TEMA 10: CREACIÓN Y MANIPULACIÓN DE OTROS OBJETOS DE LA BASE DE DATOS

1. Usuarios en MySQL.

Las bases de datos normalmente son empleadas por varios usuarios y en entornos multiusuario es necesario realizar un control de acceso para garantizar que solo puedan acceder a la base de datos los usuarios autorizados. Además, es preciso garantizar que cada usuario que accede a la base de datos solo pueda realizar las operaciones que le correspondan de acuerdo con su perfil.

En MySQL existe la instrucción CREATE USER, que permite crear usuarios para la base de datos. La sintaxis de esta instrucción es la siguiente:

```
CREATE USER [IF NOT EXISTS]
Nombre_usuario1 [IDENTIFIED BY 'contraseña1'],
[Nombre_usuario2 [IDENTIFIED BY 'contraseña2']]…
[DEFAULT ROLE NombreRol1, NombreRol2, …]
[WITH opción_with]
[PASSWORD EXPIRE {DEFAULT | NEVER | INTERVAL n DAY}];
```

Se debe especificar después de la palabra USER el nombre del usuario que se desea crear, el cual puede constar solo de un nombre de usuario o puede tener la forma nombre_usuario@nombre_host, donde nombre_host es el nombre de la máquina desde la que se efectúa la conexión. Puede tratarse de una dirección IP (que se debe indicar entre comillas) o del servidor local, en cuyo caso hay que escribir *localhost*.

Se puede crear un usuario sin contraseña, lo cual no es aconsejable, o, por el contrario, se le puede asignar una contraseña especificando, después de IDENTIFIED BY, la contraseña entre comillas simples.

En la misma instrucción se pueden crear varios usuarios separando la información de un usuario de la de otro usuario por el símbolo coma.

Se puede indicar opcionalmente el rol o los roles que se desea asignar al usuario escribiendo después de DEFAULT ROLE el nombre o nombres de dichos roles.

Es posible especificar cuándo expirará la contraseña del/de los usuario/s creados/s mediante la opción PASSWORD EXPIRE. Se pueden indicar 3 valores:

- *Default* indica que expira según está especificado en la variable del sistema *default_password_lifetime* (número de días).
- *Never* indica que no expira nunca.
- *Interval n day* indica que expira dentro de n días.

Por ejemplo, mediante la siguiente instrucción creamos dos usuarios: uno llamado *jose*, con contraseña *1234*, que se va a poder conectar desde cualquier ordenador, y otro llamado *ana*, con contraseña *5678*, que solo se va a poder conectar desde el ordenador local.

```
CREATE USER jose IDENTIFIED BY '1234',
ana@localhost IDENTIFIED BY '5678';
```

Cuando se crean usuarios, MySQL añade dichos usuarios a la tabla *User* de la base de datos *MySQL*. Esta tabla presenta el siguiente formato:

```
mysql> USE mysql
Database changed
mysql> DESC user;
+-------------------------+---------------------------------+------+-----+-----------------------+-------+
| Field                   | Type                            | Null | Key | Default               | Extra |
+-------------------------+---------------------------------+------+-----+-----------------------+-------+
| Host                    | char(255)                       | NO   | PRI |                       |       |
| User                    | char(32)                        | NO   | PRI |                       |       |
| Select_priv             | enum('N','Y')                   | NO   |     | N                     |       |
| Insert_priv             | enum('N','Y')                   | NO   |     | N                     |       |
| Update_priv             | enum('N','Y')                   | NO   |     | N                     |       |
| Delete_priv             | enum('N','Y')                   | NO   |     | N                     |       |
| Create_priv             | enum('N','Y')                   | NO   |     | N                     |       |
| Drop_priv               | enum('N','Y')                   | NO   |     | N                     |       |
| Reload_priv             | enum('N','Y')                   | NO   |     | N                     |       |
| Shutdown_priv           | enum('N','Y')                   | NO   |     | N                     |       |
| Process_priv            | enum('N','Y')                   | NO   |     | N                     |       |
| File_priv               | enum('N','Y')                   | NO   |     | N                     |       |
| Grant_priv              | enum('N','Y')                   | NO   |     | N                     |       |
| References_priv         | enum('N','Y')                   | NO   |     | N                     |       |
| Index_priv              | enum('N','Y')                   | NO   |     | N                     |       |
| Alter_priv              | enum('N','Y')                   | NO   |     | N                     |       |
| Show_db_priv            | enum('N','Y')                   | NO   |     | N                     |       |
| Super_priv              | enum('N','Y')                   | NO   |     | N                     |       |
| Create_tmp_table_priv   | enum('N','Y')                   | NO   |     | N                     |       |
| Lock_tables_priv        | enum('N','Y')                   | NO   |     | N                     |       |
| Execute_priv            | enum('N','Y')                   | NO   |     | N                     |       |
| Repl_slave_priv         | enum('N','Y')                   | NO   |     | N                     |       |
| Repl_client_priv        | enum('N','Y')                   | NO   |     | N                     |       |
| Create_view_priv        | enum('N','Y')                   | NO   |     | N                     |       |
| Show_view_priv          | enum('N','Y')                   | NO   |     | N                     |       |
| Create_routine_priv     | enum('N','Y')                   | NO   |     | N                     |       |
| Alter_routine_priv      | enum('N','Y')                   | NO   |     | N                     |       |
| Create_user_priv        | enum('N','Y')                   | NO   |     | N                     |       |
| Event_priv              | enum('N','Y')                   | NO   |     | N                     |       |
| Trigger_priv            | enum('N','Y')                   | NO   |     | N                     |       |
| Create_tablespace_priv  | enum('N','Y')                   | NO   |     | N                     |       |
| ssl_type                | enum('','ANY','X509','SPECIFIED') | NO   |     |                       |       |
| ssl_cipher              | blob                            | NO   |     | NULL                  |       |
| x509_issuer             | blob                            | NO   |     | NULL                  |       |
| x509_subject            | blob                            | NO   |     | NULL                  |       |
| max_questions           | int unsigned                    | NO   |     | 0                     |       |
| max_updates             | int unsigned                    | NO   |     | 0                     |       |
| max_connections         | int unsigned                    | NO   |     | 0                     |       |
| max_user_connections    | int unsigned                    | NO   |     | 0                     |       |
| plugin                  | char(64)                        | NO   |     | caching_sha2_password |       |
| authentication_string   | text                            | YES  |     | NULL                  |       |
| password_expired        | enum('N','Y')                   | NO   |     | N                     |       |
| password_last_changed   | timestamp                       | YES  |     | NULL                  |       |
| password_lifetime       | smallint(5) unsigned            | YES  |     | NULL                  |       |
| account_locked          | enum('N','Y')                   | NO   |     | N                     |       |
| Create_role_priv        | enum('N','Y')                   | NO   |     | N                     |       |
| Drop_role_priv          | enum('N','Y')                   | NO   |     | N                     |       |
| Password_reuse_history  | smallint(5) unsigned            | YES  |     | NULL                  |       |
| Password_reuse_time     | smallint(5) unsigned            | YES  |     | NULL                  |       |
| Password_require_current | enum('N','Y')                   | YES  |     | NULL                  |       |
| User_attributes         | json                            | YES  |     | NULL                  |       |
+-------------------------+---------------------------------+------+-----+-----------------------+-------+
51 rows in set (0.15 sec)
```

