

UNIDAD 2: ACCESO A LA INFORMACIÓN

Resultados de Aprendizaje a trabajar en esta Unidad:
RA3: Implanta métodos de control de acceso utilizando asistentes, herramientas gráficas y comandos del lenguaje del sistema gestor.
Contenidos a trabajar en esta Unidad:
- Creación, modificación y eliminación de vistas. - Creación y eliminación de usuarios. - Asignación de derechos a usuarios. Puntos de acceso al sistema. - Definición de roles. Asignación de roles a usuarios.

Contenidos de la Unidad:

1. ACCESO A LA INFORMACIÓN 2
2. VISTAS 2
3. GESTIÓN DE USUARIOS 9
4. PRIVILEGIOS DE MYSQL 13
5. GESTIÓN DE USUARIOS MEDIANTE INTERFAZ GRÁFICA 17
6. BIBLIOGRAFÍA 19

1. ACCESO A LA INFORMACIÓN

Cuando se administra la seguridad en el acceso a la información de una base de datos es común utilizar dos tipos de seguridad: la integrada con el sistema operativo y la proporcionada por el SGBD.

En la seguridad integrada se suele contar con los usuarios de un sistema de dominio o un servicio de directorio para proporcionar el acceso a determinados recursos.

En la seguridad nativa del SGBD es el propio software servidor el que proporciona los mecanismos mediante los cuales se autoriza a un usuario a utilizar distintos elementos de las bases de datos.

En esta unidad trataremos la seguridad nativa de un SGBD a través de SQL explicando el funcionamiento básico de la seguridad en MySQL.

2. VISTAS

Una vista (VIEW) es un objeto de una base de datos que permite consultar los datos mediante una consulta SQL almacenada. Se comporta de forma similar a una tabla, pero normalmente no almacena los resultados de la consulta, sino que guarda la definición de la consulta que debe ejecutarse cuando se accede a ella. Por este motivo, las vistas también se conocen como tablas virtuales.

Imaginemos una base de datos de una tienda que contiene varias tablas:

- Clientes: información de los clientes.
- Pedidos: información de los pedidos realizados.
- DetallePedidos: productos incluidos en cada pedido, cantidades y precios.

Si queremos consultar frecuentemente los pedidos junto con el nombre del cliente, tendríamos que realizar una consulta con varias tablas relacionadas. Podemos crear una vista para guardar esa consulta y utilizarla posteriormente de una manera mucho más sencilla.

Por ejemplo, sin utilizar una vista, podríamos escribir:

```
SELECT p.codigoPedido, c.nombreCliente
FROM Clientes c JOIN Pedidos p ON c.codigoCliente = p.codigoCliente;
```

Si guardamos esta consulta en una vista llamada `vistaPedidos`, podremos consultar esos mismos datos escribiendo:

```
SELECT * FROM vistaPedidos;
```

La vista permite reutilizar la consulta sin tener que escribir de nuevo toda su definición.

2.1. Almacenamiento de los datos de una vista

La vista no almacena datos, únicamente almacena la definición de una consulta SQL. Los datos se obtienen al ejecutar la consulta de la vista.

Por ejemplo, si una vista muestra el nombre de un cliente y su ciudad, y posteriormente se modifica la ciudad de ese cliente en la tabla original, la vista mostrará el valor actualizado cuando se vuelva a consultar.

Esto sucede porque una vista normal no es una copia de los datos en el momento de su creación: consulta los datos actuales de las tablas implicadas en el momento que se consulta la vista.

2.2. Aplicaciones de las vistas

Las vistas tienen diferentes aplicaciones en la administración y el acceso a bases de datos.

- **Simplificar consultas complejas**

Una consulta que incluye varias tablas, condiciones, funciones de agregación o agrupaciones puede resultar difícil de escribir y mantener. Si la consulta se utiliza con frecuencia, podemos guardarla en una vista y trabajar directamente con ella.

- **Reutilizar consultas**

Cuando una consulta se necesita en diferentes aplicaciones o procesos, una vista permite definirla una vez y utilizarla tantas veces como sea necesario.

- **Controlar el acceso a los datos**

Una vista puede mostrar únicamente determinadas columnas o registros de una tabla. Así, es posible ofrecer a determinados usuarios acceso a una parte de la información sin darles acceso directo a toda la tabla.

Por ejemplo, una vista de empleados podría mostrar su nombre, departamento y puesto, pero no su salario. Para que esta medida funcione como control de seguridad, debe acompañarse de una configuración adecuada de los permisos de MySQL.

- **Presentar los datos de una forma más comprensible**

Las vistas permiten utilizar nombres de columnas más claros, combinar información de diferentes tablas y presentar resultados calculados.

2.3. Creación de vistas

Para crear una vista se utiliza la sentencia `CREATE VIEW`.

```
CREATE VIEW nombreVista AS SELECT columnas FROM tabla WHERE condicion;
```

La sentencia está formada por los siguientes elementos:

- `CREATE VIEW`: indica que se quiere crear una vista.

