UNIDAD 9 MECANISMOS DE SEGURIDAD

1 VISTAS

1.1 Introducción

Podemos definir una vista como una consulta almacenada en la base de datos que se utiliza como una tabla virtual.

Se trata de una perspectiva de la base de datos o ventana que permite a uno o varios usuarios ver solamente las filas y columnas necesarias para su trabajo.

Las vistas se emplean para:

- Realizar consultas complejas más fácilmente, ya que permiten dividir la consulta en varias partes
- Proporcionar tablas con datos completos
- Utilizar visiones especiales de los datos
- Ser utilizadas como tablas que resumen todos los datos
- Ser utilizadas como cursores de datos en los lenguajes procedimentales (como PL/SQL)

Entre las ventajas que ofrece la utilización de vistas cabe destacar:

- **Seguridad y confidencialidad**: ya que la vista ocultará los datos confidenciales o aquellos para los que el usuario no tenga permiso.
- **Comodidad**: ya que solamente muestra los datos relevantes, permitiendo, incluso trabajar con agrupaciones de filas como si se tratase de una única fila o con composiciones de varias tablas como si se tratase de una única tabla.
- **Independencia respecto a posibles cambios** en los nombres de las columnas, de las tablas, etcétera.

1.2 Creación de vistas

Las vistas tienen la misma estructura que una tabla: filas y columnas, y se tratan de forma semejante a una tabla.

Para crear una vista se utiliza el comando `CREATE VIEW` según el siguiente formato genérico:

```
CREATE [OR REPLACE] VIEW nombredevista [( listadecolumnas)] AS consulta;
```

Donde:

- ***nombredevista*** es el nombre que tendrá la vista que se va a crear.
- ***listadecolumnas*** es opcional. Permite especificar un nombre para cada columna de la vista. Si no se especifica, cada columna quedará con el nombre asignado por la consulta.
- ***consulta*** es la SELECT que define la vista.

Ejemplo: Crear la vista emple_dep30 para la gestión de los empleados del departamento 30.

```
CREATE VIEW emple_dep30 AS
SELECT * FROM EMPLEADOS WHERE DEP_NO = 30
```

Ejemplo: Crear la vista datos_emple que contiene información de todos los empleados ocultando la información confidencial.

```
CREATE VIEW datos_emple AS
SELECT emp_no, apellido, oficio, director, fecha_alta, dep_no
FROM empleados
```

Una vez creada la vista se puede utilizar como si se tratase de una tabla (observando algunas restricciones que se verán más adelante).

Por ejemplo si escribimos:

```
SELECT * FROM emple_dep30
```

O bien:

```
SELECT * FROM datos_emple
```

En ambos casos obtendremos el mismo resultado que si introducimos la cláusula de selección en la que se basa la vista, mostrado en el apartado anterior.

También aplicar condiciones de selección, de filas y de columnas al recuperar datos de la vista:

```
SELECT apellido FROM emple_dep30
SELECT apellido, director FROM datos_emple
WHERE oficio = 'VENDEDOR'
```

Las vistas no ocupan espacio en la base de datos ya que lo único que se almacena es la definición de la vista. El gestor de la base de datos se encargará de comprobar los comandos SQL que hagan referencia a la vista, transformándolos en los comandos correspondientes referidos a las tablas originales, todo ello de forma transparente para el usuario.

1.3 Tipos de vista

Atendiendo al tipo de consulta en la que se basa la vista podemos distinguir los siguientes tipos:

Vistas con filtro de filas y de columnas.

Se crean basándose en **consultas que filtran determinadas filas o columnas**. Las vistas *emple_dep30* y *datos_emple* son ejemplos de este tipo de vistas.

También se pueden crear vistas que establezcan filtros de selección tanto a nivel de fila como a nivel de columna.

Ejemplo: Crear la vista *datos_vendedores* que muestra solamente las columnas *emp_no, apellido, director, fecha_alta, dep_no,* de aquellos empleados cuyo oficio es *VENDEDOR*.

```
CREATE VIEW datos_vendedores
(numvendedor, apellido, director, fecha_alta, dep_no) AS
SELECT emp_no, apellido, director, fecha_alta, dep_no
FROM empleados
WHERE oficio = 'VENDEDOR'
```

Agrupaciones

También se pueden crear vistas a partir de consultas que incluyen agrupaciones, como en el siguiente ejemplo:

```
CREATE VIEW resumen_dep
(dep_no, num_empleados, suma_salario, suma_comision) AS
SELECT dep_no, COUNT(emp_no), SUM(salario), SUM(comision)
FROM empleados
GROUP BY dep_no
```

En estos casos, **cada fila de la vista corresponderá a varias filas en la tabla original.**

Normalmente la mayoría de las columnas de este tipo de vistas corresponden a funciones de columna tales como *SUM, AVG, MAX, MIN*, etcétera. Por ello el estándar SQL establece en estos casos la obligatoriedad de especificar la lista de columnas.

