Si a continuación se concede el privilegio UPDATE al usuario en la base de datos, puede conceder a otros los privilegios INSERT, SELECT y UPDATE.

No se deben conceder privilegios ALTER a un usuario normal. Si se hace, el usuario puede intentar transtornar el sistema de privilegios mediante el renombrado de tablas.

Las opciones MAX_QUERIES_PER_HOUR count, MAX_UPDATES_PER_HOUR count y MAX_CONNECTIONS_PER_HOUR count son nuevasen MySQL 4.0.2. Sirven para limitar el número de consultas, actualizaciones y entradas que un usuario puede realizar durante una hora. Si el contador se pone a 0 (el valor por defecto), significa que no hay límite para ese usuario.

La opción MAX_USER_CONNECTIONS count es nueva en MySQL 5.0.3. Limita el número máximo de conexiones simultáneas que puede hacer la cuenta. Si count es 0 (valor por defecto), la variable de sistema max_user_connections determina el número de conexiones simultáneas para la cuenta.

7.2.4 Mostrar Privilegios. SHOW GRANTS

El comando SHOW GRANTS se utiliza para consultar los privilegios asignados a un usuario

```
SHOW GRANTS FOR User
```

```
-- SOWH GRANT

-- EJEMPLO-11
-- Mostrar los privilegios asignados a root
SHOW GRANTS FOR root@localhost;
```

7.2.5 EliminarPrivilegios. REVOKE

Permite eliminar privilegios que previamente hemos asignado a un usuario MySQL con el comando GRANT.

La sintaxis sería la siguiente:

```
REVOKE
Tipo_Privilegios [(lista_columnas)]
 [, Tipo_Privilegios [(lista_columnas)]] ...
    ON Nivel_Privilegio
FROMusuario[, usuario …]

REVOKE ALL PRIVILEGES, GRANT OPTION
FROM usuario [, usuario] ...
```

Veamos algunos ejemplos

```
-- REVOKE

-- EJEMPLO-12
-- Quitar los privilegios de administrador asignados a Carlos
-- en el ejemplo 10
REVOKE ALL PRIVILEGES, GRANT OPTION FROM Carlos;

-- EJEMPLO-13
-- Eliminar al usuario Carlos los privilegios de consultas a las columnas ISBN,
-- TITULO, AUTORES Y PVPSINIVA de la tabla libros de ventaseditorial
REVOKE SELECT(ISBN, TITULO, AUTORES, PVPSINIVA) ON ventaseditorial.libros FROM
Carlos;
```

7.2.6 Renombrar Cuentas. RENAME USER.

Renombra cuentas de usarioMySQL existentes

Sintaxis

```
RENAME USER old_user TO new_user [, old_user TO new_user] ...
```

7.2.7 CambiarContraseña. SET PASSWORD

El comando SET PASSWORD asigna una contraseña a una cuenta de usuario MySQL existente.

Sintaxis

```
SET PASSWORD FOR user = PASSWORD('password')
```

7.3 Transacciones

Es un conjunto de órdenes que se ejecutan de manera atómica o indivisible, es decir o se ejecutan todas o ninguna.

Para ser consideradas como tales se deben cumplir las cuatro propiedades ACID:

- Atomicidad. Asegura que se ejecutan todas o ninguna. No puede quedar a medias.
- Consistencia. O integridad que asegura que sólo se empieza lo que se puede acabar.
- Aislamiento. Asegura que ninguna operación afecta a otra pudiendo causar errores.
- Durabilidad. Una vez realizada la operación permanecerán los cambios.

Ejemplo clásico de transacción es una transferencia económica en la que debe sustraerse una cantidad de una cuenta, hacer una serie de cálculos relacionados con comisiones, intereses, etc. e ingresar en otras cuentas. Todo ello debe ocurrir de manera simultánea como si fuese una sola operación ya que de otro modo se generarían inconsistencias.

Una de las características de los sistemas gestores es si permiten o no el uso de las transacciones en sus tablas. En el caso de MySQL esto depende del tipo de tabla o motor utilizado. MySQL soporta distintos tipos de tablas tales como ISAM, MyISAM, InnoDB y BDB.

Las tabas que permiten transacciones son del tipo InnoDB. Están estructuradas de forma distinta que MyISAM, ya que se almacenan en un solo archivo en lugar de tres y, además de transacciones, permiten definir reglas de integridad referencial con la restricción FOREIGN KEY por ejemplo.

Las transacciones aportan una fiabilidad superior a la base de datos. Si disponemos de una serie de operaciones SQL que deben ejecutarse en conjunto, con el uso de transacciones podemos tener la certeza de que nunca nos quedaremos a medio camino de su ejecución. De hecho podríamos decir que las transacciones aportan una característica de “deshacer” a las aplicaciones de base de datos.

Las tablas que soportan transacciones, como es el caso de InnoDB, son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor, por otra parte pueden aumentar el tiempo de proceso de instrucciones.

Veamos el siguiente ejemplo: una cantidad de dinero es transferida de la cuenta de un cliente cc1 a otro cc2, se requerirán por lo menos dos instrucciones:

```
-- Transacciones

UPDATE cuentas SET balance=saldo- cantidad_transferida WHERE
cod_cliente=cc1;
UPDATE cuentas SET balance=saldo+cantidad_transferida WHERE
cod_cliente=cc2;
```

¿Qué ocurre si cae el sistema justo después de haber ejecutado el primer UPDATE?. Es evidente que surgirían inconsistencias que se evitarían usando las Transacciones.

Los pasos para usar transacciones en MySQL son:

- Iniciar la transacción con el uso de la sentencia *START TRANSACTION* o *BEGIN*

- Se ejecutan las instrucciones de actualización, inserción o eliminación de registros mediante las correspondientes instrucciones MySQL contenidas en el proceso.
- Para finalizar la transacción:
- Para confirmar los cambios en la Base de Datos y así confirmar la transacción ejecutamos *COMMIT*
- Si sucede algún problema y deseamos cancelar la transacción y así los cambios realizados, ejecutamos *ROLLBACK*

```
-- EJEMPLO: 14. Veamos en el siguiente ejemplo para comprobar como -- -
-- trabajan las transacciones

-- 1. Comprobamos el estado de la variable autocommit
SHOW VARIABLES LIKE 'autocommit';
-- Si devuelve el valor 1 la desactivamos con sólo de esta forma
-- podremos hacer uso de las transacciones en esta conexión
SET AUTOCOMMIT = 0;
-- También lo podríamos haber hecho iniciando una transacción con
START TRANSACTION;

-- 2. Creamos una tabla en la BBDD Test, de tipo InnoDB e insertamos --
-- unos datos
USE TEST;
CREATE TABLE Transacciones(
      IdTransac INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      Valor INT) TYPE=InnoDB;
INSERT INTO Transacciones VALUES
(null, 1),
(null, 2),
(null, 3);

-- 3. Iniciamos la transacción
BEGIN;
INSERT INTO Transacciones VALUES
(null, 4);

-- 4. Anulamos la transacción
ROLLBACK;
-- Si ahora ejecutamos un SELECT comprobamos como no se ha realizado
-- ninguna inserción.
SELECT * FROM Transacciones;

-- 5. Ahora veremos que ocurre si perdemos la conexión antes de que
-- la transacción sea completada
BEGIN;
INSERT INTO Transacciones VALUES
(null, 4);
SELECT * FROM Transacciones;
EXIT;

-- Cuando recuperemos de Nuevo la conexión podemos verificar que -- --
-- tampoco se insertó el registro

-- 6. Ahora vamos a repetir la sentencia INSERT pero haremos un COMMIT
-- antes de prefer la conexión
BEGIN;
INSERT INTO Transacciones VALUES
(null, 4);
SELECT * FROM Transacciones;
COMMIT;
EXIT;

-- Una vez ejecutado el COMMIT la transacción es completada y todos los
-- cambios afectan de forma permanente a la Base de Datos.
```

En las tablas InnoDB toda la actividad del usuario se produce dentro de una transacción. Si el modo de ejecucuónautomatic(autocommit) está activdado, cada sentencia SQL conforma una transacción individual por sí misma. MySQL siempre comienza una nueva conexión con la ejecución automatic habilitada.

Si el modo de ejecución automatic se deshabilitó con *SET AUTOCOMMIT = 0*, entonces puede considerarse que un usuario siempre tien una transacción abierta. Una sentencia SQL *COMMIT* o *ROLLBACK* termina la transacción vigente y comienza una nueva. Ambas sentencias liberan todos los bloqueos InnoDB que se establecieron durante la transacción vigente. Un *COMMIT*siginfica que los cambios hechos en la transacción actual se convierten en permamentes y se devuelven visible para los otros usuarios. Por otra parte, una sentencia *ROLLBACK*, cancela todas las modificaciones producidas en la transacción actual.

Si la conexión tiene la ejecución automática habilitada, el usuario puede igualmente llevar a cabo una transacción con varias sentencias si la comienza explícitamente con *START TRANSACTION* o *BEGIN* y la termina con *COMMIT* o *ROLLBACK*.