- `nombreVista`: nombre que recibirá la vista.
- `AS`: introduce la consulta que define la vista.
- `SELECT`: consulta que determina qué datos mostrará.

La consulta utilizada para definir la vista puede ser sencilla o incluir operaciones más complejas, como uniones entre tablas, subconsultas, agrupaciones y funciones de agregación.

Por ejemplo:

```
CREATE VIEW vistaClientes AS SELECT nombreCliente, ciudad FROM Clientes;
```

Para consultar el contenido de la vista:

```
SELECT * FROM vistaClientes;
```

También podemos realizar consultas sobre la vista:

```
SELECT nombreCliente FROM vistaClientes WHERE ciudad = 'Oviedo';
```

También podemos indicar explícitamente los nombres que tendrán las columnas de una vista.

```
CREATE VIEW vistaClientes (nombre, localidad) AS
                        SELECT nombreCliente, ciudad FROM Clientes;
```

En este caso, la vista tendrá las columnas nombre y localidad, aunque en la tabla original se llamen `nombreCliente` y `ciudad`. Esto resulta útil cuando queremos presentar nombres más sencillos o adaptar la estructura de la vista a las necesidades de una aplicación.

Una de las aplicaciones más interesantes de las vistas es guardar consultas que combinan información de varias tablas.

Supongamos que tenemos estas tablas:

- Clientes, que almacena los datos de los clientes.
- Pedidos, que almacena los pedidos y el código del cliente que los realizó.

Podemos crear una vista que muestre cada pedido junto con el nombre de su cliente:

```
CREATE VIEW vistaPedidos AS SELECT p.codigoPedido, c.nombreCliente
FROM Clientes c JOIN Pedidos p ON c.codigoCliente = p.codigoCliente;
```

Para consultar la vista:

```
SELECT * FROM vistaPedidos;
```

La consulta original queda almacenada en la definición de la vista. Cada vez que ejecutemos el SELECT sobre ella, MySQL resolverá la consulta y devolverá los datos correspondientes.

En una vista se pueden utilizar diferentes tipos de JOIN, subconsultas y otras construcciones SQL, según las necesidades del resultado.

Las vistas también permiten guardar consultas que calculan resultados a partir de los datos de las tablas.

Por ejemplo, podemos crear una vista que calcule el importe total de cada pedido.

Supongamos que `detallePedidos` contiene las columnas `codigoPedido, Cantidad y PrecioUnidad.`

```
CREATE VIEW vistaImportesPedidos AS SELECT codigoPedido, SUM(Cantidad * PrecioUnidad) AS totalPedido FROM detallePedidos GROUP BY codigoPedido;
```

Esta vista devuelve una fila por pedido, con el importe total calculado a partir de sus líneas de detalle.

2.4. Consulta y administración de vistas

Una vez creada una vista, es necesario saber cómo consultar sus resultados, comprobar su definición, modificarla y eliminarla.

Para consultar las vistas creadas, ya hemos visto que se utiliza `SELECT`, igual que para consultar una tabla:

```
SELECT * FROM vistaClientes;
```

Podemos seleccionar determinadas columnas:

```
SELECT nombre, localidad FROM vistaClientes;
```

Y aplicar condiciones:

```
SELECT * FROM vistaClientes WHERE localidad = 'Gijón';
```

Para consultar las vistas de la base de datos seleccionada, podemos utilizar:

```
SHOW FULL TABLES WHERE Table_type = 'VIEW';
```

Este comando permite distinguir las vistas de las tablas normales. También se puede consultar el listado de vistas de una base de datos desde MySQL Workbench.

Para consultar la definición de una vista, es decir, ver la sentencia SQL con la que se ha creado una vista, se utiliza:

```
SHOW CREATE VIEW vistaClientes;
```

Este comando permite comprobar la consulta que define la vista y otros detalles de su configuración. Es especialmente útil cuando administramos una base de datos que contiene vistas creadas por otras personas y necesitamos conocer su funcionamiento.

2.5. Modificar una vista

Si queremos cambiar la consulta que define una vista, podemos utilizar `CREATE OR REPLACE VIEW`.

```
CREATE OR REPLACE VIEW vistaClientes AS SELECT nombreCliente, ciudad
FROM Clientes WHERE ciudad = 'Oviedo';
```

Esta sentencia crea la vista si todavía no existe o sustituye su definición si ya existe.

En el ejemplo, la vista pasa a mostrar solamente los clientes de Oviedo.

Otra alternativa es utilizar `ALTER VIEW`:

```
ALTER VIEW vistaClientes AS SELECT nombreCliente, ciudad FROM Clientes WHERE ciudad = 'Gijón';
```

Estas sentencias modifican la definición de la vista, no la estructura de las tablas originales.

