Tema 9

Usuarios, Vistas y Transacciones

Índice

1. Usuarios y privilegios 1
   1.1. Especificación de cuentas de usuario 1
   1.2. Crear cuentas de usuarios: create user 3
   1.3. Eliminar usuarios: drop user 4
   1.4. Renombrar usuarios: rename user 4
   1.5. Modificar usuarios: alter user 4
   1.6. Cambiar la contraseña a un usuario: set password 4
   1.7. Otorgar privilegios: grant 5
   1.8. Niveles de privilegios 6
   1.9. Eliminar privilegios: revoke 8
   1.10. Límites de recursos por cuenta 8
   1.11. Gestión de los usuarios conectados a la base de datos. 9
2. Vistas 11
   2.1. Crear vistas 12
   2.2. Modificar vistas 13
   2.3. Renombrar vistas 13
   2.4. Borrar vistas 13
   2.5. Consultar vistas 13
3. Transacciones 14
4. Bloqueos 17

1. Usuarios y privilegios

La gestión de usuarios *MariaDB/MySQL* permite administrar el acceso al *SGDB* y establecer las acciones que un usuario puede realizar dentro del *SGBD*.

Se basa de dos elementos:

- Cuenta de usuario y contraseña: permite la conexión a la base de datos cuanto existe la cuenta del usuario especificada y la contraseña es válida. Una cuenta de usuario viene definida por dos elementos separados por una arroba: un nombre de usuario y un equipo desde el que se realiza la conexión. Cuentas de usuario como *root@localhost*, *root@192.168.1.10* y *root@example.com* son cuentas de usuario distintas, comparten nombre de usuario pero no equipo de conexión, con contraseña y privilegios propios. Cuando hablemos de un usuario normalmente nos estaremos refiriendo una cuenta de usuario.
- Privilegios: permiten especificar las acciones que un usuario puede realizar sobre los distintos elementos de la base de datos. Es recomendable conceder privilegios de forma restrictiva.

El usuario *root@localhost* es el administrador de la base de datos. Este usuario tiene concedidos, por defecto, todos los privilegios. Es equiparable a usuario *root* en *Linux*.

Los usuarios se almacenan en la tabla *user* de la base de datos *mysql*. Las columnas relevantes, para nosotros, en esta tabla, son *user* y *host*. La columna *password* almacena el hash de la contraseña del usuario. Para consultar los usuarios disponibles se puede usar el comando:

```
SELECT user, host FROM mysql.user;
```

Para la administración de usuarios *se* dispone de los siguientes comandos[1] comunes tanto a *MariaDB* como a *MySQL*. Otros comandos, como pueden ser los roles, son propios de *MariaDB y* se verán en un punto propio:

COMANDO	ACCIÓN
CREATE USER	Crea un usuario nuevo en la base de datos
DROP USER	Elimina un usuario de la base de datos
RENAME USER	Permite especificar un nuevo nombre de usuario a un usuario existente
ALTER USER	Permite modificar un usuario existente
SET PASSWORD	Establece la contraseña de un usuario
GRANT	Permite asignar privilegios y crear usuarios
REVOKE	Elimina privilegios de un usuario

1.1. Especificación de cuentas de usuario

Los nombres de cuentas de usuario deben cumplir las siguientes reglas:

- Su formato es *nombreDeUsuario@nombreDeEquipo*.
- Nombre de equipo se puede definir usando un nombre de equipo, un dominio, una *ip* o una red.
- En el nombre de equipo se pueden usar los comodines % (cualquier cadena de caracteres) y _ (un único carácter) con el mismo significado que tienen con el operador *like*.
- Si se usan caracteres especiales, ya sea en el nombre de usuario como en el nombre de equipo, se deben poner entrecomillados. Ejemplos de caracteres especiales son los espacios, guiones *(-)* o en el uso de los comodines % y _ en el nombre del equipo.

[1] https://mariadb.com/kb/en/library/account-management-sql-commands/
https://dev.mysql.com/doc/refman/5.7/en/account-management-sql.html

- En caso de entrecomillarse se ha de hacer por separado. Por ejemplo: *'usu-ario'@'192.168.1.%'*. Si se hace de manera conjunta se crearía el usuario *'usu-ario@192.168.1.%'@''*.
- Si solo se especifica el nombre del usuario se permite la conexión de ese usuario desde cualquier equipo. Por ejemplo creando el usuario *jose* internamente se crea el usuario *jose@''*.
- Si no se especifica el nombre de usuario se crea un usuario anónimo. Por ejemplo el usuario *''@127.0.0.1* que permitirá la conexión de cualquier usuario desde la *ip 127.0.0.1*.

La siguiente tabla muestra ejemplos de cuentas de usuario válidas.

Usuario	Host	Cuentas de usuario
juan	vivas.edu.es	El usuario *juan* conectando desde *vivas.edu.es*
''	vivas.com	El usuario anónimo conectando desde *vivas.com*
jose	'%'	El usuario *jose*, conectando desde cualquier equipo
''	''	El usuario anónimo conectando desde cualquier equipo
pablo	'%.edu.es'	El usuario *pablo*, conectando desde cualquier equipo en el dominio *edu.es*
pedro	'x.y.%'	El usuario *pedro*, conectando desde el subdominio *x.y* de cualquier dominio. Por ejemplo: *x.y.google.com*, *x.y.com*, *x.y.edu*, ...
ana	144.155.166.177	El usuario *ana*, conectando desde el equipo con la *IP* especificada
laura	'14.15.1.%'	El usuario *laura*, conectando desde cualquier equipo en la subred *14.15.1*
paula	'144.155.166.0/255.255.255.0'	Idéntico al ejemplo anterior

MariaDB/MySQL dispone de los siguientes comandos para obtener el usuario actual:

- User(): es el usuario con el que establecemos la conexión a la base de datos. Si este usuario existe en la base de datos el valor de este comando será igual a Current_user(). Sinónimos de User() son SESSION_USER() y SYSTEM_USER().
- Current_user(): es el usuario que se valida al realizar la conexión a la base de datos y que determina los privilegios del usuario.

El siguiente ejemplo ilustra las posibles diferencias entre estos dos comandos. Imaginemos que en una base de datos disponemos de los siguientes usuarios:

```
+------+-----------+
| user | host      |
+------+-----------+
| root | 127.0.0.1 |
|    % | 127.0.0.1 |
+------+-----------+
```

Si se realiza la conexión desde *127.0.0.1* con el usuario *prueba@127.0.0.1*, se permite la conexión aunque el usuario no exista puesto que existe el usuario *'%'@127.0.0.1* que permite que cualquier usuario se conecte desde la *ip 127.0.0.1*. Este es el usuario que internamente se usará para establecer la conexión.