Los campos que más nos interesan ahora de esta tabla son los siguientes:

- *Host*: Contiene el nombre de la máquina desde la que se puede conectar el usuario.
- *User*: Contiene el nombre del usuario que se ha creado.
- *Authentication_string*: Es la contraseña con la que accede el usuario correspondiente.

Veamos el contenido de estos campos para todas las filas de esta tabla:

```
SELECT host, user, authentication_string FROM user;
+-----------+------------------+-------------------------------------------------------------------------+
| host      | user             | authentication_string                                                   |
+-----------+------------------+-------------------------------------------------------------------------+
| %         | jose             | *A4B6157319038724E3560894F7F932C8886EBFCF                               |
| localhost | ana              | *F13ACB16013CCF84877CED35B9DB9105C0C98A43                               |
| localhost | mysql.infoschema | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | mysql.session    | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | mysql.sys        | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | root             | *53890EDEEA2F1FE635CC0EC35A76445350A17909                               |
+-----------+------------------+-------------------------------------------------------------------------+
6 rows in set (0.00 sec)
```

Hay varios usuarios, concretamente *root*, *mysql.infoschema*, *mysql.session* y *mysql.sys*, que son creados por MySQL cuando se realiza la instalación. Podemos observar que están además los usuarios *jose* y *ana* que acabamos de crear. Para el usuario *jose* el campo host toma el valor %, lo que quiere decir que este usuario se puede conectar desde cualquier ordenador; sin embargo, en el caso de *ana*, en el atributo *host* pone *localhost*, lo que quiere decir que solo se podrá conectar desde el ordenador local. Ambos tienen asignada una contraseña, la cual está encriptada.

Todos los campos cuyo nombre finaliza en *priv* hacen referencia a si el usuario en cuestión tiene otorgado o no el privilegio correspondiente a nivel global, según si el valor del atributo en cuestión es 'Y' (sí) o 'N' (no), respectivamente.

La opción *with* dentro de la instrucción CREATE USER puede contener uno o varios de los valores que se especifican a continuación:

- MAX_QUERIES_PER_HOUR número: indica el número de consultas y actualizaciones (inserciones, modificaciones y borrados) que ese usuario puede realizar por hora.
- MAX_UPDATES_PER_HOUR número: indica el número de actualizaciones (inserciones, modificaciones y borrados) que ese usuario puede realizar por hora.
- MAX_CONNECTIONS_PER_HOUR número: indica el número de conexiones o logueos que puede realizar el usuario como máximo en una hora.
- MAX_USER_CONNECTIONS número: indica el número máximo de conexiones simultáneas que el usuario puede establecer con el servidor.

Si en cualquiera de los cuatro valores presentados no se asigna ningún valor, se supone un cero, que quiere decir que no hay límite.

Por ejemplo, podemos crear un tercer usuario llamado *santi* que solo se pueda conectar desde el ordenador local y vamos a limitarle el número de consultas y actualizaciones por hora a 100 y el número de conexiones simultáneas a 10:

```
CREATE USER santi@localhost IDENTIFIED BY '4321'
WITH MAX_QUERIES_PER_HOUR 100
MAX_USER_CONNECTIONS 10;
```

La información sobre las restricciones impuestas al usuario se almacena también en la tabla *User*, concretamente en los atributos *max_questions*, *max_updates*, *max_connections* y *max_user_connections*. Así, si consultamos para los diferentes usuarios del servidor su nombre de usuario y estos atributos, vemos el siguiente contenido:

```
SELECT host, user, max_questions, max_updates, max_connections, max_user_connections
FROM user;
+-----------+------------------+---------------+-------------+-----------------+----------------------+
| host      | user             | max_questions | max_updates | max_connections | max_user_connections |
+-----------+------------------+---------------+-------------+-----------------+----------------------+
| %         | jose             |             0 |           0 |               0 |                    0 |
| localhost | ana              |             0 |           0 |               0 |                    0 |
| localhost | mysql.infoschema |             0 |           0 |               0 |                    0 |
| localhost | mysql.session    |             0 |           0 |               0 |                    0 |
| localhost | mysql.sys        |             0 |           0 |               0 |                    0 |
| localhost | root             |             0 |           0 |               0 |                    0 |
| localhost | santi            |           100 |           0 |               0 |                   10 |
+-----------+------------------+---------------+-------------+-----------------+----------------------+
7 rows in set (0.00 sec)
```

Se puede cambiar la contraseña de un usuario, las restricciones que tiene impuestas y la opción de expiración de la contraseña mediante el comando ALTER USER con la siguiente sintaxis:

```
ALTER USER [IF EXISTS] Nombre_usuario
[IDENTIFIED BY 'contraseña_nueva']
[WITH opción_with]
[PASSWORD EXPIRE {DEFAULT | NEVER | INTERVAL n DAY}];
```

Así, con la siguiente orden cambiamos la contraseña del usuario *jose* a '4321':

```
ALTER USER jose IDENTIFIED BY '4321';
```

Mediante la siguiente orden cambiamos el número máximo de conexiones simultáneas del usuario santi@localhost a 15:

```
ALTER USER santi@localhost
WITH MAX_USER_CONNECTIONS 15;
```

Se puede cambiar el rol o los roles asignados al usuario mediante la siguiente sintaxis, por medio de la cual se puede dejar al usuario sin ningún rol asignado (opción NONE) o se le pueden asignar todos los roles (opción ALL):

```
ALTER USER [IF EXISTS] Nombre_usuario
DEFAULT ROLE {NONE | ALL | NombreRol1, NombreRol2, …};
```

También se puede cambiar el nombre de un usuario mediante la instrucción RENAME USER, cuya sintaxis es la siguiente:

```
RENAME USER nombre_antiguo1 TO nombre_nuevo1,
[ ,nombre_antiguo2 TO nombre_nuevo2]…
```