2.6. Eliminar una vista

Para eliminar una vista se utiliza `DROP VIEW`.

```
DROP VIEW vistaClientes;
```

Si la vista no existe, MySQL devolverá un error. Para evitarlo, podemos utilizar:

```
DROP VIEW IF EXISTS vistaClientes;
```

Eliminar una vista no elimina los datos de las tablas originales, ya que la vista es un objeto que contiene la definición de una consulta.

2.7. Vistas actualizables y no actualizables

No todas las vistas sirven únicamente para consultar datos. Algunas permiten realizar operaciones de modificación sobre las tablas originales.

Es necesario distinguir entre:

- Vista actualizable: permite realizar determinadas operaciones de modificación, como `UPDATE`, `INSERT` o `DELETE`, según su definición y las restricciones aplicables.
- Vista no actualizable: permite consultar los datos, pero no admite esas operaciones de modificación a través de la vista.

Supongamos que creamos una vista que muestra el nombre y el departamento de cada empleado:

```
CREATE VIEW vistaEmpleados AS SELECT idEmpleado, nombre, departamento
FROM Empleados;
```

En una vista sencilla como esta, que conserva una relación directa entre sus filas y las de la tabla original, puede ser posible modificar los datos a través de ella.

Por ejemplo:

```
UPDATE vistaEmpleados SET departamento = 'Informática' WHERE idEmpleado = 1;
```

Si la vista es actualizable y se cumplen los permisos y las restricciones necesarias, se modificará el departamento del empleado en la tabla Empleados. La modificación se realiza sobre la tabla original. La vista no mantiene una copia independiente del dato.

Una vista puede no ser actualizable cuando su definición incluye determinadas operaciones que impiden relacionar de manera directa cada fila del resultado con una fila de la tabla original.

Entre los casos habituales están:

- Funciones de agregación, como `SUM()`, `AVG()` o `COUNT()`.
- Agrupaciones mediante `GROUP BY`.
- La eliminación de duplicados mediante `DISTINCT`.
- Determinadas consultas con `UNION`.
- Determinadas subconsultas y combinaciones de tablas.

La actualizabilidad depende de las reglas de MySQL y de la definición concreta de la vista. No basta con que la vista contenga un `SELECT` para saber si permitirá modificar datos.

Por ejemplo, no tendría sentido modificar directamente una fila de la vista `vistaImportesPedidos` para cambiar el resultado de `SUM(Cantidad * PrecioUnidad)`: ese importe se calcula a partir de las líneas del pedido. Para cambiarlo, habría que modificar los datos de las tablas que intervienen en el cálculo.

2.8. Cláusula `WITH CHECK OPTION`

Cuando una vista es actualizable y tiene una condición `WHERE`, puede ser necesario impedir que se inserten o modifiquen registros que dejen de cumplir esa condición.

Para ello existe la cláusula `WITH CHECK OPTION`.

Supongamos que creamos una vista que muestra exclusivamente los empleados del departamento de Informática:

```
CREATE VIEW vistaEmpleadosInformatica AS SELECT idEmpleado, nombre, departamento FROM Empleados WHERE departamento = 'Informática' WITH CHECK OPTION;
```

La vista muestra únicamente las filas que cumplen esta condición:

```
departamento = 'Informática'
```

Con `WITH CHECK OPTION`, las operaciones de inserción o actualización realizadas a través de la vista deben respetar la condición correspondiente.

Por ejemplo, si intentamos modificar un empleado para asignarle el departamento de Ventas a través de esta vista, la operación será rechazada porque el registro dejaría de cumplir el criterio que define la vista.

Esta cláusula ayuda a mantener la coherencia entre las operaciones realizadas a través de una vista y los criterios que esta representa.

2.9. Vistas como medida de seguridad

Las vistas pueden utilizarse como parte de una estrategia de seguridad de una base de datos.

Imaginemos que una tabla Empleados contiene identificador, nombre, departamento, salario y número de cuenta bancaria.

Un usuario de la empresa puede necesitar consultar los nombres y departamentos de los empleados, pero no necesita conocer sus salarios ni sus cuentas bancarias.

Podemos crear una vista que exponga solamente la información necesaria:

```
CREATE VIEW vistaDatosPublicosEmpleados AS SELECT idEmpleado, nombre, departamento FROM Empleados;
```

Después, el administrador puede conceder permisos sobre esa vista al usuario correspondiente, por ejemplo, permisos de consulta:

```
GRANT SELECT ON miBaseDatos.vistaDatosPublicosEmpleados
                                    TO 'usuario_consulta'@'localhost';
```

La idea es que el usuario pueda consultar los datos autorizados a través de la vista, sin necesitar acceso directo a todas las columnas de la tabla original.

No obstante, crear una vista no restringe automáticamente el acceso a la tabla original. Si el usuario tiene permisos directos sobre la tabla, podría consultar también las columnas autorizadas por esos permisos.

Por ello, el administrador debe configurar correctamente los privilegios de usuarios y objetos. MySQL permite controlar el contexto de seguridad con el que se comprueban los permisos de una vista.