Según lo anterior, la consulta:

```
SELECT USER(), CURRENT_USER(), CURRENT_USER, SESSION_USER(), SYSTEM_USER();
+------------------+----------------+--------------+------------------+------------------+
| USER()           | CURRENT_USER() | CURRENT_USER | SESSION_USER()   | SYSTEM_USER()    |
+------------------+----------------+--------------+------------------+------------------+
| prueba@localhost | @localhost     | @localhost   | prueba@localhost | prueba@localhost |
+------------------+----------------+--------------+------------------+------------------+
```

Se puede observar que el valor de *User()* es el usuario que introducimos en el cliente para conectarnos a la base de datos y *Current_user()* es el usuario con el que nos validamos en la base de datos y del cual se obtienen los privilegios.

1.2. Crear cuentas de usuarios: create user

Mediante CREATE USER se pueden añadir nuevas cuentas de usuario a la base de datos.

Su sintaxis básica es:

```
CREATE [OR REPLACE] USER [IF NOT EXISTS]
        cuentaDeUsuario1 [ IDENTIFIED BY [ PASSWORD ] 'contraseña1' ]
      [, cuentaDeUsuario2 [ IDENTIFIED BY [ PASSWORD ] 'contraseña2' ]
        ..........................
      ]
      [WITH resource_option [, resource_option] ...]
```

Donde:

- Se pueden crear varias cuentas de usuario a la vez separando sus definiciones por comas.
- *or replace:* si el usuario existe se elimina y se vuelve a crear. No se puede usar con *if not exists.*
- *If not exists* permite crear el usuario solo si no existe. En otro caso genere un aviso en vez de un error. No se puede usar con *or replace*.
- La clausula *IDENTIFIED BY* permite establecer la contraseña del usuario. Si se omite el usuario no tendrá contraseña lo cual es inseguro y no recomendado.
- Si a *IDENTIFIED BY* se le añade palabra *PASSWORD* la contraseña se ha de indicar como el valor *hash* de la contraseña. Este valor es el devuelto por la función *PASSWORD(pass)* donde *pass* es la contraseña que se quiere asignar.
- Cuando se crea un usuario se le asigna, por defecto, el privilegio *Usage*. Este permiso es sinónimo de "*no privileges"* y solo permite realizar la conexión a la base de datos.
- *resource_option*: se verá su utilización y sus posibles valores en el punto 1.10.

Por ejemplo, para especificar la contraseña de un usuario, durante su creación, tenemos las dos opciones siguientes:

- Texto plano

```
CREATE USER IF NOT EXISTS prueba@'example.com' IDENTIFIED BY 'miPass';
```

- Hash con la contraseña

```
SELECT PASSWORD('miPass') ;
+-------------------------------------------+
| PASSWORD('miPass')                        |
+-------------------------------------------+
| *3F713811F2497FD7489B537AD709A30F79138586 |
+-------------------------------------------+
CREATE OR REPLACE USER prueba2@'example.com' IDENTIFIED BY PASSWORD
                                        '*3F713811F2497FD7489B537AD709A30F79138586';
```

Para ver la sentencia de creación de un usuario se puede usar *SHOW CREATE USER cuentaDeUsuario*. Por ejemplo:

```
SHOW CREATE USER prueba@'example.com';
```

Mediante el comando *SHOW GRANTS* se puede consultar la lista de privilegios que posee un usuario.

- Para el usuario actual:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
```

- Para un usuario determinado:

```
SHOW GRANTS FOR cuentaDeUsuario;
```

1.3. Eliminar usuarios: drop user

Para eliminar uno o más cuentas de usuario de *MariaDB/MySQL* se utiliza el comando DROP USER.

Su formato es:

```
DROP USER [IF  EXISTS] cuentaDeUsuario1 [, cuentaDeUsuario2, ... ]
```

Actualmente al borrar un usuario se borran, de forma automática, todos los privilegios que tuviese asignados (en versiones anteriores a *MySQL 5.0.2* antes de borrar un usuario había que borrar todos los privilegios que tuviese asignados).

Cuanto se borra una cuenta de un usuario que está conectado en la base de datos la cuenta no se borra hasta que el usuario se desconecta (este usuario no es desconectado de forma automática).

Un ejemplo de su uso es:

```
DROP USER prueba@'example.com';
```

1.4. Renombrar usuarios: rename user

El comando RENAME USER permite renombrar una cuenta de usuario existente con un nuevo nombre. Todos los privilegios de la antigua cuenta de usuario permanecen en la nueva.

Su formato es:

```
RENAME USER cuentaDeUsuario1 TO usuarioNuevo1 [ , cuentaDeUsuario2 TO usuarioNuevo2, ... ]
```

La cuenta de usuario designada por el nuevo nombre no puede existir en la base de datos.

1.5. Modificar usuarios: alter user

La instrucción ALTER USER permite modificar usuarios ya creados.

Su sintaxis básica es:

```
ALTER USER [IF  EXISTS]
          cuentaDeUsuario1 [ IDENTIFIED BY [ PASSWORD ] 'contraseña1' ]
        [, cuentaDeUsuario2 [ IDENTIFIED BY [ PASSWORD ] 'contraseña2' ]
          ..........................
        ]
        [ WITH resource_option [, resource_option] ... ]
```

Los distintos elementos tienen el mismo significado que en la instrucción Create User.

Ejemplos de su utilización son:

```
ALTER USER prueba@example.com IDENTIFIED BY 'contraseña4';

ALTER USER prueba@example.com  WITH MAX_CONNECTIONS_PER_HOUR 20;
```

1.6. Cambiar la contraseña a un usuario: set password

Para cambiar la contraseña de un usuario creado se disponen de las siguientes alternativas:

- SET PASSWORD. Existen dos opciones:
  - Asignar contraseña al usuario actual:

    ```
    SET PASSWORD = PASSWORD(''nuevaContraseña'');
    ```

  - Asignar contraseña a un usuario mediante su cuenta de usuario:

    ```
    SET PASSWORD FOR usuario@equipo = PASSWORD('nuevaContraseña');
    ```

- ALTER USER. Entre otras opciones *alter user* permite establecer una nueva contraseña:

  ```
  ALTER USER usuario@equipo IDENTIFIED BY 'password';
  ```

- Editar la tabla *mysql.user*. Se puede actualizar directamente la tabla de usuarios mediante la instrucción *update*.

  Para que las modificaciones tengan efecto se tiene que indicar a *MariaDB/MySQL* que vuelva a leer las tablas de privilegios cargándolas en memoria mediante el comando es *FLUSH PRIVILEGES*.

  Un ejemplo de esta alternativa es:

```
UPDATE mysql.user SET Password=PASSWORD('newPass') WHERE  User='usuario' AND Host='equipo';
FLUSH PRIVILEGES;
```

1.7. Otorgar privilegios: grant

El comando GRANT permite asignar privilegios a un usuario. Los privilegios determinan las acciones que un usuario tiene permitido realizar en la base de datos.

Para poder otorgar un privilegios se ha de tener tanto el privilegio que se quiere otorgar como el privilegio GRANT OPTION.

Se debe tener en cuenta que el orden de los distintos elementos es fijo y que la lista de permisos, el objeto de la base de datos y la cuenta de usuario son obligatorios.

Su sintaxis es la siguiente:

```
GRANT permiso1 [(columnas), [permiso2[(columnas)], ...]
        ON [tipoObjeto] objetoDeLaBD
        TO cuentaDeUsuario1 [IDENTIFIED BY [PASSWORD] 'clave']
           [ , cuentaDeUsuario2 [IDENTIFIED BY [PASSWORD] 'clave']
           ..................................
        [WITH {GRANT OPTION | resource_option [, resource_option, ... ]} ...]
```

Donde:

- permiso1, permiso2: es el permiso o permisos que se quiere asignar a un objeto de la base de datos para un usuario o varios. Como mínimo se debe asignar un permiso pero se pueden asignar varios de forma simultánea.
- Columnas: Columna o columnas a la que se otorgan los privilegios. Si se especifica más de una columna estas se han de separar por comas. No es obligatorio.
- tipoObjeto: es el tipo de objeto de la base de datos de *objetoDeLaBd*. Sus posibles valores son: *table*, *function* y *procedure*. No es obligatorio.
- objetoDeLaBD: es el elemento de la base de datos al que se quiere conceder los permisos. Este elemento puede ser: una base de datos, una tabla o un procedimiento almacenado. Ver el punto siguiente: niveles de privilegios.
- cuentaDeUsuario: Es el usuario al que se le van otorgar los privilegios. Si la cuenta de usuario a la que se otorga privilegios no existe se crea. Se pueden otorgar los privilegios a más de un usuario de forma simultánea separando su definición por comas.
- IDENTIFIED BY 'clave': Es la contraseña que se especifica para el usuario que se va a crear. Al igual que con CREATE USER y ALTER USER se puede usar PASSWORD para introducir el HASH de la contraseña.
- WITH GRANT OPTION: otorga el privilegio de otorgar y quitar privilegios a un usuario. Un usuario con el privilegio *grant* solo puede otorgar los privilegios que ya tiene. Para convertir un usuario normal en administrador de una base de datos concreta se pueden utilizar los cualquiera de los comandos siguientes:
  - GRANT all on baseDeDatos.* TO nuevoAdmin@localhost WITH GRANT OPTION;
  - GRANT all, grant option on baseDeDatos.* TO nuevoAdmin@localhost;
- resource_option: se verá su utilización y sus posibles valores en el punto 1.10.

Ejemplos de uso de *grant*:

- Asignar el privilegio *select* sobre todas las tablas del *tema9* al usuario *prueba@localhost*:

```
GRANT select ON tema9.* TO prueba@localhost;
```

- Asignar los privilegios para seleccionar y borrar datos en la tabla *teléfonos* del *tema9* a los usuarios *uno@localhost* y *dos@locahost*. Además a dos se le establece su contraseña a *clave*.

```
GRANT insert, delete ON tema9.telefonos  TO uno@localhost IDENTIFIED BY 'clave', dos@localhost;
```

Se pueden otorgar privilegios tanto sobre bases de datos como sobre tablas que aun no existen.

Los privilegios para una tabla o columna se forman de forma aditiva como un *OR* lógico de los privilegios en cada uno de los cuatro niveles de privilegios siguientes:

```
global privileges
OR database privileges
OR table privileges
OR column privileges
```

1.8. Niveles de privilegios

Se pueden otorgar los siguientes niveles de privilegios (algunos privilegios solo pueden ser de un nivel):

- Privilegios globales: Se aplican a todas las bases de datos del servidor. Se almacenan en la tabla mysql.user. Su sintaxis es *.*

```
GRANT SELECT, INSERT ON *.* TO nonbreUsuario@nombreEquipo;
```

  El usuario indicado puede consultar e insertar datos en todas las tablas de todas la bases de datos.

- Privilegios de base de datos: Se aplican para todos los objetos dentro de una base de datos. Se almacenan en la tabla mysql.db. Su sintaxis es *nombreBaseDeDatos.**

```
GRANT SELECT, INSERT ON tema9.* TO nonbreUsuario@nombreEquipo;
```

  El usuario indicado puede consultar e insertar datos en todas las tablas de la base de datos *tema9*.

- Privilegios de tabla: se aplican a todas las columnas en una tabla. Se almacenan en la tabla mysql.tables_priv. Su sintaxis es *nombreBaseDeDatos.nombreTabla*. Si omitimos el nombre de la base de datos los permisos se otorgan a la tabla *nombreTabla* de la base de datos actual.

```
GRANT SELECT, INSERT ON tema9.telefonos TO nonbreUsuario@nombreEquipo;
```

  El usuario indicado puede consultar e insertar datos en la tabla *telefonos* de la base de datos *tema9*

- Privilegios de columna: se aplican a una columna de una tabla. Se almacenan en la tabla mysql.columns_priv. De deben especificar como un privilegio seguido de una o más columnas entre paréntesis.

```
GRANT SELECT (id), INSERT (id, telefono) ON tema9.telefonos TO nonbreUsuario@nombreEquipo;
```

  El usuario indicado puede consultar la columna id e insertar datos en las columna *id* y *telefono* de la tabla *telefonos* de la base de datos *tema9*. Es resto de columnas tomarán su valor por defecto tras la inserción.

- Privilegio de procedimiento almacenado y función: Se aplican sobre los procedimientos almacenados: procedimientos y funciones. Se almacenan en la tabla mysql.procs_priv.