Hay instrucciones SQL que dada sus caracterísitcas implican confirmación automática:

- DDL: CREATE, ALTER, DROP, RENAME y TRUNCATE.
- DCL: GRANT y REVOKE
- Instrucciones de control y bloqueo de tablas: START TRANSACTION, BEGIN, LOCK y UNLOCK
- Instrucciones de carga de datos. LOAD DATA.
- Instrucciones de administración: ANALIZE, CHECK, OPTIMIZE y REPAIR.

Otros comandos para la gestión de transacciones:

- SAVEPOINT id: nombra cierto paso en una transacción
- ROLLBACK TO SAVEPOINT id: permite deshacer las operaciones realizadas a partir del identificador.
- RELEASE SAVEPOINT id: elimina o libera el savepointcreado.

Cualquier operación COMMIT o ROLLBACK sin argumentos eliminara todos los savepoints creados.

```
-- EJEMPLO15

STAR TRANSACTION;
INSERT INTO Transacciones VALUES (null, 5);
SAVEPOINT 1;
INSERT INTO Transacciones VALUES (null, 6), (null, 7), (null, 8);
ROLLBACK TO 1;

-- Sólo se insertan realmente los datos del primer INSERT.
```

7.4 Bloqueo de Tablas

Bloquear una tabla o vista implica exclusividad sobre ella, nadie más puede hacer uso de la misma.

MySQL permite el bloqueo de tablas por parte de los clientes con el objeto de cooperar con otras sesiones de clientes o de evitar que estos puedan modificar datos que necesitamos en nuestra sesión.

Los bloqueos permiten simular transacciones así como acelerar operaciones de modificación o de inserción de datos.

7.4.1 Comandos de Bloqueo de Tablas

La instrucción para bloqueo de tablas es LOCK que sigue la siguiente sintaxis:

```
LOCK TABLES

tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

Para desbloquear tablas usaremos UNLOCK TABLES. Sólo podremos desbloquear todas las tablas a la vez.

Para realizar un bloqueo a todas las tablas de la base de datos usamos:

```
LOCK TABLES WITH READ LOCK
```

Bloquear una tabla implica indicar su nombre **tbl_name** o alias y el tipo de bloqueo, lectura o escritura (**READ/WRITE**).

7.4.2 Tipos de Bloqueo

Existen dos tipos de bloqueos:

- READ

  La sesión o el cliente que tiene el bloqueo puede leer pero ni él ni ningún otro cliente podrá escribir en la tabla.

  Es un bloqueo que pueden adquirir varios clientes simultáneamente y permite que cualquiera pueda leer las tablas bloqueadas.

- [LOW PRIORITY] WRITE

  La sesión o cliente que adquiere este tipo de bloqueo puede leer y escribir en la tabla pero ningún otro cliente podrá acceder a ella o bloquearla.

Veamos el siguiente ejemplo de bloque de tablas de tipo **READ**.

```
-- Bloqueo la tabla libros y la tabla ventas
lock tables libros read, ventas read;

-- No puedo acceder a nignuna otra tabla
```

```
-- Por lo tanto este comando dará error
SELECT * FROM clientes;

-- Si deseo insertar un libro
-- También dará error
INSERT INTO libros VALUES
(null, null, null, 'La Historia Interminable', 4, 5,  10.30, 15, 6, 2, 10,
'2000-12-11');

-- Si cualquier otro usuario desde otra conexión quisiera modificar la tabla
-- libros tendría que esperar a que fuese desbloqueda

-- Una vez desbloqueadas las tabla
-- Puedo realizar las dos operaciones anteriores
UNLOCK TABLES;
```

Veamos un ejemplo de bloque a modo **WRITE**

```
-- Bloqueo la tabla libros de tipo WRITE
LOCK TABLES libros WRITE;

-- Ningún usuario podrá acceder ahora mismo a esta tabla
-- Sólo el usuario que la ha bloqueado.
-- Ninguno de los comando siguientes dará error
SELECT * FROM libros;
INSERT INTO libros VALUES
(NULL, NULL, NULL, 'La Historia Interminable', 4, 5,  10.30, 15, 6, 2, 10,
'2000-12-11');

-- No puedo acceder a ninguna tabla de la base de datos
-- Por lo tanto este comando daría error
SELECT * FROM clientes;

-- En el caso que otro usuario conectado quisiese acceder a la tabla
-- libros, quedaría bloquedo

UNLOCK TABLES;
```

7.4.3 Adquisición-Liberación de un Bloqueo

Cuando necesitamos adquirir bloqueos debemos hacerlo en la misma sentencia ya que si separamos las instrucciones automáticamente se desbloquean las anteriores quedando activo sólo el último bloqueo.

En ese momento **sólo tendremos acceso a las tablas bloqueadas**.

Los bloqueos de escritura tienen por defecto mayor prioridad ya que implican modificación de datos que deben realizarse lo más rápido posible. Así una sesión adquiere un bloqueo de lectura y otra sesión pretende uno de escritura, esté tendrá prioridad sobre otras solicitudes de bloqueos de lectura subsecuentes. De este modo hasta que la sesión que solicitó el bloqueo de escritura no libere dicho bloqueo no se podrán adquirir nuevos bloqueos de lectura.

Esto cambia si el bloqueo de escritura se adquiere con la opción LOW_PRIORITY en cuyo caso si se permite que los bloqueos de lectura se adquieran antes que el de escritura. De hecho el bloqueo de escritura sólo se concederá cuando no queden bloqueos de lectura pendientes.

Es obvio que si se solicita un bloqueo de escritura sobre una tabla con bloqueo de lectura en marcha, no se podrá otorgar hasta que no finalice dicho bloqueo de lectura.

La política de bloqueo de MySQL sigue la siguiente secuencia:

- Ordena internamente las tablas a bloquear
- Si una tabla debe bloquearse para lectura y escritura sitúa la solicitud de bloqueo de escritura en primer lugar
- Se bloquea cada tabla hasta que la sesión adquiere todos sus bloqueos

De este modo se evita el conocido ***deadlock*** o bloqueo mutuo, fenómeno que hace que el acceso a los bloqueos se pueda prolongar indefinidamente al no poder ningún proceso obtenerlos.

Si adquirimos un bloqueo sobre una tabla, todos los bloqueos activos en ese momento se liberan automáticamente.

Si comenzamos una **transacción todos los bloqueos se liberan automáticamente**.

7.4.4 Bloqueos y Transacciones

El uso de bloqueos está ligado al de las transacciones, especialmente para las tablas de tipo transaccional como InnoDB y CLUSTER.

En las tablas InnoDB los bloqueos se adquieren a nivel de fila permitiéndose que varios usuarios puedan bloquear varias filas simultáneamente.

En las tablas InnoDB todo es una transacción, de hecho, si ***autocommit*** está activado (=1) cada operación SQL es en sí misma es una transacción. En este caso podemos iniciar una transacción con ***START TRANSACTION*** o ***BEGIN***y terminarla con ***COMMIT*** para que los cambios sean permanentes o con ***ROLLBACK*** para deshacer los cambios.

En caso de ***autocommit*** inactivo se considera que siempre hay una transacción en curso en cuyo caso las sentencias ***COMMIT*** y ***ROLLBACK*** suponen el final de la transacción y el principio de la siguiente.

7.4.4.1 Tipos de Bloqueos en InnoDB

En este tipo de tablas diferenciamos dos tipos de bloqueo:

- **Bloqueo Compartido** (is): permite a una transacción la lectura de filas. Varias transacciones pueden adquirir bloqueos sobre las mismas filas pero ninguna transacción puede modificar dichas filas hasta que no se liberen los bloqueos.

  Sintaxis

```
SELECT … LOCK IN SHARE MODE
```

- **Bloqueo Exclusivo** (ix): permite a una transacción bloquear filas para actualización o borrado. En este caso las transacciones que deseen adquirir bloqueo exclusivo deberán esperar a que se libere el bloqueo sobre las filas afectadas.

  Sintaxis

```
SELECT … FOR UPDATE
```

También se soporta el "bloqueo de múltiple granularidad", según el cual una transacción puede indicar que va a bloquear algunas filas de una tabla bien para lectura (IS) o para escritura (IX). Es lo que denominamos una intención de bloqueo.

De este modo es más fácil para MySQL gestionar posibles conflictos evitando los temidos deadlock ya que permite a varias transacciones compartir la reserva de una tabla puesto que el bloqueo se produce fila a fila en el momento de la modificación. Así, si dos transacciones reservan la misma tabla, solo en el momento en que una de ellas esté modificando una fila, ésta quedará bloqueada.

Ejemplos de este tipo de bloqueos son:

- Sentencias ***SELECT … LOCK IN SHARE MODE***: para bloqueos tipo IS. De este modo se crea un bloqueo de lectura sobre las filas afectadas por la consulta SELECT.
- Sentencias ***SELECT … FOR UPDATE***: para bloqueos de tipo IX. De este modo se bloquean para escritura las filas afectadas por la consulta SELECT así como las entradas de índice asociadas.