2.10. Ventajas y limitaciones de las vistas

Ventajas:

- Simplificación: permiten reutilizar consultas complejas mediante una sentencia sencilla.
- Reutilización: evitan repetir la misma consulta en diferentes aplicaciones o procesos.
- Seguridad: permiten mostrar únicamente determinadas columnas o filas cuando se combinan con permisos adecuados.
- Abstracción: separan la consulta utilizada por el usuario de los detalles de las tablas subyacentes.
- Mantenimiento: permiten centralizar la definición de consultas que se utilizan con frecuencia.

Limitaciones:

- Las vistas normales no almacenan una copia independiente de los resultados.
- No se pueden crear índices directamente sobre una vista en MySQL.
- No todas las vistas permiten insertar, actualizar o eliminar datos.
- Si se modifican o eliminan las tablas o columnas de las que depende una vista, esta puede dejar de funcionar correctamente.
- Una vista no garantiza por sí sola la seguridad: deben configurarse también los permisos correspondientes.
- Una vista compleja puede implicar una consulta costosa, dependiendo de las tablas implicadas, los índices y la forma en que MySQL procese la consulta.

3. GESTIÓN DE USUARIOS

La gestión de usuarios es una de las tareas fundamentales de la administración de un sistema gestor de bases de datos. No basta con crear bases de datos y tablas: también es necesario controlar quién puede acceder al servidor, qué operaciones puede realizar y sobre qué objetos puede ejecutarlas. En un entorno real, no todos los usuarios deben disponer de los mismos privilegios.

MySQL dispone de un sistema de cuentas de usuario y privilegios que permite establecer estas restricciones. En este apartado veremos cómo crear, consultar, modificar y eliminar usuarios, cómo gestionar sus contraseñas y cómo conceder o retirar permisos.

Antes de comenzar, es importante distinguir varios conceptos:

- Usuario: identidad que se utiliza para acceder a MySQL.
- Cuenta de usuario: en MySQL, una cuenta se identifica mediante un nombre de usuario y un equipo o dirección desde el que se permite la conexión.
- Autenticación: proceso mediante el cual MySQL comprueba la identidad de quien intenta conectarse, normalmente mediante una contraseña.
- Privilegio o permiso: autorización para realizar una operación concreta, como consultar datos, insertar registros, modificar la estructura de las tablas…
- Administración de privilegios: proceso de conceder, consultar o retirar los permisos asignados a una cuenta.

En MySQL, una cuenta se representa normalmente con el formato:

`'usuario'@'host'`

Donde usuario es el nombre de la cuenta, y host el equipo o dirección desde la que se permite conectar a MySQL.

Por ejemplo:

`'ana'@'localhost'.`

Esta cuenta permite identificar a la usuaria Ana cuando se conecta desde el equipo donde se ejecuta el servidor.

Otro ejemplo sería:

`'ana'@'192.168.1.50'`

En este caso, la cuenta de la usuaria Ana está asociada a las conexiones procedentes de esa IP. Ambas cuentas son distintas, pudiendo tener contraseñas y privilegios diferentes.

Es importante recordar que no debemos utilizar siempre el usuario `root`. Esta cuenta suele ser una cuenta administrativa con privilegios muy elevados, que puede utilizarse para tareas de administración, pero no es recomendable emplearla para todas las operaciones cotidianas.

Esto hace referencia al principio de mínimo privilegio: cada usuario debe disponer únicamente de los permisos necesarios para realizar las tareas que tiene asignadas. Aplicar este principio mejora la seguridad y recude el riesgo de modificaciones accidentales, accesos indebidos o daños en los datos.

3.1. Crear usuarios

Para crear una cuenta se utiliza la sentencia CREATE USER.

```
CREATE USER 'nombreUsuario'@'host' IDENTIFIED BY 'contraseña';
```

Donde:

- `nombreUsuario` será el nombre de la cuenta.
- `'host'` es el equipo o dirección desde la que se permite la conexión.

Para ejecutar esta operación, el usuario que realiza la creación debe disponer de los privilegios administrativos necesarios para la creación de usuarios.

Crear una cuenta no significa concederle permisos sobre las bases de datos. En MySQL, una cuenta recién creada no recibe automáticamente privilegios para consultar, insertar o modificar datos. Los permisos deben concederse expresamente. Esta separación entre creación de cuentas y concesión de privilegios permite configurar el acceso de forma controlada.

Si intentamos crear una cuenta que ya existe con la misma combinación de usuario y host, MySQL devolverá un error. Para evitarlo, podemos utilizar `IF NOT EXISTS`:

```
CREATE USER IF NOT EXISTS 'nombreUsuario'@'host' IDENTIFIED BY 'contraseña';
```

Si la cuenta ya existe, MySQL emitirá un aviso en lugar de devolver el error habitual. Esta cláusula no modifica la contraseña ni reinicializa los permisos de la cuenta existente.

3.2. Consultar usuarios existentes

Como administradores, necesitaremos comprobar qué cuentas existen y cuáles son sus características.