```
GRANT CREATE ROUTINE ON tema9.* TO nonbreUsuario@nombreEquipo;
GRANT EXECUTE ON PROCEDURE tema9.miProcedure TO nonbreUsuario@nombreEquipo;
```

  En la primera consulta el usuario especificado puede crear procedimientos almacenados relacionados a cualquier tabla del *tema9* mientas que en la segunda puede ejecutar el procedimiento *miProcedure* definido en la base de datos tema9.

Esta tabla tenemos un resumen de los distintos niveles de privilegios que se pueden especificar en on donde la palabra *table* es opcional y no influye en el permiso otorgado:

NIVEL	SINTAXIS	AFECTA
Global	*[table]* *.*	A todas los elementos de todas las base de datos.
Base de datos	*[table]* bd.*	A todas los elementos de la base de datos *bd.*
	[table] *	A todas los elementos de la base de datos actual. Si no hay una base de datos seleccionada.se otorgan privilegios de manera global
Tabla	*[table]* bd.miTabla	A la tabla *miTabla* de la base de datos bd.
	[table] miTabla	A la tabla *miTabla* de la base de datos actual.
Procedimiento	*Procedure* bd.miProcedure	Al procedimiento *miProcedure* de la base de datos *bd*.
	Procedure miProcedure	Al procedimiento *miProcedure* de la base de datos *actual*.
Función	*Function* bd.miFunction	A la función *miFunction* de la base de datos *bd*.
	Function miFunction	A la función *miFunction* de la base de datos *bd*.

MariaDB/MySQL dispone, entre otros, de los siguientes privilegios que se pueden ver con el comando

```
SHOW PRIVILEGES
```

PRIVILEGIO	Privilegio que otorga
all [privileges]	Otorga todos los privilegios excepto *grant option*
usage	No otorga ningún privilegio. Sinónimo de *"no privileges"*
alter	Modificar la estructura de una tabla
create	Crear bases de datos y tablas
delete	Borrar filas de una tabla mediante la instrucción *delete*
create user	Administrar cuentas de usuario mediante los comandos: *create user, drop user, rename user y revoke all privileges.*
create view	Crear y modificar vistas
show view	Ver la sentencia de creación de una vista
drop	Uso de la instrucción *drop* para borrar base de datos, tablas y vistas.
index	Permite la creación y eliminación de índices
insert	El uso de la instrucción *insert* para añadir filas a una tabla
select	El uso de la instrucción *select* para obtener datos de una tabla
update	El uso de la instrucción *update* para actualizar una fila de una tabla
file	El uso de *select . . . into outfile* y *load data infile*
reload	El uso de la instrucción *flush*
grant option	Permite a los usuarios otorgar y eliminar sus mismos privilegios a otros usuarios mediante el uso de *grant* y *revoke*
shutdown	Detener el servidor de bases de datos
lock tables	El uso de la instrucción *lock tables* para adquirir un bloqueo sobre una tabla
show tables	Permite mostrar las tablas de una base de datos.
show databases	Permite mostrar las bases de datos del servidor. Sin este permiso la instrucción show databases solo muestra las bases de datos con tablas para las que tengamos privilegios.
alter routine	Modificar o eliminar procedimientos almacenados
create routine	Crear procedimientos almacenados: procedimientos y funciones
event	Crear, borrar y modificar eventos: tareas programadas
execute	Ejecutar procedimientos almacenados
process	Ver los procesos del sistema mediante la instrucción *show processlist*
references	La creación de claves foráneas
trigger	La realización de operaciones con *triggers*
super	Ejecuta sentencias de superusuario.

Esta lista de privilegios se puede clasificar en los niveles:

- Globales: *create user, file, process, reload, show databases, shutdown, super* y *grant option*.
- Base de datos: *create, drop, event, grant option* y *lock tables*.
- Tabla: alter, *create view, create, delete, drop, grant option, index, insert, references, select, show view, trigger, update* y *references*.
- Columna: *insert, references, select* y *update*.
- Procedimiento almacenado: *alter routine, create routine, execute* y *grant option* se aplican a todos los procedimientos almacenados (procedimientos y funciones) mientras que *create routine* se aplica a cada procedimiento almacenado de forma individual.

1.9. Eliminar privilegios: revoke

La instrucción *REVOKE* elimina privilegios que has sido previamente otorgados con el comando *grant*.

Esta instrucción solo borrar permisos no borra usuarios.

Su sintaxis es:

```
REVOKE permiso1 [ (columnas) ] [,permiso2 [ (columnas) ], ... ] ...
ON [tipoObjeto] objetoDeLaBD
FROM cuentaDeUsuario 1  [ , cuentaDeUsuario 2, ...]
```

Cada elemento de la definición es el mismo que para *grant* variando solamente como se especifica el usuario. Pasando del *TO* usado en *Grant* al *FROM* usado en *revoke*.

Los siguientes son ejemplos de su utilización:

```
REVOKE select  ON tema9.* FROM visitante@localhost;
REVOKE all privileges, grant option ON *.* FROM visitante@localhost;
```

1.10. Límites de recursos por cuenta

Entre las tareas que puede realizar el administrador de la base de datos está la de limitar la cantidad de recursos que un usuario puede utilizar. Con ello se intenta evitar que un usuario, o varios, puedan monopolizar los recursos del servidor.

Las limitaciones que se pueden establecer son:

TIPO DE LÍMITE	DESCRIPCIÓN
MAX_QUERIES_PER_HOUR	Límite de consultas que un usuario puede realizar por hora, incluyendo actualizaciones.
MAX_UPDATES_PER_HOUR	Límite de actualizaciones que un usuario puede realizar por hora.
MAX_CONNECTIONS_PER_HOUR	Límite de conexiones que un usuario puede realizar por hora
MAX_USER_CONNECTIONS	Limite de conexiones que un usuario pueda tener de forma simultanea
MAX_STATEMENT_TIME	Tiempo en segundos de duración máxima de ejecución de un procedimiento almacenado. En caso de superar el valor la ejecución del procedimiento es abortado. En *MySQL* el tiempo se mide en milisegundos.

Un valor cero significa especifica un valor ilimitado.

Estos límites corresponden con la clausula *resource_option* que se ha visto en las instrucciones *create user, alter user* y *grant*.

Un ejemplo de utilización es el siguiente:

```
GRANT USAGE ON *.* TO  visitante@localhost
WITH
        MAX_QUERIES_PER_HOUR  0
        MAX_UPDATES_PER_HOUR  40;
```

1.11. Gestión de los usuarios conectados a la base de datos.

En *MariaDB/MySQL* cada conexión contra la base de datos, aun con el mismo usuario, genera un hilo de ejecución que recibe un código de identificación único llamando *id*. Este código se inicializa a 0 cada vez que es servidor se arranca.

Para conocer el código de identificación del hilo asociado al usuario actual se puede usar el comando:

```
SELECT connection_id();
+-----------------+
| connection_id() |
+-----------------+
|              55 |
+-----------------+
```

Si lo que se desea es obtener la lista de todos los hilos, entre ellos los de los usuarios conectados la base de datos, se puede usar el comando *show processlist*. Si se tiene el privilegio *process* muestra todos los hilos de ejecución, en caso contrario muestra solo los hilos propios del usuario que ejecuta el comando.