```
/* Ejemplo 16: Usamos la base de datos Maratoon del tema anterior
Supongamos que queremos agregar un registro en la tabla corredor, pero
asegurándonosde que existe el club al que pertenece.
Para asegurar de que existe el club haríamos una consulta sobre club para
comprobarloy después insertamos al corredor. Sin embargo nadie nos asegura
que entre medio se elimine o modifique dicho club. Para evitarlo haríamos
la consulta de este modo: */

USE maratoon;
SELECT *FROM club WHERE Nombre = 'Nutrias Pantaneras Ubrique'
LOCK IN SHARE MODE;

/* Con lo que conseguirmos un bloqueo de lectura de modo que mientras no se
confirme o deshaga la transacción nadie podrá modificar los datos de
equipo. */
INSERT INTO corredor  VALUES (null, 'Pedro', 'Moreno Escot', 'Ubrique',
'1978-07-12', 'H', 38, 5, 1);

-- Si ejecutase este comando no me permitiría hacerlo
UPDATE CLUB SET Nombre='Nutrias Lugareñas' WHERE CodClub=1;

-- Sólo lo permitiría hasta que confirmase la transacción
COMMIT;
```

```
/* Ejemplo 17: Si queremos modificar el nombre corto de todos los clubs sin
temora que cualquier otro usuario pueda hacer lo mismo mientras tanto*/
SELECT NombreCorto FROM club FOR UPDATE;

-- Ahora sólo el usuario que tiene la sesión podrá modificar dicha columna
UPDATE Club SET NombreCorto='NUP' WHERE CodClub=1;

-- Una vez realizadas las modificaciones se confirma la transacción
COMMIT;
```

7.4.4.2 Niveles de aislamiento

Los niveles de aislamiento hacen referencia al grado de acutalización de los datos que leemos de una base de datos.

Así distinguimos 4 niveles:

- READ UNCOMMITED. Las lecturas se realizan sin bloqueo de manera que podemos estar leyendo un dato que ya ha sido modificado por otra transacción. Es lo que se denomina lectura inconsistente o sucia.
- READ COMMITED. En este tipo de lectura en cada instrucción se usan los valores más recientes y no los de comienzo de bloqueo. Es decir, si en alguna instrucción se modifica un valor que después se usará en otra, el valor será el modificado y no el original cuando se adquirió el bloqueo.
- REPETEABLE READ. Es el valor por defecto. En este caso se obtiene el valor establecido al comienzo de la transacción. Es decir, aunque durante la transacción los valores camben, se tendrán en cuenta siempre los de comienzo.
- SERIALIZABLE READ. Es como la anterior con la diferencia de que ahora de manera interna se convierten los SELECT en SELECT … LOCK IN SHARE MODE si autocommit está desactivado.

Con el comando SET TRANSACTION podemos establecer el nivel deseado de aislamiento de nuestras transacciones indicando si es a nivel global o sólo para nuestra sesión actual.

La sintaxises:

```
SET [GLOBAL | SESSION] TRANSACTIONISOLATION LEVEL

{ REPEATABLE READ
| READ COMMITTED
| READ UNCOMMITTED
| SERIALIZABLE
    }
```

7.5 Funciones enMySQL

MySQL dispone de un conjunto bastante extenso de funciones que nos van a permitir sacar mucho más rendimiento a nuestros datos con menor esfuerzo.

Las funciones se usan dentro de expresiones y generalmente producen tres tipos diferentes de resultados:

- Modifican la información original (convertir a minúscula una cadena que está en mayúscula)
- Las que modifican el formato en el que se muestra el resultado
- Obtienen un nuevo valor a partir de los valores de los parámetros.

Clasificación de las funciones MySQL:

- Funciones de lista
- Funciones Matemáticas
- Funciones de Cadena
- Funciones de Fecha y Hora

7.5.1 Funciones de lista de valores

Sintaxis	Función	Ejemplo	Resultado
GREATEST(*value1*,*value2*,...)	Retorna el argumento mayor	SELECT GREATEST(*1,5,6,7*)	7
LEAST(*value1*,*value2*,...)	Retorna el argumento MENOR	SELECT LEAST(*1,5,6,7*)	1
COALESCE(*value*,...)	Retorna el primer valor NO NULO	SELECT COALESCE(NULL,1);	1

7.5.2 Funciones Matemáticas

Sintaxis	Función	Ejemplo	Resultado
ABS(*X*)	Retorna el valor absoluto de *X*.	SELECT ABS(2);	2
ACOS(*X*)	Retorna el arcocoseno de x	SELECT ACOS(1);	0
ASIN(*X*)	Retorna el arcoseno de *X*,	SELECT ASIN(0.2);	0.20135792079033
ATAN(*X*)	Retorna la arcotangente de *X*	SELECT ATAN(2);	1.1071487177941
COS(*X*)	Retorna el coseno de *X*	SELECT COS(PI())	-1
CEILING(*X*) , CEIL(*X*)	Retorna el entero más pequeño no menor a *X*. · Redondea por exceso	SELECT CEILING(1.23);	2
FLOOR(*X*)	Retorna el valor entero más grande pero no mayor a *X*. · Redondea por defecto	SELECT FLOOR(1.23);	1
MOD(*N*,*M*) , *N* % *M*, *N* MOD *M*	Operación de módulo. Retorna el resto de *N* dividido por *M*.	SELECT MOD(234, 10);	4
PI()	Retorna el valor de π (pi)	SELECT PI()	3.141593
POW(*X*,*Y*) , POWER(*X*,*Y*)	Retorna el valor de *X* a la potencia de *Y*.	SELECT POW(2,2)	4
RADIANS(*X*)	Retorna el argumento *X*, convertido de grados a radianes. (Tenga en cuenta que π radianes son 180 grados.)	SELECT RADIANS(90);	1.5707963267949

RAND(),	Retorna un valor aleatorio en coma flotante del rango de 0 a 1	SELECT RAND();	0.9233482386203
ROUND(*X*)	Retorna el argumento *X*, redondeado al entero más cercano	SELECT ROUND(-1.23)	-1
SIGN(*X*)	Retorna el signo del argumento como -1, 0, o 1, en función de si *X* es negativo, cero o positivo.	SELECT SIGN(-32)	-1
SIN(*X*)	Retorna el seno de *X*, donde *X* se da en radianes.	SELECT SIN(PI())	1.2246063538224e-16
SQRT(*X*)	Retorna la raíz cuadrada de un número no negativo. *X*.	SELECT SQRT(4)	2
TAN(*X*)	Retorna la tangente de *X*, donde *X* se da en radianes.	SELECT TAN(PI())	-1.2246063538224e-16
TRUNCATE(*X*,*D*)	Retorna el número *X*, truncado a *D* decimales	SELECT TRUNCATE(1.223,1)	1.2

```
-- Obtener mediante una instrucción SQL seno de 45 grados
-- Para hacer este ejercicio primero tenemos que pasar 45 grados a radianes
-- Luego calcular el seno

SELECT SIN(RADIANS(45)) Seno_45;


-- Si quisiéramos un SELECT más descriptivo mostrando un texto
-- el ángulo, los radianes y el seno lo podremos hacer así

SELECT
    'Ángulo de:',
    45,
    RADIANS(45) radianes,
    SIN(RADIANS(45)) seno;

-- Obtener mediante una instrucción SQL coseno y el seno de 60 grados
SELECT
    'Ángulo de:',
    60,
    RADIANS(60) radianes,
    SIN(RADIANS(60)) seno,
    COS(RADIANS(60)) coseno;

-- Actualizar la tabla libros de la base de datos geslibros para
-- redondear el precio_venta de todos los libros por exceso
-- Es decir si un libro tiene precio_venta 20.50 se redondeará a 3.00
-- También si tiene 23.01 se redondeará a 24

UPDATE geslibros.libros
SET
    precio_venta = CEILING(precio_venta);

-- Ahora queremos redondear el campo precio_coste pero esta vez por defecto
-- es decir si cuesta 20.90 el precio será 20.00
```

```
UPDATE geslibros.libros
SET
    precio_coste = FLOOR(precio_coste);

-- Redondea las columnas precio_venta y precio_coste de la tabla libros
-- al entero más cercano
-- Es decir si el precio es 45.67 redondea a 46.00
-- En cambio si el precio es 45.49 redondea a 45.00
UPDATE geslibros.libros
SET
    precio_venta = ROUND(precio_venta),
    precio_coste = ROUND(precio_coste);

-- Obtener un número aleatorio entre 0 y 1 este último no inclusive
-- Jamás podrá ser 1 en cambio si podrá obtener el 0.
SELECT RAND();

-- Obtener un número aleatorio entre 0 y 9
SELECT FLOOR(RAND() * 10);

-- Obtener un número aleatorio entre 1 y 9
SELECT CEILING(RAND() * 9);

-- Simula el lanzamiento de un dado
-- Obtener entonces un número aleatorio entre 1 y 6
SELECT CEILING(RAND() * 6);