La información de las cuentas de MySQL se almacena en el esquema del sistema `mysql`. Podemos consultar los nombres y hosts de las cuentas mediante:

```
SELECT User, Host FROM mysql.user;
```

Esta consulta permite listar las cuentas registradas en el servidor. El acceso a esta información dependerá de los privilegios del usuario que ejecute la consulta.

Para consultar la cuenta con la que estamos conectados podemos utilizar:

```
SELECT USER();
```

```
SELECT CURRENT_USER();
```

Aunque sus nombres son parecidos, tienen finalidades distintas:

- `USER()` devuelve el nombre de usuario y el host que el cliente presentó al establecer la conexión.

- `CURRENT_USER()` devuelve la cuenta de MySQL que el servidor ha utilizado para identificar al usuario y comprobar sus privilegios.

En situaciones habituales pueden devolver valores iguales o similares, pero no siempre son idénticos. Por ejemplo, si una conexión se realiza mediante una cuenta anónima o intervienen reglas de identificación de cuentas, ambos resultados pueden diferir.

Para consultar la definición de una cuenta, se utiliza:

```
SHOW CREATE USER ‘nombreCuenta’@’host’;
```

Esta sentencia muestra información sobre la configuración de la cuenta, incluidos aspectos de autenticación y otras propiedades, según la versión de MySQL y los privilegios disponibles. Es útil para revisar cómo está configurado un usuario.

3.3. Eliminar usuarios

Cuando una cuenta deja de ser necesaria, por ejemplo, porque un empleado abandona la empresa, o una aplicación deja de utilizarse, puede ser conveniente eliminarla. Para ello se utiliza:

```
DROM USER ‘nombreUsuario’@’host’;
```

Esta sentencia elimina la cuenta indicada. Es importante diferenciar entre eliminar un usuario y eliminar sus datos. `DROP USER` no elimina las bases de datos ni las tablas que existan en el servidor. Tampoco elimina automáticamente todos los objetos de la base de datos que el usuario haya creado.

Para evitar errores si la cuenta no existe, podemos utilizar:

```
DROP USER IF EXISTS ‘nombreUsuario’@’host’;
```

Si la cuenta no existe, MySQL emitirá un aviso en lugar de producir el error habitual.

Antes de eliminar una cuenta, es recomendable comprobar que estamos indicando correctamente tanto el nombre del usuario como el host, para evitar eliminar una cuenta por error.

3.4. Modificar usuarios

La sentencia `ALTER USER` permite modificar las características de una cuenta existente. Entre otras posibilidades, permite cambiar la contraseña, bloquear o desbloquear una cuenta y configurar determinadas opciones de autenticación.

Para cambiar la contraseña de una cuenta, se utiliza:

```
ALTER USER 'nombreUsuario'@'host' IDENTIFIED BY 'nuevaContraseña';
```

Esta sentencia cambia la contraseña de la cuenta indicada.

La nueva contraseña debe cumplir los requisitos de seguridad que tenga configurados el servidor. MySQL aplica el mecanismo de autenticación correspondiente a la cuenta y gestiona el almacenamiento de las credenciales.

Para cambiar la contraseña del usuario actual, es decir, nuestra propia contraseña, podemos utilizar:

```
ALTER USER USER() IDENTIFIED BY ‘nuevaContraseña';
```

MySQL también permite cambiar el nombre de un usuario:

```
RENAME USER ‘nombreUsuario’@’host’ to ‘nuevoNombre’@’host’;
```

Además, MySQL permite bloquear temporalmente una cuenta sin eliminarla:

```
ALTER USER ‘nombreUsuario’@'host' ACCOUNT LOCK;
```

```
ALTER USER ‘nombreUsuario’@'host' ACCOUNT UNLOCK;
```

El bloqueo impide que la cuenta se utilice para iniciar nuevas conexiones mediante la autenticación normal. No elimina la cuenta ni sus privilegios.

Puede resultar útil cuando una persona deja temporalmente de necesitar acceso o cuando se quiere suspender una cuenta mientras se investiga un posible problema de seguridad.

`ALTER USER` permite configurar otras opciones como:

- La caducidad de las contraseñas.
- Las restricciones relativas a la reutilización de las contraseñas.
- Las condiciones de autenticación.
- El bloqueo temporal asociado a intentos fallidos de inicio de sesión, cuando está configurado.

Estas opciones dependen de la versión de MySQL y de la configuración del servidor.

4. PRIVILEGIOS DE MYSQL

Los privilegios determinan las operaciones que puede realizar cada usuario. MySQL dispone de numerosos privilegios, que pueden concederse en distintos niveles.

4.1. Conceder privilegios

Un usuario puede obtener privilegios para manipular objetos de una base de datos con el comando GRANT. Así mismo, se pueden denegar permisos con comando REVOKE.