```
SHOW processlist;
+----+------+-----------------+-------+---------+------+-------+------------------+----------+
| Id | User | Host            | db    | Command | Time | State | Info             | Progress |
+----+------+-----------------+-------+---------+------+-------+------------------+----------+
|  1 | root | localhost:50976 | tema9 | Sleep   |   17 | INIT  | NULL             | 0,000    |
| 55 | root | localhost:51304 | NULL  | Query   |    0 | NULL  | show processlist | 0,000    |
| 57 | pepe | localhost:51310 | NULL  | Sleep   |  290 |       | NULL             | 0,000    |
+----+------+-----------------+-------+---------+------+-------+------------------+----------+
```

Donde cada columna significa:

- ID: identificador de conexión.
- USER: usuario que lanza el hilo. Se pueden encontrar las siguientes posibilidades:
  - nombre de usuario: usuario que ha lanzado el hilo.
  - *system user*: hilo interno del servidor.
  - *event_scheduler*: hilo lanzando por el programador de eventos.
- HOST: equipo y puerto que utiliza un usuario para conectarse. Si el usuario es *system user* esta columna aparece vacía.
- DB: La base de datos seleccionada o *null* si no se ha seleccionado ninguna.
- COMMAND[2]: El tipo de comando que está ejecutando el hilo.
- TIME: Tiempo que lleva el hilo en el estado actual.
- STATE[3]: Acción, evento o estado que indica que hace el hilo. Para *show processlist* es nulo.
- INFO: La instrucción que se está ejecutando o *null* si no se está ejecutando nada.
- PROGRESS[4]: el progreso total del el hilo de 0 a 100%.

Esta misma información se puede consultar con el comando:

[2] https://mariadb.com/kb/en/library/thread-command-values/
https://dev.mysql.com/doc/refman/5.7/en/thread-commands.html

[3] https://mariadb.com/kb/en/library/thread-states/
https://dev.mysql.com/doc/refman/5.7/en/general-thread-states.html

[4] https://mariadb.com/kb/en/progress-reporting/

```
SELECT * FROM information_schema.PROCESSLIST
```

A la que se le añaden las columnas de información:

- TIME_MS: Tiempo que lleva el hilo en el estado actual pero en milisegundos.
- STAGE: La etapa en la que se encuentra actualmente el proceso
- MAX_STAGE: Número máximo de etapas.
- MEMORY_USED: Memoria en *bytes* utilizada por el hilo.
- EXAMINED_ROWS: Filas examinadas por el hilo. Solo toma valor con *UPDATE, DELETE* y declaraciones similares. Para *SELECT* y otras declaraciones, su valor es cero.
- QUERY_ID: identificador de consulta.
- INFO_BINARY: Información en formato binario.

Los hilos de ejecución se pueden abortar con el comando *Kill*. Si el hilo corresponde con el de un usuario su conexión se cierra.

```
KILL NumeroDeConexion;   -- valor de la columna id
```

Si se desea obtener información sobre el número y tipo de conexiones que se han realizado contra el servidor *MariaDB/MySQL* se puede usar el comando *show status* filtrando el resultado con *like*:

```
SHOW STATUS LIKE '%connect%';
+-----------------------------------------------+-------+
| Variable_name                                 | Value |
+-----------------------------------------------+-------+
| Aborted_connects                              | 7     |
| Connection_errors_accept                      | 0     |
| Connection_errors_internal                    | 0     |
| Connection_errors_max_connections             | 0     |
| Connection_errors_peer_address                | 0     |
| Connection_errors_select                      | 0     |
| Connection_errors_tcpwrap                     | 0     |
| Connections                                   | 36    |
| Max_used_connections                          | 5     |
| Slave_connections                             | 0     |
| Slaves_connected                              | 0     |
| Ssl_client_connects                           | 0     |
| Ssl_connect_renegotiates                      | 0     |
| Ssl_finished_connects                         | 0     |
| Threads_connected                             | 4     |
+-----------------------------------------------+-------+
```

2. Vistas

Una vista es una tabla virtual que accede y muestra datos pertenecientes una o más tablas o vistas (incluso puede no acceder a ninguna) y cuyo contenido está definido por una consulta. Al igual que una tabla, está formada por filas y columnas. Un vista se puede ver como un "alias" a una consulta.

Una vista no contiene datos sino que los obtiene, cada vez que se ejecuta, de las tablas a las que hace referencia.

Las vistas se utilizan para:

- Ocultar la estructura de la base de datos: ya sea mostrando una consulta sobre una o varias tablas como una única tabla como permitiendo la modificación de la estructura de la base de datos de forma transparente a los usuarios.
- Restringir el acceso a ciertas columna o filas de una tabla debido a que se evita que tengan acceso a la tabla o tablas originales. Por ejemplo no mostrando todas las columnas o a un usuario en particular mostrando solo datos del departamento al que tiene acceso.
- Facilitar la ejecución de consultas complejas que se ejecutan de manera frecuente.

Hay que considerar las siguientes reglas para la creación de vistas:

- Una vista pertenece a una base de datos creándose por defecto en la base de datos actual. Si al nombre de la vista se le antepone el nombre de la base de datos se crea en la base de datos especificada
- Una vista puede hacer referencia a una o más tablas o a otras vistas
- Dentro de la misma base de datos el nombre de una vista ha de ser único. No puede coincidir con una tabla o una vista existentes.
- Una vista no puede tener nombres de columnas repetidos.
- Si se elimina una tabla o vista a la que la vista hace referencia la vista producirá un error.

La creación de vistas tiene las siguientes restricciones:

- La sentencia SELECT no puede tener una subconsulta en la clausula FROM.
- La sentencia SELECT no puede referirse a variables del sistema o del usuario.
- Todas las tablas o vistas a las que se hacen referencia deben de existir.
- La consulta asociada con la vista no puede contener errores.
- No se puede asociar un trigger con una vista.
- Aunque en un vista se puede utilizar ORDER BY este no se tiene en cuenta si la consulta asociada con la vista tiene su propio ORDER BY.
- No se puede determinar el resultado final si la vista si la invocamos con una LIMIT y la consulta asociada tienen su propio LIMIT.
- La vista no puede referenciar a una tabla temporal ni se puede crear una vista temporal.

2.1. Crear vistas

Para crear una vista, o reemplazarla, se dispone del comando *CREATE VIEW.* Se necesita tener el privilegio *CREATE VIEW*. Una vista siempre pertenece a una base de datos.

Su sintaxis es:

```
CREATE
        [OR REPLACE]
        [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
        [SQL SECURITY { DEFINER | INVOKER }]
    VIEW [IF NOT EXISTS] [bd.]nombreDeLaVista[(listaColumnas)]
    AS  consultaSelect;
```

Donde:

- *OR REPLACE*: si la vista ya existe esta es reemplazada. En este caso es el mismo comando que ALTER VIEW. Si aparece se necesita el privilegio DROP *VIEW* sobre la vista.
- *DEFINER*: permite especificar el usuario que ejecuta la vista. Si no se dispone del privilegio *SUPER* solo se puede especificar el propio usuario o role. Si no se especifica un valor se usará el valor definido en *SQL SECURITY*.
- *SQL SECURITY*: determina con que cuenta de usuario (y por lo tanto con que privilegios) se ejecuta la vista. *Definer*, que es el valor por defecto, representa al usuario que ha creado la vista mientras que *invoker* representa al usuario que ejecuta la vista.
- *consultaSelect*: es la consulta que forma la vista y que obtiene datos de tablas y otras vistas. Para que la vista funcione la consulta no puede dar error en el momento de la ejecución (por ejemplo que se modifique ya estructura de una tabla sobre las que se define la vista).
- *IF NOT EXISTS*: solo crea la vista si no existe. Si ya existe produce un *warning* en vez de un error.
- *bd*: base de datos donde se va a crear al vista y a la cual va a pertenecer. Si no aparece se crea en la base de datos actual.
- *nombreDeLaVista*: nombre que se le asigna a la vista y que se utilizará para referenciarla. Una vista no puede tener el mismo nombre que una tabla u otra vista en la misma base de datos.
- *listaColumnas*: se utilizan para asignar un nombre a las columnas devueltas por la vista. Se tienen que especificar tantas columnas como columnas devuelva la consulta asociada a la vista. Si no se definen el nombre de las columnas será el devuelto por la consulta asociada a la vista. Al igual que en una tabla en una vista no pueden tener nombres repetidos.
- *consultaSelect*: consulta *select* válida que ejecuta la vista y de la cual obtiene los datos. La vista se crea con las columnas devueltas por la consultas *select* asociada en el momentos de la definición. Si a posteriori, tras modificar las tablas o vistas que forman el *select*, la consulta select devolviese más filas estás no se verían reflejadas en la vista.

Vamos a ver un ejemplo de creación de una vista:

```
CREATE VIEW emple_depart (Nombre, Oficio, Sueldo, Departamento, Ciudad) AS
        SELECT apellido, oficio, salario, nombre, loc FROM empleados NATURAL JOIN depart;
```

Una vez creada la vista se pueden ejecutar consultas sobre ella como si fuese una tabla.

```
select * from emple_depart where sueldo >3000;
+--------+------------+--------+--------------+-----------+
| Nombre | Oficio     | Sueldo | Departamento | Ciudad    |
+--------+------------+--------+--------------+-----------+
| Cerezo | Director   |  13000 | Contabilidad | Sevilla   |
| Rey    | Presidente |   4100 | Contabilidad | Sevilla   |
| Negro  | Director   |   3005 | Ventas       | Barcelona |
+--------+------------+--------+--------------+-----------+
```

2.2. Modificar vistas

La instrucción *ALTER VIEW* permite modificar una vista que ya existe y su efecto es el mismo que CREATE OR REPLACE.

Su formato es:

```
ALTER
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    [SQL SECURITY { DEFINER | INVOKER }]
VIEW  [bd.]nombreDeLaVista[(listaColumnas)]
AS  consultaSelect;
```

Donde el significado de cada elemento es el mismo que en el caso de la creación de una vista.

Un ejemplo de utilización es:

```
ALTER VIEW emple_depart (Nombre, Oficio, Sueldo, Ciudad) AS
        SELECT apellido, oficio, salario, loc FROM empleados NATURAL JOIN depart;
```

2.3. Renombrar vistas

Se puede usar la instrucción *RENAME TABLE*, que ya hemos visto en temas pasados. para cambiar el nombre de una vista. Se debe tener en cuenta que los permisos no se migran.

En temas pasados hemos usado este comando para mover tablas entre bases de datos pero no permite mover vistas.

Un ejemplo de su utilización es:

```
RENAME TABLE emple_depart TO emple_depart2;
```

2.4. Borrar vistas

Para borrar una, o varias, vistas se puede utilizar la instrucción *DROP VIEW*.

Su formato es:

```
DROP VIEW [ IF EXISTS ] nombreDeLaVista1 [ , nombreDeLaVista2 ] ...
```

Un ejemplo de su utilización es:

```
DROP VIEW emple_depart2;
```

2.5. Consultar vistas

Las vistas creadas se almacenan en la tabla *views* de la base de datos *information_schema*. La columna *table_schema* representa la base de datos en la que se ha creado la vista.

Para obtener la sentencia de creación de una vista se dispone de la instrucción:

```
SHOW CREATE VIEW nombreDeLaVista;
```

Si queremos analizar la estructura de la vista podemos usar:

```
DESCRIBE  nombreDeLaVista;
```

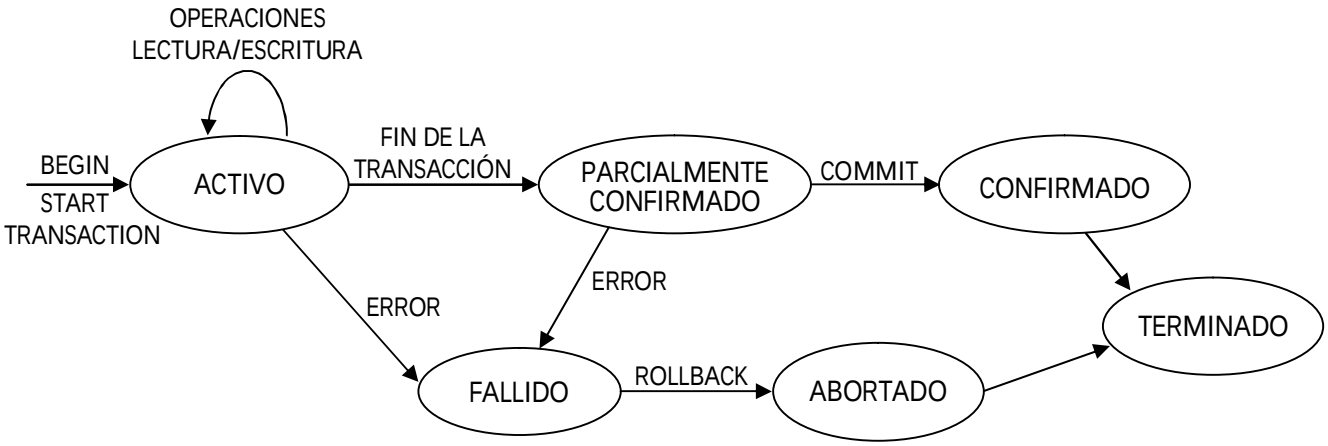
3. Transacciones

Una transacción es una unidad de trabajo formada por instrucciones, de lectura y/o escritura, que se ejecutan en la base de datos de forma atómica, es decir, o se ejecutan todas o no se ejecuta ninguna. Permiten garantizar la consistencia e integridad de una base de datos.

En *MariaDB/MySQL* el soporte para las transacciones es proporcionado por el motor de almacenamiento *InnoDB*. Su uso provoca una mayor carga en el sistema al tener al tener que cumplir una serie de propiedades conocidas como propiedades *ACID*.

- Atomicidad (*Atomicity*): garantiza la ejecución de una transacción no se queda a medias, o se ejecutan todas sus instrucciones o no se ejecuta ninguna
- Consistencia (*Consistency*): la base de datos, tras la ejecución de una transacción, debe continuar en un estado consistente.
- Aislamiento (*Isolation*): la ejecución de una transacción debe ser independiente a las ejecuciones de las demás transacciones concurrentes y el resultado producido debe ser equivalente a que las transacciones de ejecuten de forma lineal.
- Persistencia (*Durability*): cuando una transacción ha sido completada de forma satisfactoria los cambios que ha podido producir se han de salvar de forma permanente incluso en caso de fallo del sistema.

El siguiente diagrama representa la secuencia de ejecución de una transacción:



Donde cada estado significa:

- Activo: estado en el que se encuentra una transacción al comenzar su ejecución. En este estado se realizan todas las operaciones de lectura/escritura que contiene la transacción.
- Parcialmente confirmado: es el estado al que se pasa cuando la última instrucción de la transacción se ha ejecutado. En él, mediante técnicas de control de concurrencia, se llevan a cabo una serie de comprobaciones.
- Fallido: se llega a este estado si se produce un error durante la ejecución de las instrucciones de la transacción o no se superan las comprobaciones que se realizan en el punto anterior.
- Abortado: se pasa a este estado tras deshacer los cambios producidos por una transacción fallida y restaurar la base de datos al estado anterior al comienzo de la transacción.
- Confirmado: en este estado la transacción ha finalizado de forma correcta y los datos se han salvador de forma permanente.
- Terminado: Es el estado final de una transacción tras la cual desaparece del sistema.

Según el esquema anterior el funcionamiento de las transacciones es el siguiente:

- La transacción se comienza con la instrucción *START TRANSACTION* (también se puede usar *BEGIN*).
- Realizar operaciones de lectura/escritura sobre la base de datos.
- Finalizar la transacción. Pueden ocurrir dos cosas:
  - Las operaciones terminan de forma correcta: mediante la instrucción *COMMIT* se hacen permanentes los cambios producidos.
  - Se produce un error durante la ejecución de las operaciones: se puede usar el comando *ROLLBACK* para deshacer los cambios producidos por la transacción.

Los cambios producidos por una transacción pueden ser vistos por:

- La propia transacción.
- Los demás transacciones, si tienen permisos, cuando la transacción original ha hecho *COMMIT*.

Por defecto una instrucción ejecutada fuera de una transacción confirma de forma automática los cambios que produce. Para consultar su estado y modificar su comportamiento se pueden utilizar los comandos:

- Consultar su estado: `show variables like 'autocommit';`
- Desactivar el *autocommit*: `set autocommit=0;`
- Activar el *autocommit*: `set autocommit=1;`

Si queremos conocer si estamos dentro de una transacción se puede consultar la variable del sistema *in_transaction* que devuelve 1 si estamos en una transacción y 0 si no lo estamos.

Veremos el siguiente ejemplo para ilustrar el funcionamiento de una transacción:

- Creamos una tabla con un solo campo y motor de almacenamiento *InnoDB*. Insertaremos en la tabla 2 valores de prueba.