-- Simula el lanzamiento de 4 dados
SELECT
    CEILING(RAND() * 6) AS dado1,
    CEILING(RAND() * 6) AS dado2,
    CEILING(RAND() * 6) AS dado3,
    CEILING(RAND() * 6) AS dado4;
```

7.5.3 Funciones de Cadena de Caracteres

Sintaxis	Función	Ejemplo	Resultado
ASCII(*str*)	Retorna el valor numérico del carácter más a la izquierda de la cadena de caracteres *str*	SELECT ASCII('2')	50

CHAR(*N*,...)	interpreta los argumentos como enteros y retorna la cadena de caracteres que consiste en los caracteres dados por los códigos de tales enteros	SELECT CHAR(77,121,83,81,'76')	'MySQL'
CHAR_LENGTH(*str*)	Retorna la longitud de la cadena de caracteres *str*, medida en caracteres	SELECT CHAR_LENGTH('JUAN CARLOS');	11
CONCAT(*str1*,*str2*,...)	Retorna la cadena resultado de concatenar los argumentos	SELECT concat('Rodríguez García',', ', 'José');	Rodríguez García, José
CONCAT_WS(*separator*,*str1*,*str2*,...)	CONCAT con separador. El primer argumento es el separador para el resto de argumentos	SELECT concat_ws(', ','Rodríguez García', 'José');	Rodríguez García, José
ELT(*N*,*str1*,*str2*,*str3*,...)	Retorna *str1* si *N* = 1, *str2*if*N* = 2, y así	SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo')	'ej'
FIELD(*str*,*str1*,*str2*,*str3*,...)	Retorna el índice de *str* en la lista *str1*, *str2*, *str3*, ...	SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')	2
FIND_IN_SET(*str*,*strlist*)	Retorna un valor en el rango de 1 a *N* si la cadena *str* está en la lista de cadenas *strlist* consistente de *N*subcadenas	SELECT FIND_IN_SET('b','a,b,c,d');	2
INSERT(*str*,*pos*,*len*,*newstr*)	Retorna la cadena *str*, con la subcadena comenzando en la posición *pos* y *len* caracteres reemplazados por la cadena *newstr*	SELECT INSERT('Quadratic', 3, 4, 'What')	'QuWhattic'
INSTR(*str*,*substr*)	Retorna la posición de la primera ocurrencia de la subcadena*substr* en la cadena *str*.	SELECT INSTR('foobarbar', 'bar');	4
LEFT(*str*,*len*)	Retorna los *len* caracteres empezando por la izquierda de la cadena *str*.	SELECT LEFT('foobarbar', 5);	'fooba'
LOCATE(*substr*,*str*) ,	La primera sintaxis retorna la posición de la primera ocurrencia de la subcadena*substr* en la cadena *str*.	SELECT LOCATE('bar', 'foobarbar')	4
LOCATE(*substr*,*str*,*pos*)	Retorna la posición de la primera ocurrencia de la subcadena*substr* en la cadena *str*, comanzando en la posición *pos*	SELECT LOCATE('bar', 'foobarbar') SELECT LOCATE('bar', 'foobarbar',5)	7
LOWER(*str*)	Convierte minúsculas	SELECT LOWER('QUADRATICALLY')	'quadratically'
LTRIM(*str*)	Retorna la cadena *str* con los caracteres en blanco iniciales eliminados.	SELECT LTRIM(' barbar');	'barbar'
REPEAT(*str*,*count*)	cadena consistente de la cadena *str* repetida *count* veces	SELECT REPEAT('MySQL', 3);	'MySQLMySQLMySQL'

REPLACE(*str*,*from_str*,*to_str*)	Retorna la cadena *str* con todas las ocurrencias de la cadena *from_str* reemplazadas con la cadena *to_str*.	SELECT REPLACE('www.mysql.com', 'w', 'Ww');	'WwWwWw.mysql.com'
REVERSE(*str*)	Retorna la cadena *str* con el orden de los caracteres invertido	SELECT REVERSE('abc')	'cba'
RIGHT(*str*,*len*)	Retorna los *len* caracteres de la derecha de la cadena *str*.	SELECT RIGHT('foobarbar', 4);	'rbar'
RPAD(*str*,*len*,*padstr*)	Retorna la cadena *str*, alineada a la derecha con la cadena *padstr* con una longitud de *len* caracteres. Si *str* es mayor que *len*, el valor de retorno se corta a *len* caracteres.	SELECT RPAD('hi',5,'?');	'hi???'
RTRIM(*str*)	Retorna la cadena *str* con los espacios precedentes eliminados.	SELECT RTRIM('barbar ');	'barbar'
SUBSTRING(str,pos), SUBSTRING(str,pos,len)	Retornan una subcadena de la cadena str comenzando en la posición pos de longitud len	SELECT SUBSTRING('Quadratically',5); SELECT SUBSTRING('Quadratically',5,6); SELECT SUBSTRING('Sakila', -3);	'ratically' 'ratica' 'ila'
SUBSTRING_INDEX(*str*,*delim*,*count*)	Retorna la subcadena de la cadena *str* antes de *count* ocurrencias del delimitador *delim*.	SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2)	'www.mysql'
TRIM(*str*)	Elimina todos los espacios	SELECT TRIM(' bar ')	'bar'
UPPER(*str*)	Retorna la cadena *str* con todos los caracteres cambiados a mayúsculas	SELECT UPPER('Hej')	'HEJ'

Veamos la siguiente colección de ejemplos:

```
-- Colección de ejemplos basados en la base de datos
-- geslibros.sql

-- Nota: Se recomianda la ejecución de cada script para ver resultados


-- Muestra los autores junto con su longitud del nombre
SELECT
    id, nombre, CHAR_LENGTH(nombre) longitud
FROM
    autores;

-- Muestra el nombre de los autores pero con la nacionalidad
-- concatenada y separada por ","

SELECT
    id, CONCAT(nombre, ', ', nacionalidad) NombreNacionalidad
FROM
    autores;
```

```
-- Muestra el nombre la nacionalidad y el año de nacimiento
-- en una sola columna usando CONCAT_WS para especificar separador
SELECT
    id,
    CONCAT_WS(', ',
            nombre,
            nacionalidad,
            YEAR(fechaNac)) NombreNacionalidad
FROM
    autores;

-- Mostrar los tres primeros caracteres del nombre del autor
-- y además se muestran en MAY. Renombra la columna con Código
SELECT
    id, nombre, UPPER(LEFT(nombre, 3)) Código
FROM
    autores;

-- Mostrar los tres últimos caracteres del segundo apellido del autor
-- Mostrar en MAY
SELECT
    id, nombre, UPPER(RIGHT(nombre, 3)) Código
FROM
    autores;

-- Devolver la posición en la que empieza el primer apellido
-- dentro de la cadena nombre de autor
-- Lo que hago es buscar con LOCATE la posición del primer ' ' y sumarle 1
SELECT
    id, nombre, LOCATE(' ', nombre) + 1 AS Posición
FROM
    autores;

-- Devolver el nombre de autor todo en minúsculas
--
SELECT
    id, nombre, lower (nombre) AS NombreMin
FROM
    autores;

-- Actualizar la columna nombre de autor eliminado los posibles
-- espacios en blanco que pueda tener el nombre por la izq y der
UPDATE autores
SET
    nombre = TRIM(nombre);

-- Devuelve el nombre de los autores a partir del 5 caracter
SELECT
    id, nombre, SUBSTRING(nombre, 5) AS Cadena
FROM
    autores;

-- Devuelve sólo los apellidos de los autores
-- Este script no valdría para autores con nombre de pila compuesto
-- LOCATE(' ', nombre) + 1 -> devuelve la posición en la que se encuentra
-- el primer ' ' y le suma 1.
SELECT
    id,
    nombre,
    SUBSTRING(nombre,
        LOCATE(' ', nombre) + 1) AS Apellidos
FROM
    autores;

-- Devuelve los tres primeros caracteres del primer apellido
SELECT
    id,
```

```
    nombre,
    SUBSTRING(nombre,
        LOCATE(' ', nombre) + 1, 3) AS Apellidos
FROM
    autores;

-- Devuelve sólo el nombre de pila de los autores
-- Devuelve hasta la primera aparición del caracter ' '
-- SUBSTRING_INDEX(nombre, ' ', 1)
SELECT
    id, nombre, SUBSTRING_INDEX(nombre, ' ', 1) AS NombrePila
FROM
    autores;

-- Devuelve el segundo apellido de los autores
-- Sería válido si todos los autores tuviesen sus dos apellidos
-- SUBSTRING_INDEX(nombre, ' ', -1) -> devuelve la cadena nombre empezando
-- por lo derecha (-1) hasta que encuentra el primer espacio en blanco.