La sintaxis del comando GRANT en MySQL es la siguiente:

```
GRANT tipoPrivilegio [(columnas)] [, tipoPrivilegio [(columnas)]] ...
    ON [tipoObjeto] TO usuarioORol [, usuarioORol] ...
```

```
[WITH GRANT OPTION]
```

`tipoPrivilegio` es la clase de permiso que se quiere otorgar. Típicamente serán select, insert, update... En la siguiente tabla se pueden consultar todos los tipos de privilegios en MySQL.

Privilegio	Significado
ALL [PRIVILEGES]	Da todos los permisos simples excepto GRANT OPTION.
ALTER	Permite el uso de ALTER TABLE.
ALTER ROUTINE	Modifica o borra rutinas almacenadas
CREATE	Permite el uso de CREATE TABLE.
CREATE ROUTINE	Permite crear rutinas almacenadas
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLES.
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER y REVOKE.
CREATE VIEW	Permite el uso de CREATE VIEW.
DELETE	Permite el uso de DELETE.
DROP	Permite el uso de DROP TABLE.
EXECUTE	Permite ejecutar rutinas almacenadas.
FILE	Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE.
INDEX	Permite el uso de CREATE INDEX y DROP INDEX.
INSERT	Permite el uso de INSERT.
LOOK TABLES	Permite el uso de LOOK TABLES en tablas para las que tenga el permiso SELECT.
PROCESS	Permite el uso SHOW FULL PROCESSLIST.
RELOAD	Permite el uso de FLUSH.
REPLICATION CLIENT	Permite al usuario preguntar dónde están los servidores maestro o esclavo.
REPLICATION SLAVE	Necesario para los esclavos de replicación
SELECT	Permite el uso de SELECT.
SHOW DATABASES	Permite el uso de SHOW DATABASES.
SHOW VIEW	Permite el uso de SHOW CREATE VIEW.
SHUTDOWN	Permite el uso de mysqladmin shutdown.
UPDATE	Permite el uso de UPDATE.
USAGE	Sinónimo de 'no privileges', permite únicamente la conexión con el gestor.
GRANT OPTION	Permite dar permisos.

Estos tipos de privilegios se pueden aplicar a las siguientes expresiones o tipos de objetos:

Expresión	Se aplica el permiso a...
nombre_tabla	La tabla con nombre nombre_tabla
*	Todas las tablas de la BBDD que se está usando.

.	Todas las tablas de todas las BBDD.
base_datos.*	Todas las tablas de la base de datos base_datos.
base_datos.nombre_tabla	Solo la tabla nombre_tabla de la base de datos base_datos.

Con TO usuario se indica el usuario o el rol al que se quiere otorgar el permiso.

Adicionalmente, se pueden indicar ciertas opciones precedidas de la cláusula WITH:

Expresión	Función
GRANT OPTION	Permite conceder a otros usuarios los permisos que tiene ese usuario, por tanto, el administrador debe ser muy cauto a la hora de conceder esta opción.
MAX_QUERIES_PER_HOUR count	Permite restringir el número de consultas por hora que puede realizar un usuario.
MAX_UPDATES_PER_HOUR count	Permite restringir el número de modificaciones por hora que puede realizar un usuario.
MAX_CONNECTIONS_PER_HOUR count	Permite restringir las conexiones (login) por hora que realiza un usuario.
MAX_USER_CONNECTIONS count	Permite limitar el número de conexiones simultáneas que puede tener un usuario.

En cualquiera de las opciones MAX_ si a count se le da valor 0, significa que es ilimitado.

La sentencia básica es la siguiente:

```
GRANT privilegios ON objeto TO 'usuario'@'host';
```

Supongamos que hemos creado el usuario consulta y queremos que pueda consultar los datos de la base de datos tienda.

```
GRANT SELECT ON tienda.* TO 'consulta'@'localhost';
```

El usuario podrá consultar las tablas y vistas de esa base de datos. Sin embargo, no podrá insertar, modificar o eliminar datos.

Podemos conceder varios privilegios en una única sentencia:

```
GRANT SELECT, INSERT, UPDATE ON tienda.* TO 'gestor'@'localhost';
```

Si queremos concede permisos únicamente sobre una tabla concreta:

```
GRANT SELECT ON tienda.Clientes TO 'consulta'@'localhost';
```

Si queremos ponerle un límite de consultas a una tabla concreta:

```
GRANT SELECT ON tienda.Clientes TO ‘consulta’@’localhost’ WITH MAX_QUERIES_PER_HOUR 10 MAX_UPDATES_PER_HOUR 10;
```

4.2. Retirar permisos

La sentencia REVOKE deniega permisos a un usuario sobre un objeto. Su sintaxis es:

```
REVOKE tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]]… ON (objeto) FROM usuario [, usuario]…
```

Una sintaxis más sencilla sería:

```
REVOKE tipo_privilegios ON objeto FROM 'usuario'@'host';
```

Por ejemplo, para retirar un permiso concreto:

```
REVOKE UPDATE ON tienda.* FROM 'gestor'@'localhost';
```

Esta sentencia retira el privilegio UPDATE en ese ámbito. Sin embargo, debemos tener en cuenta un detalle: si el usuario conserva el privilegio UPDATE por otra concesión más amplia, por ejemplo, a nivel global, retirar el permiso a nivel de base de datos no necesariamente elimina su capacidad de actualizar esos datos.