Solo se debe indicar el nombre antiguo del usuario y el nuevo nombre y en la misma orden se puede cambiar el nombre a varios usuarios. Por ejemplo, para cambiar el nombre del usuario *ana@localhost* por *anita@localhost* pondremos:

```
RENAME USER ana@localhost TO anita@localhost;
```

Para eliminar un usuario se usa la orden DROP USER, cuya sintaxis es la siguiente:

```
DROP USER [IF EXISTS] Nombre_usuario1 [, Nombre_usuario2]…;
```

Como se puede observar, solo se debe indicar el nombre del usuario que se desea borrar y en la misma instrucción DROP USER se pueden eliminar varios usuarios. Por ejemplo, con la siguiente orden se borrará al usuario *anita*, que se conecta desde el ordenador local.

```
DROP USER anita@localhost;
```

2. Permisos en MySQL.

Los permisos en MySQL se pueden referir a varios niveles:

- Nivel global: los permisos globales se aplican a todas las bases de datos de un servidor dado.
- Nivel de base de datos: los permisos a nivel de base de datos se aplican a todos los objetos de una base de datos dada.
- Nivel de tabla: los permisos a nivel de tabla se aplican a todos los atributos de la tabla.
- Nivel de columna: los permisos a nivel de columna se aplican a columnas específicas en una determinada tabla.
- Nivel de rutina: los permisos a nivel de rutina se aplican a rutinas específicas (procedimientos o funciones).

De hecho, en la base de datos *MySQL*, además de la tabla *User*, que contiene información sobre los usuarios y los permisos a nivel global que tienen otorgados, se encuentran las siguientes tablas:

- *Db*: contiene privilegios a nivel de base de datos.
- *Tables_priv*: contiene privilegios a nivel de tabla.
- *Columns_priv*: contiene privilegios a nivel de atributo.
- *Procs_priv*: contiene privilegios para procedimientos y funciones.

Los permisos que se pueden especificar en MySQL se muestran en la tabla de la página siguiente, donde se indica por cada privilegio su nombre, su significado y el nivel al que se puede otorgar.

Permiso	Significado	Nivel/es
ALL [PRIVILEGES]	Todos los permisos excepto GRANT OPTION	
ALTER	Permite modificar el diseño de una tabla con ALTER TABLE	Global, base de datos, tabla
ALTER ROUTINE	Permite modificar o borrar rutinas almacenadas (procedimientos y funciones).	Global, base de datos, rutina
CREATE	Permite crear bases de datos y tablas	Global, base de datos, tabla
CREATE ROLE	Permite crear roles	Global
CREATE ROUTINE	Permite crear rutinas almacenadas	Global, base de datos
CREATE TABLESPACE	Permite crear *tablespaces*, alterarlos y borrarlos.	Global
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLE	Global, base de datos
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER y REVOKE ALL PRIVILEGES	Global
CREATE VIEW	Permite crear y alterar vistas	Global, base de datos, tabla
DELETE	Permite el uso de DELETE	Global, base de datos, tabla
DROP	Permite eliminar bases de datos, tablas y vistas	Global, base de datos, tabla
DROP ROLE	Permite eliminar roles	Global
EVENT	Permite crear, modificar, eliminar y visualizar eventos.	Global, base de datos
EXECUTE	Permite ejecutar rutinas almacenadas	Global, base de datos, rutina
FILE	Permite al servidor leer y escribir ficheros	Global
GRANT OPTION	Permite conceder o retirar a otros usuarios los privilegios poseídos.	Global, base de datos, tabla, rutina
INDEX	Permite crear y eliminar índices	Global, base de datos, tabla
INSERT	Permite el uso de INSERT	Global, base de datos, tabla, columna
LOCK TABLES	Permite el uso de LOCK TABLES (bloquear tablas) en tablas para las que tenga el permiso SELECT	Global, base de datos
PROCESS	Permite el uso de SHOW PROCESSLIST	Global
REFERENCES	Permite crear claves ajenas	Global, base de datos, tabla, columna
RELOAD	Permite el uso de FLUSH (limpiar la memoria caché)	Global
SELECT	Permite el uso de SELECT	Global, base de datos, tabla, columna
SHOW DATABASES	Permite el uso de SHOW DATABASES, que muestra todas las bases de datos	Global
SHOW VIEW	Permite el uso de SHOW CREATE VIEW	Global, base de datos, tabla
SHUTDOWN	Permite el uso de mysqladmin shutdown (para parar el servidor)	Global
TRIGGER	Permite operaciones sobre disparadores (triggers)	Global, base de datos, tabla
UPDATE	Permite el uso de UPDATE	Global, base de datos, tabla, columna
USAGE	Es un sinónimo de ningún privilegio	

Tabla 1: Permisos en MySQL.

El comando para asignar privilegios en MySQL es GRANT, cuya sintaxis es la siguiente:

```
GRANT privilegio1 [(lista_atributos1)] [, privilegio2 [(lista_atributos2)]]…
ON {nombre_tabla | * | *.* | nombre_BD.* | nombre_BD.nombre_tabla |
    nombre_BD.nombre_rutina}
TO {nombre_usuario1 | nombre_rol1} [,{nombre_usuario2 | nombre_rol2}…]
[WITH GRANT OPTION];
```

Los privilegios que se desean otorgar se deben indicar después de la palabra GRANT y pueden ser los que se han especificado en la tabla 1. Se pueden conceder varios privilegios en la misma orden separados por comas y cada privilegio se puede otorgar sobre todos los atributos, que es lo que se lleva a cabo por defecto, o sobre uno o varios atributos en concreto, que se deben especificar entre paréntesis y separados por comas. Además de privilegios, se pueden conceder roles.

Se pueden otorgar privilegios a nivel de tabla, en cuyo caso hay que especificar después de ON el nombre de la tabla. Si la tabla se encuentra en una base de datos diferente de la actual, al nombre de la tabla se le debe anteponer el nombre de la base de datos y un punto. Para otorgar un privilegio a nivel de la base de datos actual, es decir, a todas las tablas de la base de datos actual, se especificará después de ON un asterisco (*). Si el privilegio se desea otorgar sobre todas las tablas de una base de datos diferente de la actual, se deberá especificar nombre_BD.*. Se pueden asignar privilegios globales, es decir, a nivel de todas las bases de datos, especificando después de ON *.*. Finalmente, se pueden conceder privilegios a nivel de una rutina (procedimiento o función), en cuyo caso se debe especificar nombre_BD.nombre_rutina.