SELECT
    id, nombre, SUBSTRING_INDEX(nombre, ' ', -1) AS Apellido2
FROM
    autores;

-- Devuelve los dos apellidos de los autores
SELECT
    id, nombre, SUBSTRING_INDEX(nombre, ' ', -2) AS Apellidos
FROM
    autores;
```

7.5.4 Funciones de Fecha y Hora

7.5.4.1 Cláusula INTERVAL

Es una cláusula que se utiliza en numerosas funciones de fecha y hora

Sintaxis

```
INTERVAL Expresión Unidad
```

Utilizamos principalmente valores de interval para la aritmética de fecha y hora como se muestra a continuación:

```
date  +  INTERVAL  expresión unidad
date  -  INTERVAL  expresion unidad
```

MySQL define formatos estándar para **expresión** y **unidad** como se ilustra en la siguiente tabla:

Unidad	Expresión
DAY	DAYS
DAY_HOUR	'DAYS HOURS'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
HOUR	HOURS

Unidad	Expresión
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
MICROSECOND	MICROSECONDS
MINUTE	MINUTES
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
MONTH	MONTHS
QUARTER	QUARTERS
SECOND	SECONDS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
WEEK	WEEKS
YEAR	YEARS
YEAR_MONTH	'YEARS-MONTHS'

Los valores de intervalo también se utilizan en varias funciones temporales como **DATE_ADD**, **DATE_SUB** , **TIMESTAMPADD** y **TIMESTAMPDIFF** .

Veamos algunos **ejemplos**

```
--Ejemplos INTERVAL

SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
-- Salida: '1998-01-01 00:00:00'

SELECT INTERVAL 1 DAY + '1997-12-31';
-- Salida: '1998-01-01'

SELECT '1998-01-01' - INTERVAL 1 SECOND;
-- Salida: '1997-12-31 23:59:59'

SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 SECOND);
-- Salida: '1998-01-01 00:00:00'

SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY);
-- Salida: '1998-01-01 23:59:59'

SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND); --
Salida: '1998-01-01 00:01:00'

SELECT DATE_SUB('1998-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND);
-- Salida: 1997-12-30 22:58:59'

SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-- Salida: '1997-12-02'
```

```
SELECT DATE_ADD('1992-12-31 23:59:59.000002', INTERVAL '1.999999'
SECOND_MICROSECOND);
-- Salida: '1993-01-01 00:00:01.000001'

SELECT   TIMESTAMPADD ( MINUTE ,30, now() ) AS DENTRO_TREINTA_MINUTOS;
-- Salida: '2020-03-24 20:15:57'

SELECT id, nombre, apellidos, TIMESTAMPDIFF(YEAR, fecha_nac, now()) Edad FROM
empleados
-- Devolvería la edad de los empleados de la base de datos gesventas.
```

7.5.4.2 Función DATE_FORMAT()

Formatea el valor de tipo fecha según la cadena FORMAT .

Sintaxis

```
DATE_FORMAT(date,format)
```

Para crear la cadena *format* podremos usar los siguientes especificadores:

Especificador	Descripción
%a	Día de semana abreviado (Sun..Sat)
%b	Mes abreviado (Jan..Dec)
%c	Mes, numérico (0..12)
%D	Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd, ...)
%d	Día del mes numérico (00..31)
%e	Día del mes numérico (0..31)
%f	Microsegundos (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minutos, numérico (00..59)
%j	Día del año (001..366)
%k	Hora (0..23)
%l	Hora (1..12)
%M	Nombre mes (January..December)
%m	Mes, numérico (00..12)
%p	AM o PM
%r	Hora, 12 horas (hh:mm:ss seguido de AM o PM)
%S	Segundos (00..59)
%s	Segundos (00..59)
%T	Hora, 24 horas (hh:mm:ss)
%U	Semana (00..53), donde domingo es el primer día de la semana
%u	Semana (00..53), donde lunes es el primer día de la semana
%V	Semana (01..53), donde domingo es el primer día de la semana; usado con %X
%v	Semana (01..53), donde lunes es el primer día de la semana; usado con %x
%W	Nombre día semana (Sunday..Saturday)
%w	Día de la semana (0=Sunday..6=Saturday)
%X	Año para la semana donde domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V
%x	Año para la semana, donde lunes es el primer día de la semana, numérico, cuatro dígitos; usado con %v
%Y	Año, numérico, cuatro dígitos
%y	Año, numérico (dos dígitos)
%%	Carácter '%' literal

Veamos algunos ejemplos

```
--Ejemplos DATE_FORMAT

SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-- Salida: 'Saturday October 1997'

SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
-- Salida: '22 22 10 10:23:00 PM 22:23:00 00 6'

SELECT DATE_FORMAT(NOW(), '%W %d %M %Y');
```

```
-- Salida: 'Tuesday 24 March 2020'
```

7.5.4.3 Función GET_FORMAT()

Retorna una cadena de formato. Esta función es útil en combinación con las funciones DATE_FORMAT() ySTR_TO_DATE().

Sintaxis

```
GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')
```

Tabla de ejemplos

LLamad a función	Resultado
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%S'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

Veamos algunos ejemplos aplicables

```
--Ejemplos GET_FORMAT

SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
'03.10.2003'

SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
'2003-10-31'
```

7.5.4.4 Tabla de Funciones

Sintaxis	Función	Ejemplo	Resultado
ADDDATE(date,INTERVAL expr type), ADDDATE(expr,days)	Incrementa el valor de una fecha Type: DAY, MONTH, YEAR, …	SELECT ADDDATE('1998- 31-02', INTERVAL 31 DAY); SELECT date_add('2012-04-08', INTERVAL 31 MONTH); SELECT ADDDATE('1998-01-02', 31);	'1998-02-02' 2014-11-08 '1998-02-02'

ADDTIME(expr,expr2)	Añade expr2 a expr y retorna el resultado. expr esunaexpresión de fecha u hora y fecha, y expr2es unaexpresión temporal.	SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');	'03:00:01.999997'
CURDATE() , CURRENT_DATE, CURRENT_DATE()	Retorna la fechahoraria actual como valor enformato 'YYYY-MM-DD' o YYYYMMDD	SELECT CURDATE();	'1997-12-15'
CURTIME(), CURRENT_TIME, CURRENT_TIME()	Retorna la hora actual como valor enformato 'HH:MM:SS' o HHMMSS	SELECT CURTIME();	'23:50:26'
NOW(), CURRENT_TIMESTAMP, CURRENT_TIMESTAMP()	Retorna la fecha y hora actual como valor enformato 'YYYY-MM-DD HH:MM:SS' o YYYYMMDDHHMMSS	SELECT NOW();	'1997-12-15 23:50:26'
DATE(expr)	Extrae la parte de fecha de la expresión de fecha o fecha y hora expr.	SELECT DATE('2003-12-31 01:02:03');	'2003-12-31'
DATEDIFF(expr,expr2)	retorna el número de días entre la fechainicial expr y la fecha final expr2.	SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');	1
DATE_ADD(date,INTERVAL expr type), DATE_SUB(date,INTERVAL expr type)	operacionesaritméticas de fechas. date es un valor DATETIME o DATE especificando la fecha de inicio. expr esunaexpresión que especifica el intervalo a añadir o borrar de la fecha de inicio. expr esunacadena; puedecomenzar con un '-' para intervalosnegativos. type esuna palabra clave que indicacómodebeinterpretarse la expresión.	SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);	'1997-12-02'
DAYNAME(date)	Retorna el nombre del día de la semana para date.	SELECT DAYNAME('1998-02-05');	'Thursday'
DAYOFMONTH(date)	Retorna el día del mes para date, en el rango 1 a 31.	SELECT DAYOFMONTH('1998-02-03');	3
DAYOFWEEK(date)	Retorna el índice del día de la semana para date (1 = domingo, 2 = lunes, ..., 7 = sábado)	SELECT DAYOFWEEK('1998-02-03');	3
DAYOFYEAR(date)	Retorna el día del año para date, en el rango 1 a 366.	SELECT DAYOFYEAR('1998-02-03');	34
EXTRACT(type FROM date)	Extrae partes de la fecha en lugar de realizar aritmética de fecha	SELECT EXTRACT(YEAR FROM '1999-07-02'); SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');	1999 199907
HOUR(time)	Retorna la hora para time. El rango del valor de retornoes 0 a 23 para valores de horas del día.	SELECT HOUR('10:05:03');	10
LAST_DAY(date)	Toma una fecha o fecha/hora y retorna el valor correspondiente para el últimodía del mes.	SELECT LAST_DAY('2003-02-05');	'2003-02-28'

MAKEDATE(year,dayofyear)	Retorna una fecha, dado un año y día del año.	SELECT MAKEDATE(2001,31), MAKEDATE(2001,32)	'2001-01-31', '2001-02-01'
MAKETIME(hour,minute,second)	Retorna un valor horario calculado a partir de los argumentos hour, minute, y second .	SELECT MAKETIME(12,15,30);	'12:15:30'
MICROSECOND(expr)	Retorna los microsegundos a partir del a expresión de hora o fecha/hora	SELECT MICROSECOND('12:00:00.123456');	123456
MINUTE(time)	Retorna el minuto de time, en el rango 0 a 59.	SELECT MINUTE('98-02-03 10:05:03');	5
MONTH(date)	Retorna el mes para date, en el rango 1 a 12.	SELECT MONTH('1998-02-03');	2
MONTHNAME(date)	Retorna el nombre completo del mes para date.	SELECT MONTHNAME('1998-02-05');	'February'
QUARTER(date)	Retorna el cuarto del año para date, en el rango 1 a 4.	SELECT QUARTER('98-04-01');	2
SECOND(time)	Retorna el segundo para time, en el rango 0 a 59.	SELECT SECOND('10:05:03');	3
SEC_TO_TIME(seconds)	Retorna el argumento seconds , convertido a horas, minutos y segundos, como un valor en formato 'HH:MM:SS'o HHMMSS,	SELECT SEC_TO_TIME(2378);	'00:39:38'
SUBTIME(expr,expr2)	Resta expr2 de expr y retorna el resultado expr es una expresión de hora o fecha/hora, y expr2 es una expresión de hora.	SELECT SUBTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');	'1997-12-30 22:58:58.999997'
TIME(expr)	Extrae la parte de hora de la expresión hora o fecha/hora expr.	SELECT TIME('2003-12-31 01:02:03');	'01:02:03'
TIMEDIFF(expr,expr2)	TIMEDIFF() retorna el tiempo entre la hora de inicio expr y la hora final expr2. expr y expr2 son expresiones de hora o de fecha/hora, pero ambas deben ser del mismo tipo.	SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001');	'-00:00:00.000001'
TIMESTAMP(expr)	retorna la expresión de fecha o fecha/hora expr como valor fecha/hora.	SELECT TIMESTAMP('2003-12-31');	'2003-12-31 00:00:00'
TIMESTAMP(expr,expr2)	suma la expresión de hora expr2 a la expresión de fecha o de fecha/hora expr y retorna el resultadocomo valor fecha/hora.	SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');	'2004-01-01 00:00:00'
TIMESTAMPADD(interval,int_expr,datetime_expr)	Suma la expresión entera int_expr a la expresión de fecha o de fecha/hora datetime_expr. La unidad for int_expr la da el argumento interval , que debeseruno de lossiguientes valores: FRAC_SECOND, SECOND,MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, o YEAR.	SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');	'2003-01-02 00:01:00'

TIMESTAMPDIFF(interval,datetime_expr1,datetime_expr2)	Retorna la diferenciaentera entre las expresiones de fecha o de fecha/hora datetime_expr1 ydatetime_expr2.	SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');	3
TIME_TO_SEC(time)	Retorna el argumento time convertidoensegundos.	SELECT TIME_TO_SEC('22:23:00');	80580
TO_DAYS(date)	Dada la fecha date, retorna un número de día (el número de dias desde el año 0).	SELECT TO_DAYS('1997-10-07');	729669
UTC_DATE()	Retorna la fecha UTC actual como valor enformato 'YYYY-MM-DD' o YYYYMMDD	SELECT UTC_DATE(), UTC_DATE() + 0;	'2003-08-14', 20030814
UTC_TIME()	Retorna la hora UTC actual como valor enformato 'HH:MM:SS' or HHMMSS	SELECT UTC_TIME(), UTC_TIME() + 0;	'18:07:53', 180753
UTC_TIMESTAMP()	Retorna la fecha y hora UTC actual como valor enformato 'YYYY-MM-DD HH:MM:SS' o YYYYMMDDHHMMSS	SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;	'2003-08-14 18:08:04', 20030814180804
WEEKDAY(date)	Retorna el índice de días de la semana para date (0 = lunes, 1 = martes, ... 6 = domingo).	SELECT WEEKDAY('1998-02-03 22:23:00');	1
WEEKOFYEAR(date)	Retorna la semana de la fechacomonúmero del rango 1 a 53.	SELECT WEEKOFYEAR('1998-02-20');	8

Veamos la siguiente colección de ejemplos:

```
SELECT * FROM empresa.empleados;

SELECT
    id,
    nombre,
    apellidos,
    fecha_nac,
    TIMESTAMPDIFF(YEAR, fecha_nac, NOW()) Edad
FROM
    empleados WHERE ID =1;

-- Salida: 1 John   Smith Perez  1965-10-20   54

SELECT NOW();
-- Salida: '2020-03-26 18:01:39'

SELECT DATE_FORMAT(NOW(), '%W %d %M %Y');
-- Salida: 'Thursday 26 March 2020'

SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
-- Salida: '22 22 10 10:23:00 PM 22:23:00 00 6'

SELECT DATE_FORMAT(NOW(), '%W %d %M %Y');
-- Salida: 'Thursday 26 March 2020'

SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
-- Salida: '03.10.2003'

SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
```

```
-- Salida: '03.10.2003'

SELECT ADDDATE(NOW(), INTERVAL 31 DAY);
-- Salida: '2020-04-26 18:01:06'

SELECT ADDDATE(NOW(), 31);
-- Salida: '2020-04-26 18:03:45'

SELECT ADDTIME(NOW(), '12:00:12');
-- Salida: '2020-03-27 06:04:08'

SELECT CURRENT_TIME;
-- Salida:'18:04:05'

SELECT CURDATE();
-- Salida: '2020-03-26'

SELECT CURRENT_TIMESTAMP;
-- Salida: '2020-03-26'

SELECT DATE(NOW());
-- Salida: '2020-03-26'

SELECT
    id,
    nombre,
    apellidos,
    fecha_nac,
    DATEDIFF(NOW(), fecha_nac) Días
FROM
    empleados;
-- Salida:
--     id     nombre apellidos    fecha_nac    Días
--     1      John   Smith Perez  1965-10-20   19881
--     2      Franklin     Wong Jason   1955-12-20   23473
--     3      Alicia Zelaya Durden1968-07-19   18878
--     4      Jennifer     Wallace Kors 1941-12-25   28581
--     5      Ramesh Narayan Praga1972-05-25   17472
--     6      Joyce  English Balenciaga  1972-07-31   17405
--     7      Ahmad  Jabbar Seokjin      1969-08-06   18495
--     8      James  Borg Park    1937-01-02   30399

SELECT DATE_SUB(NOW(), INTERVAL 31 DAY);
-- Salida: '2020-02-24 18:05:33'

SELECT DAYNAME(NOW());
-- Salida: '2020-02-24 18:05:33'

SELECT DAYOFMONTH('1998-02-03');
-- Salida: 3

-- Americano 1 domingo
SELECT DAYOFWEEK(now());
-- Salida: 5

SELECT DAYOFYEAR(NOW());
-- Salida: 86

SELECT EXTRACT(YEAR FROM NOW());
SELECT YEAR(NOW());
-- Salida: '2020'

SELECT EXTRACT(MONTH FROM NOW());
SELECT MONTH(NOW());
-- Salida: 3

SELECT EXTRACT(YEAR_MONTH FROM NOW());
-- Salida: '202003'
```

```
SELECT HOUR(NOW());
-- Salida: 18

SELECT LAST_DAY(NOW());
-- Salida: '2020-03-31'

SELECT MAKEDATE(2001,58);
-- Salida: '2001-02-27'

SELECT MAKETIME(12,33,45);
-- Salida: '12:33:45'

SELECT MICROSECOND('12:00:00.123456');
-- Salida: '123456'

SELECT MINUTE(NOW());
-- Salida: '9'

SELECT MONTH('1998-02-03');
-- Salida: '2'

SELECT MONTHNAME('1998-02-05');
-- Salida: 'February'

SELECT QUARTER(NOW());
-- Salida: '1'

SELECT SECOND('10:05:03');
-- Salida:
SELECT SEC_TO_TIME(6000);
-- Salida: '1'

SELECT SUBTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
-- Salida: '1997-12-30 22:58:58.999997'

SELECT TIME('2003-12-31 01:02:03');
-- Salida: '1997-12-30 22:58:58.999997'

SELECT TIMEDIFF('11:30:14', '10:00:00');
-- Salida: '1997-12-30 22:58:58.999997'

SELECT TIMESTAMP('2003-12-31');
-- Salida:
SELECT TIMESTAMP(NOW(), '10:30:50');
-- Salida: '1997-12-30 22:58:58.999997'

SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-- Salida:
SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-- Salida: '3'

SELECT
    id,
    nombre,
    apellidos,
    TIMESTAMPDIFF(YEAR, fecha_nac, NOW()) edad
FROM
    empleados;
-- Salida:
-- 1   John   Smith Perez  54
-- 2   Franklin     Wong Jason   64
-- 3   Alicia Zelaya Durden51
-- 4   Jennifer     Wallace Kors 78
-- 5   Ramesh Narayan Praga47
-- 6   Joyce  English Balenciaga  47
-- 7   Ahmad  Jabbar Seokjin      50
-- 8   James  Borg Park    83
```

```
SELECT TIME_TO_SEC('22:23:00');
-- Salida: '80580'

SELECT TO_DAYS('1997-10-07');
-- Salida: '729669'

SELECT UTC_DATE() +5;
-- Salida: '20200331'

SELECT UTC_TIME() + 1 ;
-- Salida: '171203'

SELECT UTC_TIMESTAMP() + 1;
-- Salida: '20200326171211'

SELECT WEEKDAY('1998-02-03 22:23:00');
-- Salida: '1'

SELECT WEEKOFYEAR('1998-12-31');
-- Salida: '53'