También se pueden retirar varios privilegios con una única sentencia:

```
REVOKE INSERT, DELETE ON tienda.* FROM 'gestor'@'localhost';
```

4.3. Comrpobar permisos

Tras crear una cuenta o modificar sus permisos, debemos comprobar que la configuración es correcta.

Para ello se utiliza:

```
SHOW GRANTS;
```

Esta sentencia muestra los privilegios y roles asociados a la cuenta que se está utilizando en la sesión.

También podemos consultar los permisos de otra cuenta:

```
SHOW GRANTS FOR 'cuenta'@'host';
```

El resultado mostrará los permisos en forma de sentencias GRANT.

4.4. Gestión de roles

Cuando hay muchos usuarios con funciones similares, administrar individualmente todos sus permisos puede resultar complicado.

MySQL permite utilizar roles, que son conjuntos de privilegios que pueden asignarse a diferentes cuentas.

Para crear un nuevo rol, se utiliza la siguiente sintaxis:

```
CREATE ROLE 'nombreRol';
```

Por ejemplo, podemos crear un rol y concederle permisos:

```
CREATE ROLE 'rol_consulta';

GRANT SELECT ON tienda.* TO 'rol_consulta';
```

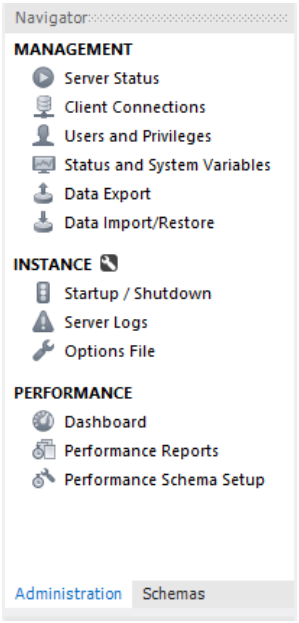
Después, podemos asignar ese rol a un usuario:

```
GRANT 'rol_consulta' TO 'consulta'@'localhost';
```

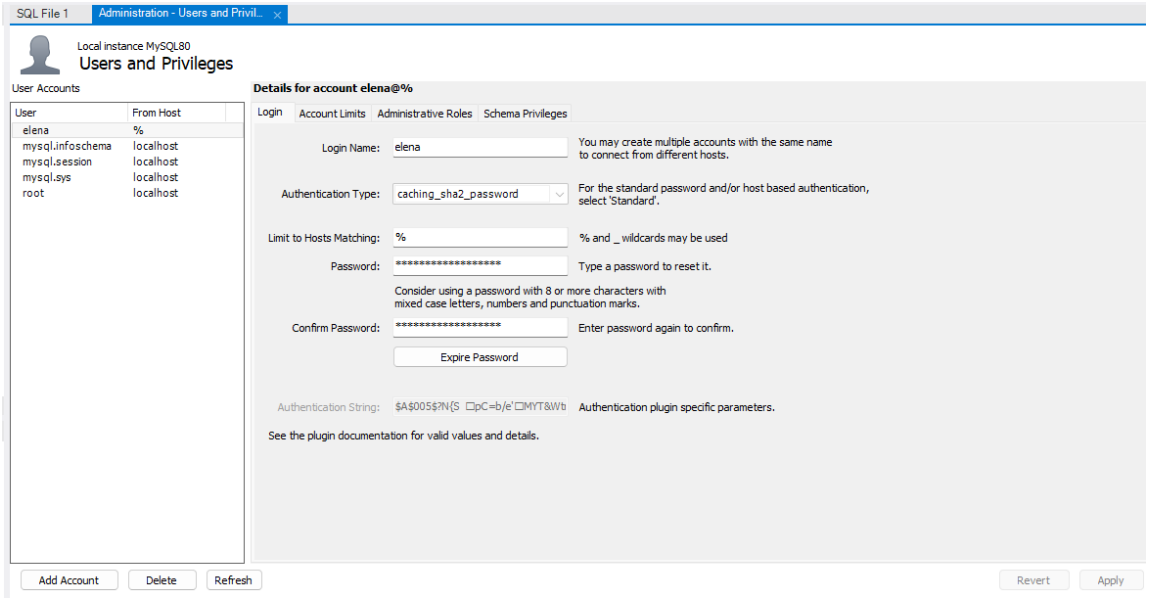
5. GESTIÓN DE USUARIOS MEDIANTE INTERFAZ GRÁFICA

La creación de usuarios y gestión de sus correspondientes permisos se puede realizar completamente desde la interfaz gráfica MySQL Workbench.