Los privilegios CREATE ROLE, CREATE TABLESPACE, CREATE USER, DROP ROLE, FILE, PROCESS, RELOAD, SHOW DATABASES, SHUTDOWN y SUPER son permisos administrativos que solo se pueden otorgar globalmente, es decir, a nivel de servidor, empleando la sintaxis *.*.

Los únicos privilegios que se pueden especificar para una tabla son ALTER, CREATE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, GRANT OPTION, INDEX, INSERT, REFERENCES, SELECT, SHOW VIEW, TRIGGER y UPDATE. Los únicos privilegios que se pueden especificar para un atributo son SELECT, INSERT, UPDATE y REFERENCES.

Se pueden conceder los privilegios indicados a uno o varios usuarios y/o roles, que se especifican después de la palabra TO. Para conceder el/los privilegio/s a varios usuarios y/o roles, se deben separar sus nombres por el símbolo coma (,).

La opción WITH GRANT OPTION concede al usuario que recibe el permiso la opción de dar a otros usuarios cualesquiera de sus permisos. Por este motivo, es conveniente asignar este permiso con cautela.

Veamos algunos ejemplos de órdenes GRANT. En primer lugar, creemos el usuario *dba* y asignémosle todos los permisos para hacer cualquier cosa. Además, le vamos a dar la opción de pasar sus permisos a cualquier otro usuario:

```
CREATE USER dba IDENTIFIED BY 'aaaa';

GRANT ALL PRIVILEGES ON *.* TO dba
WITH GRANT OPTION;
```

Concedamos al usuario *jose* la opción de consultar cualquier tabla de la base de datos *Pedidos*:

```
GRANT SELECT ON Pedidos.* TO jose;
```

Ahora entremos en una sesión con el usuario *jose*. Veremos cómo podemos consultar tablas de la base de datos *Pedidos*, pero no podemos por ejemplo insertar datos en la tabla *Pedido*.

```
C:\Users\Jose>cd c:\Program Files\MySQL\MySQL Server 8.0\bin

c:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u jose -P 33060 -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.34 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE pedidos;
Database changed
mysql> SELECT * FROM pedido;
+--------+------------+
| refped | fecped     |
+--------+------------+
| P0001  | 2025-02-16 |
| P0002  | 2025-02-18 |
| P0003  | 2025-02-23 |
| P0004  | 2025-02-25 |
+--------+------------+
4 rows in set (0.14 sec)

mysql> INSERT INTO pedido VALUES ('P0005', CURRENT_DATE);
ERROR 1142 (42000): INSERT command denied to user 'jose'@'localhost' for
table 'pedido'
```

Ahora desde la sesión del usuario *root* concedámosle al usuario *jose* la opción de insertar, modificar y borrar datos sobre la tabla *Pedido*:

```
mysql> USE pedidos;
Database changed
mysql> GRANT insert, update, delete ON Pedidos.pedido TO jose;
Query OK, 0 rows affected (0.05 sec)
```

Desde la sesión del usuario *jose* ahora sí que podremos añadir un pedido y luego eliminarlo sin problema:

```
mysql> INSERT INTO pedido VALUES ('P0005', CURRENT_DATE);
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM pedido;
+--------+------------+
| refped | fecped     |
+--------+------------+
| P0001  | 2025-02-16 |
| P0002  | 2025-02-18 |
| P0003  | 2025-02-23 |
| P0004  | 2025-02-25 |
| P0005  | 2026-03-02 |
+--------+------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM pedido WHERE RefPed = 'P0005';
Query OK, 1 row affected (0.04 sec)
```

Desde la sesión del usuario *root* concedámosle a este usuario la opción de crear tablas, alterarlas y eliminarlas en la base de datos *Pedidos*:

```
GRANT create, alter, drop ON pedidos.* TO jose;
```

Para poder visualizar los privilegios de un determinado usuario se puede hacer uso del comando SHOW con la siguiente sintaxis:

```
SHOW GRANTS FOR Nombre_usuario;
```

Así, para visualizar los permisos que tiene otorgados el usuario *jose* escribiremos lo siguiente:

```
SHOW GRANTS FOR jose;
+-------------------------------------------------------------------+
| Grants for jose@%                                                 |
+-------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jose'@'%'                                  |
| GRANT SELECT, CREATE, DROP, ALTER ON `pedidos`.* TO 'jose'@'%'    |
| GRANT INSERT, UPDATE, DELETE ON `pedidos`.`pedido` TO 'jose'@'%' |
+-------------------------------------------------------------------+
3 rows in set (0.03 sec)
```

Por otro lado, los privilegios concedidos a los usuarios también se pueden retirar o revocar, para lo que se emplea el comando REVOKE, cuya sintaxis es la misma que la del comando GRANT con las siguientes salvedades:

- En vez de GRANT hay que escribir REVOKE.
- En vez de escribir TO antes de los usuarios a los que se les conceden los privilegios, se debe escribir FROM antes de los usuarios a los que se les retiran los privilegios.
- No se puede emplear la opción with.

Por tanto, su sintaxis queda como sigue:

```
REVOKE privilegio1 [(lista_atributos1)] [, privilegio2 [(lista_atributos2)]]…
ON {nombre_tabla | * | *.* | nombre_BD.* | nombre_BD.nombre_tabla |
    nombre_BD.nombre_rutina}
FROM {nombre_usuario1 | nombre_rol1}[,{nombre_usuario2 | nombre_rol2}…];
```

Así, mediante la siguiente orden retiramos al usuario *jose* la posibilidad de añadir datos en la tabla *Pedido*:

```
REVOKE insert ON Pedidos.pedido FROM jose;
```

Si ahora mostramos los privilegios de este usuario, veremos que ya no aparece el privilegio que le acabamos de retirar:

```
SHOW GRANTS FOR jose;
+-------------------------------------------------------------------+
| Grants for jose@%                                                 |
+-------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jose'@'%'                                  |
| GRANT SELECT, CREATE, DROP, ALTER ON `pedidos`.* TO 'jose'@'%' |
| GRANT UPDATE, DELETE ON `pedidos`.`pedido` TO 'jose'@'%'       |
+-------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

3. Roles en MySQL.

Un rol abarca un conjunto de privilegios sobre el sistema y/o sobre objetos de la base de datos. Para crear un rol se utiliza la orden CREATE ROLE con la siguiente sintaxis:

```
CREATE ROLE [IF NOT EXISTS] nombre_rol1, nombre_rol2,…;
```

Una vez creado el rol, es posible asignarle privilegios con la orden GRANT estudiada, de igual manera que se otorgan privilegios a usuarios. Después de asignar al rol los privilegios que se considere que debe tener asignados, es posible asignar ese rol mediante una orden GRANT a uno o varios usuarios. También es posible asignar este rol a un usuario al crearlo mediante CREATE USER y la opción DEFAULT ROLE. Así, los usuarios tendrán a partir de ese momento todos los privilegios que se concedieron al rol asignado. Otra opción para asignar un rol a un usuario ya creado es utilizar la instrucción ALTER USER con la opción DEFAULT ROLE.

Por ejemplo, podemos crear un rol llamado *RolVentas* para ser utilizado por los usuarios del departamento de Ventas. Los usuarios de este departamento deben poder realizar consultas, inserciones, modificaciones y borrados sobre las tablas *Pedido* y *LineaPedido* y deben poder consultar la tabla *Articulo*. Por ello, vamos a crear el rol y asignarle los permisos necesarios para poder realizar todas estas operaciones:

```
CREATE ROLE RolVentas;
GRANT select, insert, update, delete ON Pedidos.Pedido TO RolVentas;
GRANT select, insert, update, delete ON Pedidos.LineaPedido TO RolVentas;
GRANT select ON Pedidos.Articulo TO RolVentas;
```

También se pueden visualizar los privilegios otorgados a un rol mediante la orden *show grants*:

```
SHOW GRANTS FOR RolVentas;
+-----------------------------------------------------------------------------------------+
| Grants for RolVentas@%                                                                  |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `RolVentas`@`%`                                                   |
| GRANT SELECT ON `pedidos`.`articulo` TO `RolVentas`@`%`                                 |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `pedidos`.`lineapedido` TO `RolVentas`@`%` |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `pedidos`.`pedido` TO `RolVentas`@`%`      |
+-----------------------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

Ahora cada vez que se incorporen nuevos empleados al departamento de Ventas, con asignarles el rol *RolVentas*, recibirán de forma conjunta todos los privilegios que tiene concedidos este rol. Así, si se incorporan a este departamento dos usuarios (*luis* y *loli*), solo los tendremos que crear y asignarles este rol para que puedan trabajar:

```
CREATE USER loli IDENTIFIED BY '1111', luis IDENTIFIED BY '2222'
DEFAULT ROLE RolVentas;
```

Podemos visualizar los privilegios del usuario *luis* con la siguiente orden:

```
SHOW GRANTS FOR luis;
+-------------------------------------+
| Grants for luis@%                   |
+-------------------------------------+
| GRANT USAGE ON *.* TO `luis`@`%`    |
| GRANT `RolVentas`@`%` TO `luis`@`%` |
+-------------------------------------+
2 rows in set (0.00 sec)
```

Esto nos indica que el usuario *luis* tiene concedido el rol *RolVentas*. Si no sabemos qué privilegios conlleva este rol, los podemos consultar mediante la orden:

```
SHOW GRANTS FOR RolVentas;
+-----------------------------------------------------------------------------------------+
| Grants for RolVentas@%                                                                  |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `RolVentas`@`%`                                                   |
| GRANT SELECT ON `pedidos`.`articulo` TO `RolVentas`@`%`                                 |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `pedidos`.`lineapedido` TO `RolVentas`@`%` |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `pedidos`.`pedido` TO `RolVentas`@`%`      |
+-----------------------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

La concesión de un rol a un usuario mediante la orden GRANT no surtirá efecto automáticamente porque por defecto los roles concedidos de esta manera no están activos. Se puede consultar el rol activo en una determinada sesión ejecutando la instrucción SELECT CURRENT_ROLE(); Si se indica NONE, querrá decir que no está activo ningún rol.

Si queremos que todas las asignaciones de roles con el comando GRANT se activen automáticamente, debemos asignar el valor 1 a la variable del sistema *activate_all_roles_on_login* en el archivo *my.ini*.

Para eliminar un rol se debe hacer uso de la instrucción DROP ROLE, cuyo formato es el siguiente:

```
DROP ROLE [IF EXISTS] nombre_rol1, nombre_rol2,…;
```

4. Vistas.

Las vistas son objetos de la base de datos que incluyen mediante una consulta un subconjunto de datos de la base de datos. A veces se llama a las vistas "tablas virtuales" porque se puede trabajar con ellas en la mayoría de los casos como si fuesen tablas, pero no lo son porque las vistas no contienen datos almacenados. Las tablas que aparecen en la consulta asociada a una vista reciben el nombre de tablas subyacentes a la vista. Cada vez que se realiza una operación sobre una vista, se ejecuta la sentencia SELECT asociada a la vista para obtener los datos de la misma. Las vistas no almacenan datos excepto si se trata de vistas materializadas.

Las vistas pueden ser útiles por diversos motivos:

- Una vista simplifica la complejidad de una consulta porque se puede realizar una consulta sobre una vista basada en una consulta compleja mediante una SELECT sencilla.
- Las vistas permiten que el administrador de la base de datos solo ponga a disposición de determinados usuarios aquellos datos a los que estos deben poder acceder creando las vistas correspondientes. Para los usuarios ver los datos contenidos en tablas o en vistas es exactamente igual, pero no para el administrador, quien debe velar por la integridad de los datos. Además, es posible dar permisos sobre vistas como si se tratase de tablas, permitiendo de esta manera proteger las tablas originales.
- Una vista proporciona una capa consistente incluso si cambian las columnas de la tabla o tablas subyacentes.

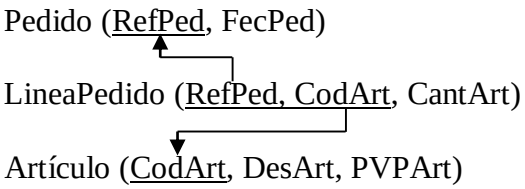
Para crear una vista se debe emplear la sentencia CREATE VIEW, cuyo formato es el siguiente:

```
CREATE [OR REPLACE]
VIEW Nombre_Vista [(campo1, campo2, …, campon)]
AS sentencia_select
[WITH CHECK OPTION];
```

La opción OR REPLACE sirve para que en el caso de que ya exista una vista con el nombre indicado, no se produzca un mensaje de error y esa vista sea sustituida por la que se está creando.

A toda vista es necesario asignarle un nombre, que no puede coincidir con el nombre de ninguna otra vista ni tabla dentro de la base de datos actual. Después de indicar el nombre de la vista se pueden especificar entre paréntesis y separados por comas los nombres de sus atributos. En caso de omitir esta lista, se asignan a los atributos de la vista los mismos nombres que aparecen en la sentencia SELECT con la que se crea la vista.

Vamos a trabajar con vistas a partir de algunas tablas de la base de datos *Pedidos*, cuyo esquema relacional se muestra a continuación:



A modo de ejemplo, se va a crear una vista llamada *ArticulosBaratos* con la descripción y precio de los artículos cuyo precio es inferior a 0,50 €. En este caso no se van a especificar los nombres de los atributos de la vista, por lo que coincidirán con los de la tabla *Articulo* sobre la que se define:

```
mysql> CREATE VIEW ArticulosBaratos
    -> AS SELECT DesArt, PVPArt FROM Articulo WHERE PVPArt < 0.50;
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM ArticulosBaratos;
+----------------+--------+
| DesArt         | PVPArt |
+----------------+--------+
| Goma de borrar |   0.15 |
| Sacapuntas     |   0.25 |
+----------------+--------+
2 rows in set (0.00 sec)
```

En la sentencia SELECT asociada a la vista pueden aparecer varias tablas y/o vistas en su cláusula FROM.

Si a algún atributo de la vista se le desea asignar un nombre diferente de lo que viene especificado en la cláusula SELECT de la consulta con la que se crea la vista, se puede obrar de una de las dos siguientes maneras:

- Asignar un alias al atributo en la sentencia SELECT.
- Escribir después del nombre de la vista dentro de un paréntesis los nombres de todos sus atributos separados por comas.

A modo de ejemplo, vamos a crear una vista que contenga por cada artículo solicitado en más de un pedido, su código, descripción, el número de pedidos en que ha sido solicitado y el número de unidades totales solicitadas del artículo. Para obtener estos dos últimos datos tenemos que realizar agrupamientos y aplicar funciones de grupo (*count* y *sum*) a dos atributos. La podemos crear con la siguiente instrucción:

```
CREATE VIEW ArticulosPedidos AS
SELECT A.CodArt, DesArt, COUNT(RefPed), SUM(CantArt)
FROM LineaPedido L JOIN Articulo A ON L.CodArt = A.CodArt
GROUP BY A.CodArt, DesArt
HAVING COUNT(RefPed) > 1;
```

```
mysql> SELECT * FROM ArticulosPedidos;
+--------+---------------+--------------+--------------+
| CodArt | DesArt        | count(RefPed) | sum(CantArt) |
+--------+---------------+--------------+--------------+
| A0043  | Bolígrafo azul |             3 |           20 |
+--------+---------------+--------------+--------------+
1 row in set (0.00 sec)
```

Pero de esta manera hay dos atributos en la vista (los dos últimos), cuyos nombres pueden no resultar muy adecuados. Para asignarles otros nombres podríamos haber ejecutado una de las dos siguientes órdenes:

```
CREATE VIEW ArticulosPedidos (CodArt, DesArt, NumPed, Unidades) AS
SELECT A.CodArt, DesArt, COUNT(RefPed), SUM(CantArt)
FROM LineaPedido L JOIN Articulo A ON L.CodArt = A.CodArt
GROUP BY A.CodArt, DesArt
HAVING COUNT(RefPed) > 1;

CREATE VIEW ArticulosPedidos AS
SELECT A.CodArt, DesArt, COUNT(RefPed) NumPed, SUM(CantArt) Unidades
FROM LineaPedido L JOIN Articulo A ON L.CodArt = A.CodArt
GROUP BY A.CodArt, DesArt
HAVING COUNT(RefPed) > 1;
```

Sobre las vistas no solo se pueden realizar consultas, sino también inserciones, borrados y modificaciones, que afectarán a los datos reales almacenados en la tabla o las tablas subyacentes a la vista.

Para que una vista sea actualizable, es decir, para poder realizar inserciones, borrados y modificaciones sobre la vista, se deben cumplir las siguientes condiciones:

- Debe haber una relación uno a uno entre los registros de la vista y los registros de la tabla subyacente.
- No debe haber en la definición de la vista funciones de resumen (SUM, MAX, MIN, etc.), ni tampoco cláusulas GROUP BY, HAVING y DISTINCT.
- No debe haber en la definición de la vista uniones (operador UNION) ni reuniones externas (OUTER JOIN).

Una vista actualizable puede contener columnas actualizables y no actualizables. Una columna es actualizable si es una referencia simple a una columna de la tabla subyacente; en caso contrario, la columna será no actualizable o de solo lectura. Por tanto, una columna será no actualizable cuando contenga una expresión.

Si la vista es actualizable, el sistema convertirá cualquier sentencia INSERT, UPDATE o DELETE sobre la vista en la correspondiente sentencia sobre la tabla subyacente.

Si una vista actualizable contiene una cláusula WHERE, la condición limita las filas de la tabla subyacente disponibles para su modificación por medio de instrucciones UPDATE o

DELETE sobre la vista. Sin embargo, una sentencia UPDATE puede modificar una fila de forma que tras esta modificación dicha fila ya no satisfaga la condición de la cláusula WHERE para aparecer en la vista. De forma similar, una sentencia INSERT puede añadir filas en la tabla subyacente que no satisfagan la cláusula WHERE y que, por tanto, no sean visibles por medio de la vista. Si se especificó la opción CHECK OPTION cuando se creó la vista, al hacer cualquier INSERT o UPDATE, se comprueba que las nuevas filas sean visibles por medio de la vista; si no es el caso, se rechaza la operación correspondiente. Si no se especifica CHECK OPTION, se permiten sentencias INSERT o UPDATE sobre la vista que creen filas no visibles por medio de la vista.

Para que una orden UPDATE sobre una vista sea correcta, además de las condiciones indicadas anteriormente para que una vista sea actualizable, se deben cumplir las siguientes condiciones:

- Todos los atributos que se modifiquen mediante la orden UPDATE deben ser actualizables. Un atributo es actualizable si es una columna simple de la tabla subyacente, o sea, si no es una expresión.
- Si la vista tiene varias tablas subyacentes, solo se pueden modificar atributos de una tabla, es decir, en la cláusula SET no se puede asignar valor a atributos de varias de las tablas subyacentes a la vista.

Para poder realizar inserciones en una vista, además de las condiciones para que una vista sea actualizable, se deben cumplir las dos condiciones indicadas a continuación:

- La vista debe contener todos los atributos obligatorios de la tabla que no tengan definido en la tabla valor por defecto.
- Todas las columnas de la vista deben ser columnas actualizables.

A modo de ejemplo, se va a crear a partir de la tabla *Empleado* de la base de datos *Empresa*, una vista con el nombre, puesto, salario y comisión de los empleados del departamento número 3.

```
CREATE VIEW emp3
AS SELECT NomEmp, Puesto, Salario, Comision
FROM Empleado WHERE NumDep = 3;
```

Vamos a realizar una inserción sobre esta vista. Obviamente el SGBD intentará realizar la inserción sobre la tabla *Empleado*, que es la tabla subyacente a la vista.