```
CREATE TABLE prueba (id INT) ENGINE=InnoDB;
INSERT INTO prueba (id) VALUES (1),(2);
```

- Abrimos dos conexiones y comenzamos una transacción en una de ellas. En la otra visualizamos el contenido de la tabla prueba. Luego realizamos el *COMMIT* para confirmar los cambios y volvemos a visualizar la tabla en ambas conexiones.

	Conexión 1	Conexión 2

PASO 1	`START TRANSACTION;`
	`INSERT INTO prueba VALUES(3);`
	`SELECT * FROM prueba;`
	`+------+`
	`| id |`
	`+------+`
	`| 1 |`
	`| 2 |`
	`| 3 |`
	`+------+`
		`SELECT * FROM prueba;`
		`+------+`
		`| id |`
		`+------+`
		`| 1 |`
		`| 2 |`
		`+------+`

PASO 2	`select @@in_transaction;`
	`+------+`
	`| @@in_transaction |`
	`+------+`
	`| 1 |`
	`+------+`
		`select @@in_transaction;`
		`+------+`
		`| @@in_transaction |`
		`+------+`
		`| 0 |`
		`+------+`

PASO 3	`COMMIT;`
	`SELECT * FROM prueba;`
	`+------+`
	`| id |`
	`+------+`
	`|  1 |`
	`|  2 |`
	`|  3 |`
	`+------+`
		`SELECT * FROM prueba;`
		`+------+`
		`| id |`
		`+------+`
		`|  1 |`
		`|  2 |`
		`|  3 |`
		`+------+`

PASO 4	`select @@in_transaction;`
	`+------+`
	`| @@in_transaction |`
	`+------+`
	`|  0 |`
	`+------+`
		`select @@in_transaction;`
		`+------+`
		`| @@in_transaction |`
		`+------+`
		`|  0 |`
		`+------+`

- Volvemos a realizar las mismas operaciones pero esta vez en vez de realizar un *COMMIT* realizamos un *ROLLBACK* para deshacer los cambios producidos.

	Conexión 1	Conexión 2

PASO 1	`START TRANSACTION;`
	`INSERT INTO prueba VALUES(4);`
	`SELECT * FROM prueba;`
	`+------+`
	`| id |`
	`+------+`
	`|  1 |`
	`|  2 |`
	`|  3 |`
	`|  4 |`
	`+------+`
		`SELECT * FROM prueba;`
		`+------+`
		`| id |`
		`+------+`
		`|  1 |`
		`|  2 |`
		`|  3 |`
		`+------+`

PASO 2	`select @@in_transaction;`
	`+------+`
	`| @@in_transaction |`
	`+------+`
	`|  1 |`
	`+------+`
		`select @@in_transaction;`
		`+------+`
		`| @@in_transaction |`
		`+------+`
		`|  0 |`
		`+------+`

PASO 3	`ROLLBACK;`
	`SELECT * FROM prueba;`
	`+------+`
	`| id |`
	`+------+`
	`|  1 |`
	`|  2 |`
	`|  3 |`
	`+------+`
		`SELECT * FROM prueba;`
		`+------+`
		`| id |`
		`+------+`
		`|  1 |`
		`|  2 |`
		`|  3 |`
		`+------+`

PASO 4	`select @@in_transaction;`
	`+------+`
	`| @@in_transaction |`
	`+------+`
	`|  0 |`
	`+------+`
		`select @@in_transaction;`
		`+------+`
		`| @@in_transaction |`
		`+------+`
		`|  0 |`
		`+------+`

Además de *COMMIT* y de *ROLLBACK* existen una serie de comandos, normalmente que modifican la estructura de la base de datos, que finalizan la transacción en curso puesto que realizan un *commit* de forma implícita antes de su ejecución, por lo que, tras su ejecución, no se pueden deshacer los cambios realizados por la transacción.

ALTER TABLE	TRUNCATE TABLE	DROP INDEX
CREATE TABLE	CREATE DATABASE	START TRANSACTION / BEGIN
RENAME TABLE	DROP DATABASE	LOCK/UNLOCK TABLES
DROP TABLE	CREATE INDEX	SET AUTOCOMMIT=1

Cuando se utiliza *ROLLBACK* se deshacen todos los cambios hasta el inicio de la transacción y finalizándose esta.

Existe la posibilidad de crear puntos intermedios de restauración que permiten deshacer los cambios producidos hasta un punto de restauración en concreto.

Para ello se disponen de las instrucciones:

- Crear punto de restauración: SAVEPOINT nombrePuntoRecuperación;
- Deshacer a un punto de restauración: ROLLBACK TO SAVEPOINT nombrePuntoRecuperación;

Dentro de una transacción no puede existir más de un punto de restauración con el mismo nombre. Si se crea un nuevo punto con el nombre de uno ya creado el punto más antiguo se elimina.

Cuando se deshace una transacción hasta un punto de restauración todos los puntos creados después se eliminan.

Si no existe ningún punto de restauración con el nombre proporcionado se produce el error siguiente:

```
ERROR 1305 (42000): SAVEPOINT nombrePuntoRecuperación does not exist
```

Commit y *Rollback* eliminan todos los puntos de restauración y terminan la transacción.

4. Bloqueos

Un bloqueo es una variable que se asocia a un elemento de la base de datos y determina su estado ante operaciones de lectura y/o escritura. Se utiliza como técnica de control de concurrencia y se suele usar en motores de almacenamiento que no soportan transacciones, como puede ser *MyISAM*.

Se pueden obtener dos tipos de bloqueos:

- Bloqueos de lectura o compartidos: solo permiten realizar lecturas sobre la tabla, quedando bloqueadas, y a la espera, las operaciones de escritura.
- Bloqueos de escritura o exclusivos: permiten realizar operaciones de lectura/escritura al poseedor del bloqueo mientras que las demás conexiones que quieran leer y/o escribir tienen que esperar a que el poseedor del bloqueo lo libere.

Un problema que puede surgir del uso de bloqueos es el interbloqueo o *deadlock*. Se produce cuando dos o más conexiones compiten por conseguir los mismos recursos esperando que los demás liberen recursos que necesita. El *SGBD* dispone de técnicas para detectar y prevenir estos interbloqueos.

MariaDB/MySQL dispones de los siguientes comandos para bloquear/desbloquear tablas:

- Bloquear tablas:

  ```
  LOCK TABLE nombreTabla1 {READ|WRITE} [ , nombreTabla2 {READ|WRITE},... ]
  ```

  Donde *Read* o *Write* indica el tipo de bloqueo que se quiere obtener: lectura y compartido o escritura y exclusivo.

- Liberar tablas:

  ```
  UNLOCK TABLES;
  ```

  Se liberan todos los bloqueos que se hubiesen obtenido, no se puede liberar el bloqueo de una única tabla.

Lock/unlock tables no se pueden usar dentro de una transacción ya que provoca un *commit* implícito.

Un ejemplo de su utilización es:

	Conexión 1	Conexión 2

PASO 1	`LOCK TABLE prueba WRITE;`
	`TRUNCATE TABLE prueba;`
	`INSERT INTO prueba VALUES(7);`
	`SELECT * FROM prueba;`
	`+------+`
	`| id |`
	`+------+`
	`|  7 |`
	`+------+`
		`SELECT * FROM prueba;`
		-- Se queda esperando a que se libere el bloqueo de escritura.

PASO 2	-- Se liberan los bloqueos
	`UNLOCK TABLES;`
		-- Se muestra el resultado
		`+------+`
		`| id |`
		`+------+`
		`|  7 |`
		`+------+`

PASO 3	`LOCK TABLE prueba READ;`
	`SELECT * FROM prueba;`
	`+------+`
	`| id |`
	`+------+`
	`|  7 |`
	`+------+`
		`SELECT * FROM prueba;`
		`+------+`
		`| id |`
		`+------+`
		`|  1 |`
		`|  2 |`
		`|  3 |`
		`+------+`

PASO 4	`INSERT INTO prueba VALUES(8);` -- se produce un error ya que solo se tiene el bloqueo de lectura y no de escritura.	`INSERT INTO prueba VALUES(8);` -- Se queda esperando a que se libere el bloqueo de lectura.
PASO 5	-- Se liberan los bloqueos `UNLOCK TABLES;`	-- Se produce la inserción.