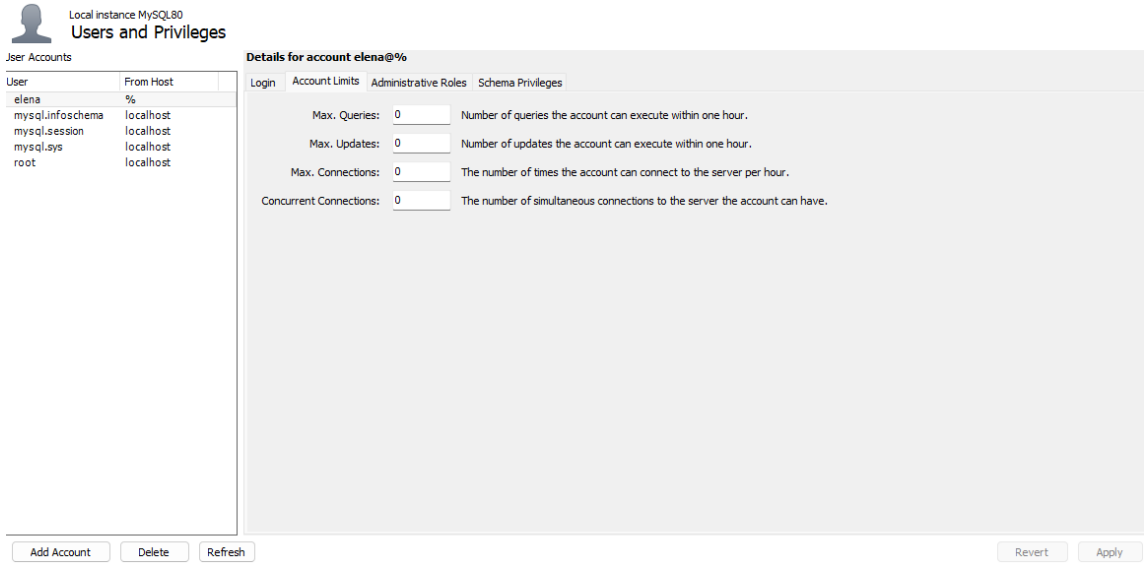
Para ello, deberemos acceder a una conexión con usuario root u otro usuario con sus mismos permisos. En el menú lateral izquierdo, deberemos acceder al apartado de "Administración".



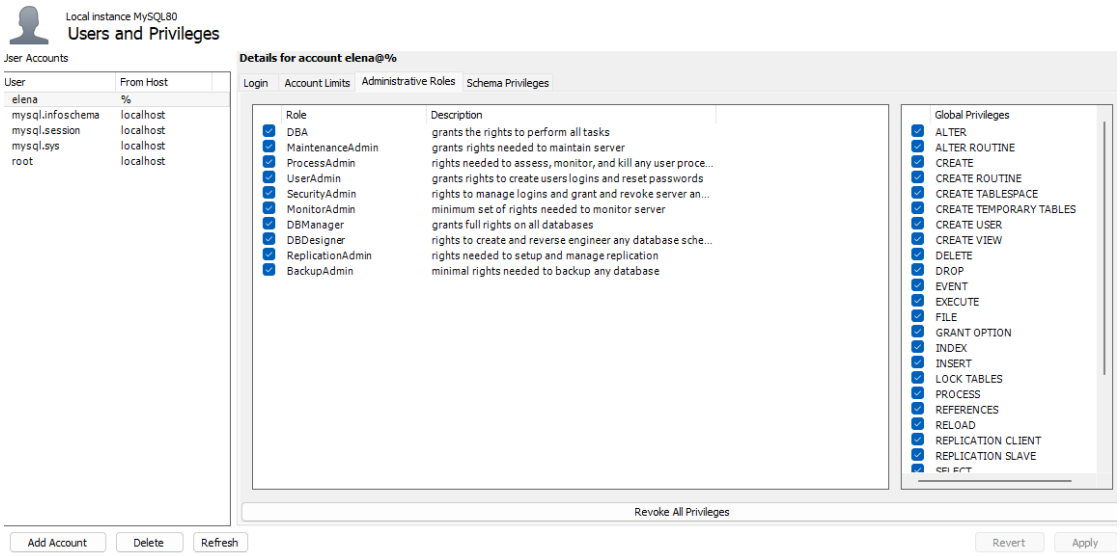
Seleccionando la opción usuarios y privilegios, se abre una ventana como la que se ve a continuación, que cuenta con diferentes pestañas, siendo la primera de ellas la referente a la creación de usuario y su correspondiente contraseña.



En la segunda pestaña podemos indicar los límites de la cuenta, es decir, la medida de seguridad con la que podemos establecer el número máximo de acciones que puede hacer el usuario en una hora.



Así mismo, podemos indicarle los diferentes privilegios que va a tener sobre el sistema.



6. BIBLIOGRAFÍA

Documentación MySQL oficial. Uso de vistas. Disponible en: https://dev.mysql.com/doc/refman/8.0/en/views.html

Documentación MySQL OFICIAL. Administración de cuentas. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/account-management-statements.html