```
mysql> INSERT INTO emp3
    -> VALUES ('Emilio García Bello', 'Empleado', 2350, 100);
ERROR 1423 (HY000): Field of view 'empresa_m.emp3' underlying table doesn't
have a default value.
```

No se ha podido llevar a cabo la inserción porque hay dos atributos de la tabla *Empleado* (*NumEmp* y *FecIngreso*) que no aparecen en la vista, en la tabla tienen asignada la restricción

not null y no tienen asignado valor por defecto. Ocurre que el SGBD no sabe qué valor asignar a estos atributos. El mensaje que nos muestra MySQL indica que el valor que se desea asignar al atributo *NumEmp* es incorrecto porque realmente no se proporciona valor para dicho atributo en la orden *insert*.

Ahora vamos a ver si es posible eliminar al empleado llamado Pedro Gómez Sanz de la tabla *Empleado* por medio de la vista. Se debería poder por cumplirse todas las condiciones especificadas anteriormente para que una vista sea actualizable.

```
mysql> DELETE FROM emp3
    -> WHERE NomEmp = 'Pedro Gómez Sanz';
Query OK, 1 row affected (0.00 sec)
```

Y podemos ver consultando la tabla *Empleado* que el empleado llamado Pedro Gómez Sanz ha sido en efecto eliminado.

Vamos a crear ahora una vista llamada *SalariosAnuales* con los números de empleado, nombres y salarios anuales de los empleados de la tabla *Empleado* que trabajan en el departamento número 2. A los atributos de la vista los vamos a llamar *NumEmp*, *NomEmp* y *SalAnuEmp*, respectivamente. Hemos de tener en cuenta que los salarios en la tabla *Empleado* son mensuales, por lo que para obtener los anuales, si no consideramos pagas extra, habrá que multiplicarlos por 12. Crearemos la vista con la siguiente sentencia:

```
CREATE VIEW SalariosAnuales
AS SELECT NumEmp, NomEmp, Salario * 12 SalAnuEmp
FROM Empleado
WHERE NumDep = 2;

mysql> SELECT * FROM SalariosAnuales;
+--------+--------------------+-----------+
| NumEmp | NomEmp             | SalAnuEmp |
+--------+--------------------+-----------+
|      4 | Albert Rius García |  37200.00 |
|      5 | Georgina Ruiz Plá  |  17040.00 |
|      6 | Laura Díaz Folgado |  15840.00 |
+--------+--------------------+-----------+
3 rows in set (0.00 sec)
```

A través de esta vista podremos realizar modificaciones de atributos de la tabla *Empleado* con la condición de que el atributo que se modifica en la vista *SalariosAnuales* sea un atributo actualizable, es decir, no sea un atributo definido mediante una expresión, como ocurre con el atributo *SalAnuEmp*. Así, por ejemplo, podremos cambiar el primer apellido de Laura Díaz Folgado por Díez:

```
mysql> UPDATE SalariosAnuales
    -> SET NomEmp = 'Laura Díez Folgado'
    -> WHERE NomEmp = 'Laura Díaz Folgado';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT * FROM Empleado;
+--------+----------------------+----------+-----------+------------+---------+----------+--------+
| NumEmp | NomEmp               | Puesto   | NumEmpJefe | FecIngreso | Salario | Comision | NumDep |
+--------+----------------------+----------+-----------+------------+---------+----------+--------+
|      1 | Alberto Rey Ruiz     | Gerente  |       NULL | 2014-01-02 | 5500.00 |     0.00 |      1 |
|      2 | Luis Grande Gil      | Director |          1 | 2014-01-02 | 3200.00 |     0.00 |      1 |
|      3 | Ana Ruiz Almeida     | Empleado |          2 | 2014-01-02 | 1525.00 |     0.00 |      1 |
|      4 | Albert Rius García   | Director |          1 | 2016-02-02 | 3100.00 |     0.00 |      2 |
|      5 | Georgina Ruiz Plá    | Empleado |          4 | 2016-02-02 | 1420.00 |     0.00 |      2 |
|      6 | Laura Díez Folgado   | Empleado |          4 | 2016-12-12 | 1320.00 |     0.00 |      2 |
…
|     11 | María Galiano Lastra | Vendedor |         10 | 2020-01-15 | 1300.00 |   900.00 |      3 |
+--------+----------------------+----------+-----------+------------+---------+----------+--------+
11 rows in set (0.00 sec)
```

Sin embargo, si intentamos modificar su salario, al no ser este un atributo actualizable o estar definido mediante una expresión, se producirá un error:

```
mysql> UPDATE SalariosAnuales
    -> SET SalAnuEmp = 17000
    -> WHERE NomEmp = 'Laura Díez Folgado';
ERROR 1348 (HY000): Column 'SalAnuEmp' is not updatable
```

Se puede modificar la sentencia SELECT asociada a una vista mediante la sentencia ALTER VIEW, cuya sintaxis es la siguiente:

```
ALTER VIEW Nombre_Vista [(campo1, campo2, …, campon)]
AS sentencia_select
```

Por ejemplo, vamos a realizar una modificación sobre la vista *SalariosAnuales* creada anteriormente, de manera que el salario anual tenga en cuenta también dos pagas extras al año (14 pagas mensuales en lugar de 12):

```
ALTER VIEW SalariosAnuales
AS SELECT NumEmp, NomEmp, Salario * 14 SalAnuEmp
FROM Empleado
WHERE NumDep = 2;

mysql> SELECT * FROM SalariosAnuales;
+--------+--------------------+-----------+
| NumEmp | NomEmp             | SalAnuEmp |
+--------+--------------------+-----------+
|      4 | Albert Rius García |  43400.00 |
|      5 | Georgina Ruiz Plá  |  19880.00 |
|      6 | Laura Díez Folgado |  18480.00 |
+--------+--------------------+-----------+
3 rows in set (0.00 sec)
```

Para eliminar una o varias vistas de la base de datos, se usa la sentencia DROP VIEW, cuya sintaxis es la siguiente:

```
DROP VIEW [IF EXISTS] vista1, vista2, …
```

La opción IF EXISTS sirve para que si no existe alguna de las vistas indicadas no se muestre el mensaje de error que aparece por defecto. Por ejemplo, para borrar la vista *SalariosAnuales*, deberemos escribir:

```
mysql> DROP VIEW SalariosAnuales;
Query OK, 0 rows affected (0.00 sec)
```

5. Sinónimos.

El concepto de sinónimo no existe en MySQL, pero sí en otros SGBDs como Oracle. Así, en Oracle, cuando un usuario tiene acceso a una tabla de otro usuario y quiere realizar cualquier operación sobre la misma, debe anteponer al nombre de la tabla el nombre del usuario y un punto. Por ejemplo, si el usuario *jose* tiene acceso a la tabla *Departamento* de *santi*, para consultar dicha tabla tendrá que teclear la orden:

```
select * from santi.Departamento;
```

Mediante el uso de sinónimos *jose* puede crear un sinónimo para referirse a la tabla de *santi* sin necesidad de incluir su nombre.

Un sinónimo es un nuevo nombre que se puede dar a una tabla o a una vista. Con los sinónimos se pueden utilizar dos nombres para referirse a un mismo objeto de la base de datos. Resulta útil sobre todo para acceder a tablas y vistas de otros usuarios. Para crear un sinónimo se utiliza la sintaxis:

```
CREATE SYNONYM nombre_sinónimo FOR [nombre_usuario.]nombre_tabla;
```

Para eliminar sinónimos se empleará la siguiente sintaxis:

```
DROP SYNONYM [nombre_usuario.]nombre_sinónimo;
```

6. Índices.

Los índices son una herramienta útil para incrementar el rendimiento de la base de datos y se pueden definir como estructuras de datos que aceleran las operaciones de búsqueda basadas en los campos sobre los que se definen los índices. No obstante, los índices también suponen una sobrecarga para la base de datos, por lo que se deben manejar con cuidado.

Supongamos que disponemos de la siguiente consulta sobre la tabla *Articulo* de la base de datos *Pedidos*:

```
SELECT PVPArt FROM Articulo WHERE DesArt = 'Bolígrafo rojo normal';
```

Pues bien, para devolver el resultado de la consulta, el SGBD debería recorrer todas las filas de la tabla *Articulo* para encontrar aquellas filas que cumplen la condición especificada. Si hay muchas filas en la tabla *Articulo* y solo unas pocas filas (cero o una, en este caso) que devolver, este método resulta muy ineficiente. Pero si existiese un índice sobre el atributo *DesArt*, se podría usar un método más eficiente para encontrar las filas que cumplen la condición especificada en la cláusula WHERE. Por ejemplo, solo debería recorrer varios niveles de profundidad en un árbol de búsqueda.

Una vez que se crea un índice, no se requiere mayor intervención, ya que el sistema actualizará el índice cuando se vean modificados los datos de la tabla y utilizará el índice en consultas cuando considere que hacerlo sería más eficiente que un recorrido secuencial de la tabla.

Después de crear un índice, el sistema debe mantenerlo sincronizado con la tabla, lo que añade una sobrecarga a las operaciones de manipulación de datos. Debido a esto, aquellos índices que no sean usados apenas o nunca deben ser eliminados.

Además de crear índices en el momento en que se crea una tabla (dentro de la instrucción CREATE TABLE) escribiendo *index (atr1, atr2, ...)*, también es posible hacerlo con posterioridad mediante el comando CREATE INDEX. MySQL crea índices automáticamente para los campos únicos, clave primaria y claves ajenas.

Los índices no tienen por qué abarcar la totalidad de un atributo, sino que pueden ser parciales, lo que se lleva a cabo básicamente sobre campos largos de tipo texto.

Además, los índices pueden ser multicolumna, es decir, pueden abarcar más de un atributo de una tabla, lo que puede resultar de utilidad cuando se usan esos atributos en diversas cláusulas de sentencias SELECT.

La instrucción para crear un índice es CREATE INDEX, que requiere de la siguiente sintaxis:

```
CREATE INDEX nombre_índice
ON nombre_tabla (atributo_índice, ….)

atributo_índice: nombre_atributo [(longitud)] [{ASC | DESC}]
```

Como se puede observar, después de la palabra INDEX se debe indicar el nombre que se le asigna al índice. Hay que indicar después de la palabra ON la tabla sobre la que se crea el índice y el o los atributos correspondientes. Los índices no tienen por qué ser completos, sino que pueden ser parciales, indicando en tal caso la longitud que abarcan entre paréntesis. Además, se puede indicar si se desea que el índice sea en orden ascendente o descendente. El orden por defecto es ascendente.

Por ejemplo, vamos a crear un índice parcial sobre los 10 primeros caracteres de la descripción de un artículo:

```
mysql> CREATE INDEX IDesArt
    -> ON Articulo(DesArt(10));
Query OK, 0 rows affected (0.51 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Creemos ahora otro índice sobre el atributo *PVPArt* en orden descendente. Usaremos la siguiente orden:

```
mysql> CREATE INDEX IPVPArt
    -> ON Articulo (PVPArt DESC);
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Para visualizar los índices definidos sobre una tabla se puede usar el comando SHOW INDEX, cuya sintaxis es la siguiente:

```
SHOW {INDEX | INDEXES | KEYS}
{FROM | IN} nombre_tabla
[{FROM | IN} nombre_BD]
```

Como se puede observar, se puede indicar después de SHOW tanto INDEX, como INDEXES, como KEYS. Se debe indicar después FROM o IN y la tabla cuyos índices se desean mostrar, y si no es una tabla de la base de datos actual, FROM o IN y la base de datos en cuestión. Así, para visualizar los índices que hay definidos sobre la tabla *Articulo* de la base de datos actual, escribiremos:

```
mysql> SHOW INDEXES FROM Articulo;
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| articulo |          0 | PRIMARY  |            1 | CodArt      | A         |           5 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| articulo |          1 | IDesArt  |            1 | DesArt      | A         |           4 |       10 |   NULL |      | BTREE      |         |               | YES     | NULL       |
| articulo |          1 | IPVPArt  |            1 | PVPArt      | D         |           5 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
3 rows in set (0.00 sec)
```

Se puede observar cómo además de los dos índices que se acaban de crear explícitamente con la instrucción CREATE INDEX (*IDesArt* e *IPVPArt*), MySQL creó implícitamente un índice para el atributo clave primaria de la tabla (*CodArt*).

No existe una instrucción ALTER INDEX para realizar modificaciones sobre los índices creados.

Por su parte, la instrucción para eliminar un índice es la instrucción DROP INDEX, cuya sintaxis es la siguiente:

```
DROP INDEX nombre_índice ON nombre_tabla
```

Como se puede observar, se debe indicar el nombre del índice y el nombre de la tabla sobre la que se ha definido. Así, para borrar el índice *IPVPArt* definido sobre el atributo *PVPArt* de la tabla *Articulo*, deberemos emplear la siguiente instrucción:

```
mysql> drop index IPVPArt ON Articulo;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```