-- Base de datos: maratoon
-- Corredores que cumplen años el próximo mes

SELECT
    id,
    nombre,
    apellidos,
    fechaNacimiento,
    edad,
    MONTH(fechaNacimiento) mes
FROM
    corredores
WHERE
    MONTH(NOW()) + 1 = MONTH(fechaNacimiento)
ORDER BY id;
-- Salida:
```

7.6 Importación y Exportación de datos en MySQL

A veces nos interesará exportar datos a otras bases de datos o ficheros, o insertar grandes cantidades de registros procedentes de un fichero de datos o de un fichero que incluye sentencias ***INSERT*** o ***REPLACE***. Para exportar disponemos del programa de copia de seguridad mysqldump y de la sentencia SELECT, mientras que para la inserción de datos disponemos de la sentencia ***LOAD DATA*** y el comando ***source***, que nos permite ejecutar ficheros o comandos o instrucciones SQL.

7.6.1 LOAD DATA

Es una cláusula SQL para insertar datos a gran velocidad a partir de un fichero de texto plano con cierto formato.

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
    ]
    [LINES
        [STARTING BY '']
        [TERMINATED BY '\n']
    ]
    [IGNORE number LINES]
    [(col_name,...)]
```

Donde:

- ***LOW_PRIORITY***: indica que su ejecución se realizará cuando no haya clientes leyendo la tabla. Solo sirve para el caso de tablas MyISAM.
- ***CONCURRENT***: permite que otros procesos o clientes accedan a la lectura de datos de la tabla en la que tienen lugar las inserciones.
- ***LOCAL***. Hace que el fichero especificado sea leído desde el cliente que realiza la conexión. En caso de no estar presente el servidor entiende que debe leer el fichero en el propio servidor. La ruta al fichero debe ser absoluta o relativa en cuya caso se toma como directorio base el de la base de datos activa (para Windows debe usarse doble barra en la especificación de ruta).
- ***INFILE*** 'fichero': indica el fichero que contiene los datos
- ***REPLACE|IGNORE***: hacen que en caso de existir el valor de una clave ésta se actualice (REPLACE) o se omita continuando con el resto de inserciones (IGNORE)
- ***LINES STARTING BY***: indica que las líneas del fichero contiene los datos comienzan por la cadena especificada.
- ***LINES TERMINATED BY***: indica la cadena que hay al final de cada línea en el fichero. Sirve para delimitar los registros de datos.
- ***FIELDS|COLUMNS***: es lo mismo que LINES. Sirve para delimitar el final de cada campo de datos.
- ***OPTIONALLY ENCLOSED BY***: indica que los campos pueden estar delimitados por un carácter como comillas dobles.
- ***FIELDS ESCAPED BY***: indica el carácter que se usará para "escapar" los caracteres de manera que no se confundan con los usados para delimitar campos, indicar el separador de campos y el carácter fin de línea. Por ejemplo, algunos campos pueden estar delimintados por comillas dobles y al mismo tiempo puede haber campos que contengan comillas como parte del valor del campo. Si no hubiese carácter de escape MySQL entendería las comillas como el final del campo, cuando no es así.
- ***IGNORE***. Sirve para especificar el número de líneas a omitir al principio del fichero. Sirve para cuando hay cabeceras o datos que no corresponden con campos de las tablas.

Los valores por defecto, si omitimos las clausulas FIELDS y LINES son equivalentes a lo siguiente:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY '';
```

Veamos del siguiente ejemplo:

Disponemos de un fichero con datos ***llegadasubrique.txt*** para la tabla **registro** de la base de datos **maratoon**. Cada fila o registro está separado por un salto de línea y cada campo por un punto y coma.

```
--EjemplosLOAD DATA

LOAD DATA INFILE 'C:\\llegadasubrique.txt' IGNORE INTO TABLE
maratoon.registro FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n';
```

El contenido del fichero ***llegadasubrique.txt*** sería:

```
NULL; 2; 16; 2012-4-11 10:00:00.00000; 2012-4-11 11:09:10.00000; NULL
NULL; 2; 15; 2012-4-11 10:00:00.00000; 2012-4-11 11:09:12.00000; NULL
NULL; 2; 13; 2012-4-11 10:00:00.00000; 2012-4-11 11:09:17.00000; NULL
NULL; 2; 12; 2012-4-11 10:00:00.00000; 2012-4-11 11:09:20.00000; NULL
```

Una vez ejecutado el script comprobamos que efectivamente en la tabla registros de la base de datos maratoon, se han añadido 4 nuevos registros con los datos importados del archivo llegadasubrique.txt.

7.6.2 LOAD XML

Este comando permite importar datos de un fichero XML a una tabla de una determinada base de datos.

La sintaxis es la siguiente:

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE [db_name.]tbl_name
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<tagname>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Las opciones son las mismas que el resto de comandos salvo ***ROWS IDENTIFIED BY '<tagname>'***, que sirve para indicar la etiqueta que iniciará cada fila de la tabla.

Este comando soporta tres formatos XML:

- **Formato 1.**

El formato *xml* se compone de filas que comienzan por *row*, cada columna y sus valores aparecen en forma de*aributo=valor*:

```
<ROW column1="value1" column2="value2" .../>
```

- **Formato 2.**

Las columnas aparecen como etiquetas y sus valores como el contenido de las mismas. Este es el formato que usa por defecto MYSQL.

```
<row>
        <column1>value1</column1>
        <column2>value2</column2>
</row>
```

- **Formato 3.**

Los nombres de las columnas son los valores de los atributos de las etiquetas 'field' y el contenido son valores de dichas etiquetas:

```
<row>
        <fieldname='column1'>value1</field>
        <fieldname='column2'>value2</field>
```

```
</row>
```

Veamos el siguiente ejemplo:

Se desean importar los datos contenidos en el archivo ***corredores.xml*** hacia la tabla ***corredores*** de la base de datos ***maratoon*** del tema 6.

Veamos en primer lugar el contenido en formato *xml* del archivo ***corredores.xml*** :

```
-- corredores.xml

<regcorredores>
	<Nombre>Diego</Nombre>
	<Apellidos>Moreno García</Apellidos>
	<Ciudad>Ubrique</Ciudad>
	<FechaNacimiento>1960-6-5</FechaNacimiento>
	<Sexo>H</Sexo>
	<Edad>null</Edad>
	<Categoria_id>1<Ccateogoria_id>
	<Club_id>1</Club_id>
</regcorredores>
<regcorredores>
	<Nombre>Sergio</Nombre>
	<Apellidos>Gallardo Galindo</Apellidos>
	<Ciudad>Ubrique</Ciudad>
	<FechaNacimiento>1960-6-5</FechaNacimiento>
	<Sexo>H</Sexo>
	<Edad>null</Edad>
	<Categoria_id>3</Categoria_id>
	<Club_id>1</Club_id>
</regcorredores>
<regcorredores>
	<Nombre>Marta</Nombre>
	<Apellidos>Panal Valle</Apellidos>
	<Ciudad>Ubrique</Ciudad>
	<FechaNacimiento>1960-6-5</FechaNacimiento>
	<Sexo>M</Sexo>
	<Edad>null</Edad>
	<Categoria_id>3</Categoria_id>
	<Club_id>1</Club_id>
</regcorredores>
```

A continuación usamos el comando LOAD XML para importar los datos:

```
--Ejemplos LOAD XML

LOAD XML INFILE 'C:\\corredores.xml' INTO TABLE maratoon.corredores ROWS
IDENTIFIED BY '<regcorredores>';
```

7.6.3 Exportación de Datos

Para la exportación de datos se usa la cláusula SELECT con la siguiente sintaxis

```
[INTO OUTFILE 'file_name' export_options]
```

En cuanto a las opciones de exportación son las mismas que para las cláusulas FIELDS y LINES de LOAD DATA. Su sintaxis es:

```
[FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
 ]
[LINES
        [STARTING BY '']
        [TERMINATED BY '\n']
]
```

La cláusula FIELDS se refiere a las opciones de cada columna:

- TERMINATED BY 'carácter': nos permite elegir el carácter delimitador que se usará para separar cada columna. Por defecto, el valor que se usa es el tabulador, pero podemos usar ';', ',', etc.
- [OPTIONALLY] ENCLOSED BY 'carácter': sirve para elegir el carácter usado para entrecomillar cada columna. Por defecto no se entrecomilla ninguna columna, pero podemos elegir cualquier carácter. Si se añade la palabra OPTIONALLY sólo se entrecomillarán las columnas de texto y fecha.
- ESCAPED BY 'carácter': sirve para indicar el carácter que se usará para escapar aquellos caracteres que pueden dificultar la lectura posterior del fichero. Por ejemplo, si terminamos las columnas con ',' y no las entrecomillamos, un carácter ',' dentro de una columna de texto se interpretará como un separador de columnas. Para evitar esto se puede escapar esa coma con otro carácter. Por defecto se usa el carácter '\'.

La cláusula LINES se refiere a las opciones para cada fila:

- STARTING BY 'carácter': permite seleccionar el carácter para comenzar cada línea. Por defecto no se usa ningún carácter para ello.
- TERMINATED BY 'carácter': permite elegir el carácter para terminar cada línea. Por defecto es el retorno de línea, pero se puede usar cualquier otro carácter o caracteres, por ejemplo '\r\n'.

Veamos el siguiente ejemplo:

Deseamos exportar la tabla ***corredor*** de la base de datos ***maratoon*** en formato ***csv***, usando el carácter ";" como separador de campo y el carácter "\n" salto de línea como final de registro. El fichero que contendrá los datos será ***corredores.csv***:

Usamos entonces el comando **SELECT … INTO FILE** para realizar la importación:

```
SELECT * FROM maratoon.corredores
INTO OUTFILE 'd:\corredores.csv'
FIELDS TERMINATED BY ';'
      OPTIONALLY ENCLOSED BY '\"'
LINES TERMINATED BY '\n';
```

A partir del comando anterior obtendríamos el fichero ***corredores.csv*** creado en la carpeta raíz de la unidad d: de nuestro sistema. En caso de ya existir el fichero corredores.csv daría error de sistema. Una vez creado el archivo el contenido sería el siguiente:

```
1;"Juan";"García Pérez";"Ubrique";"1965-07-31";"H";50;7;1
2;"Juan José";"Pérez Morales";"Ubrique";"1945-08-30";"H";70;8;1
3;"Eva";"Rubiales Alva";"Ubrique";"1980-08-25";"M";35;5;1
4;"Josefa";"Rios Pérez";"Villamartín";"1990-10-15";"M";25;4;2
5;"Pedro";"Ortega Ríos";"Villamartín";"1994-05-14";"H";21;4;2
6;"Francisco";"Morales Almeida";"Villamartín";"1970-02-01";"H";46;6;2
7;"Macarena";"Fernández Pérez";"Villamartín";"1980-05-03";"M";35;5;2
8;"Jesús";"Romero Reguera";"Villamartín";"1970-06-05";"H";45;6;2
9;"Pedro";"García Ramírez";"Ubrique";"1967-07-31";"H";48;6;1
10;"María";"Pérez Moreno";"Ubrique";"1975-08-30";"M";40;6;1
11;"Almudena";"Romero Alva";"Arcos";"1986-08-25";"M";29;4;4
12;"Francisco";"Pérez Amor";"Arcos";"1992-10-15";"H";23;4;4
13;"Juan";"Rodríguez Ríos";"Ubrique";"1978-05-14";"H";37;5;1
14;"Cristina";"García Almeida";"Villamartín";"1978-02-01";"M";38;5;2
15;"Romira";"Jiménez Pérez";"Arcos";"1984-05-03";"M";31;5;4
16;"José";"Rincón Pérez";"Villamartín";"1960-06-05";"H";55;7;2
17;"JJose";"Moreo Pérez";"Ubrique";"1960-06-05";"H";55;7;3
```

7.6.4 Mysqldump.

El método más utilizado para crear copias de seguridad de MySQL se basa en el uso del comando mysqldump. Este comando se incluye dentro de las utilidades del propio servidor MySQL.

Esta utilidad se ha de utilizar en modo comando y desde la carpeta donde se encuentra instalada la utilidad. Si tenemos MySQL instalado mediante XAMPP la ruta sería C:\XAMPP\MYSQL\BIN.

La sintaxis de mysqldump es:

```
Mysqldump [opciones] nombrebasedatos [tablas]
Mysqldump [opciones] --databases DB1 [DB2 DB3 …]
Mysqldump [opciones] --all-databases
```

Veamos el siguiente ejemplo:

- Realizar una copia de seguridad de la base de datos maratoon, especificando en el comando ***mysqldump*** el nombre del servidor con *–h*, el nombre del usuario con *–u* y la contraseña con el parámetro *–p*. Añadir el parámetro –t para indicar que sólo se copiarán los datos y no la estructura de las tablas. Con el siguiente comando antes de realizar la copia de seguridad solicita al usuario la contraseña:

```
C:\xampp\mysql\bin>mysqldump -h localhost -u root -p -t maratoon>
d:maratoon.sql
```

- Realizar una copia de seguridad de la base de datos maratoon y lineasventas:

```
C:\xampp\mysql\bin>mysqldump -h localhost -u root -p -t --databases
maratoon lineasventas > d:copialocalhost.sql
```

- Si lo que deseo es realizar una copia de seguridad de todas las bases de datos:

```
C:\xampp\mysql\bin>mysqldump –h localhost –u root –p –t --all-
databases>h:localhost.sql
```

Exportar en XML

Una de las grandes utilidades de mysqldump también es exportar la base de datos en formato xml.Veamos los siguientes ejemplos:

- Obtener el archivo maratoon.xml con la base de datos completa maratoon

```
C:\xampp\mysql\bin>mysqldump –h localhost –u root –p --xml maratoon>
d:maratoon.xml
```

- Para obtener el documento maratoon.xml sólo con los datos, sin la estructura xml de las distintas tablas de la base de datos usaríamos el parámetro *–t*:

```
C:\xampp\mysql\bin>mysqldump –h localhost –u root –p –t --xml maratoon>
d:maratoon.xml
```

- En cambio si quisiéramos sólo la estructura de las distintas tablas de la base de datos maratoon, sin los datos, usuaríamos el parámetro *--no-data*

```
C:\xampp\mysql\bin>mysqldump –h localhost –u root –p --no-data --xml
maratoon> d:maratoon.xml
```

- Ahora si qusiéramos exportar en formato xml todos los datos de la tabla corredores de la base de datos maratoon

```
C:\xampp\mysql\bin>mysqldump –h localhost –u root –p -t --xml maratoon
corredores> d:maratoon.xml
```

7.7 MySQL en modo línea de comando

Es muy normal que utilicemos MySQL a través de páginas PHP y para administrar la base de datos utilicemos un programa como PhpMyAdmin o como es nuestro caso el MySQLWorkbench, pero a veces no nos queda otro remedio que acceder a la base de datos a través de la línea de comandos.

MySQL tiene un programa, que se llama con el mismo nombre de la base de datos (mysql) que sirve para gestionar la base datos por línea de comandos. Ese programa, en una instalación de Xamppara Windows se encuentra en un directorio como C:\xampp\mysql\bin.

Conectar con el servidor MySQL

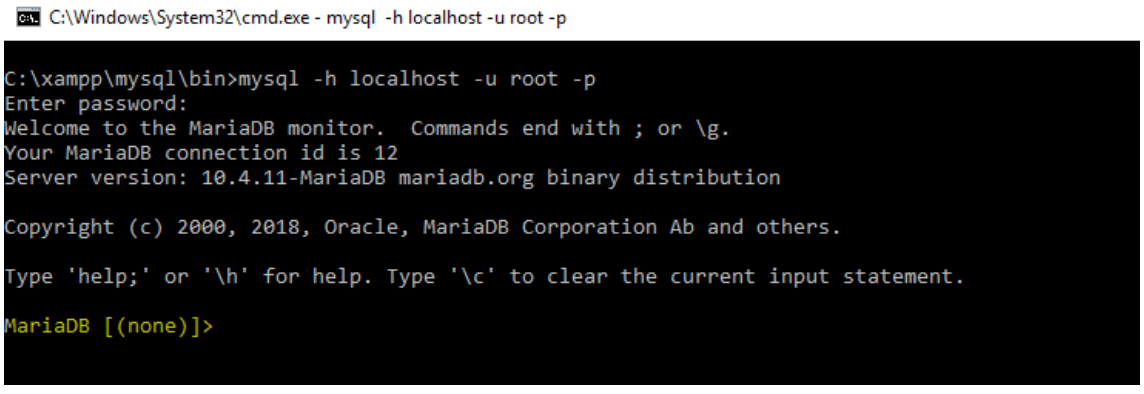
Lo primero que tendremos que hacer es conectar con el sistema gestor de MySQL. Para ello, desde la línea de comandos invocamos a MySQL indicándoles las opciones de conexión a nuestro servidor localhost correspondiente

```
mysql -h localhost -u root -p
```

Lo primero que nos preguntará será el password para el usuario root. Una vez introducida la clave, ya estaremos dentro de la línea de comandos de MySQL. Con ello el prompt cambiará a algo como esto:

```
MariaDB [(None)]>
```

Quedando el modo comando de la siguiente forma



Ahora dentro de la consola de MySQL puedo ejecutar cualquier comando SQL acabándolo siempre con ";". Para finalizar la sesión uso la sentencia *QUIT*.

Veamos la siguiente secuencia de comandos:

```
MariaDB [(none)]> use maratoon;
Databasechanged
MariaDB [maratoon]> show tables;
+--------------------+
| Tables_in_maratoon |
+--------------------+
| carrera            |
```

```
| categoria          |
| club               |
| corredor           |
| registro           |
+--------------------+
5 rows in set (0.00 sec)
MariaDB [maratoon]>select * from carrera;
+------------+----------------------+------------------+-----------+----------------+
| CodCarrera | Nombre               | Ciudad           | Distancia | MesCelebracion |
+------------+----------------------+------------------+-----------+----------------+
|          1 | Nutrias Pantaneras   | Ubrique          |     11000 |             10 |
|          2 | Media Martoon Sevilla | Sevilla          |     22000 |              6 |
|          3 | Media Martoon Jerez  | Jerez            |     22000 |              5 |
|          4 | Salida Sanluqueña    | Sanlucar de Bda. |     11500 |              3 |
|          5 |  2                   |  16              |         0 |              0 |
|          6 |  1                   |  15              |         0 |              0 |
+------------+----------------------+------------------+-----------+----------------+
6 rows in set (0.00 sec)

MariaDB [maratoon]>quit;
```