Aunque algunos gestores de bases de datos permiten saltar esta restricción. No es aconsejable ya que las columnas correspondientes de la vista quedarán con nombres como *COUNT(EMP_NO), SUM(SALARIO), SUM(COMISION)* lo cual no resulta operativo para su posterior utilización.

Composiciones.

Una vista se puede crear a partir de una consulta que recupera información de varias tablas relacionadas.

La siguiente vista incluye información de las tablas *empleados (emp_no, apellido y oficio)* y *departamentos (dnombre, localidad)* relacionadas a partir de la columna común *dep_no*.

```
CREATE VIEW datos_emp_dep AS
SELECT emp_no, apellido, oficio, dnombre, localidad
FROM empleados, departamentos
WHERE empleados.dep_no = departamentos.dep_no
```

Así mismo, se pueden crear vistas que incluyan todas o varias de las posibilidades estudiadas. Por ejemplo la siguiente vista permite trabajar con datos de dos tablas, agrupados y seleccionando las filas que interesan (en este caso todos los departamentos que tengan mas de un empleado):

```
CREATE VIEW resumen_emp_dep
(departamento, localidad, num_empleados, suma_salario)
AS
SELECT dnombre, localidad, COUNT(emp_no), SUM(salario)
FROM empleados, departamentos
WHERE empleados.dep_no = departamentos.dep_no
GROUP BY dnombre, localidad
HAVING COUNT(emp_no) > 1
```

Vistas sobre vistas.

La consulta que define a una vista puede incluir a su vez una vista.

Por ejemplo, la siguiente vista *dat_emp_dep30* se crea a partir de la vista *emple_dep30* filtrando algunas de las columnas:

```
CREATE VIEW dat_emp_dep30 AS
SELECT emp_no, apellido, oficio, director,
fecha_alta, dep_no
FROM emple_dep30;
```

1.4 Restricciones para la creación y uso de vistas

Tenemos las siguientes restricciones

- Es obligatorio especificar la lista de nombres de columnas cuando la consulta devuelve funciones de agrupamiento como SUM, COUNT, etcétera.
- No se pueden utilizar funciones de agrupación sobre columnas de vistas que se basan a su vez en funciones de agrupación ya que en la práctica supondría un doble agrupamiento que no está permitido por el estándar.

1.5 Actualizaciones en vistas

Como ya hemos explicado, las vistas son tablas virtuales, y los datos que manejan, en realidad, pertenecen a otras tablas. Por tanto, **cualquier comando de actualización (INSERT, UPDATE o DELETE) sobre una vista será traducido por el SGBD a una actualización de la tabla sobre la que se basa**.

Por ejemplo, el siguiente comando introducirá, a través de la vista *emple_dep30,* una nueva fila en la tabla *departamentos*:

```
INSERT INTO emple_dep30
VALUES (8998, 'CORTES', 'VENDEDOR', 7698, '20/02/99', 180000, NULL, 30);
```

Para que se puedan utilizar comandos de actualización sobre una vista, deberá existir una correspondencia inequívoca en filas y columnas entre la vista y la tabla sobre la que basa.

La regla anterior se puede desglosar en:

- **Las vistas basadas en agrupaciones no pueden ser actualizadas** ya que no existe una correspondencia directa entre las filas de la vista y las filas de las tablas originales.
- Por la misma razón **no se pueden actualizar las vistas creadas a partir de composiciones con varias tablas**.
- **Tampoco se pueden actualizar vistas cuyas columnas se han creado a partir de expresiones o funciones** que pueden enmascarar el valor de la columna en la tabla original.

Nota: Algunos gestores de base de datos permiten saltar estas restricciones en ciertas circunstancias. Estas características suelen estar documentadas en los manuales del producto.

1.6 Vistas con validación

Cuando una vista se crea utilizando una consulta que incluye una condición de selección (cláusula WHERE) e intentamos insertar o modificar una fila que no cumple la condición, ¿permitirá el SGBD la inserción/modificación?.

Supongamos que queremos insertar en la vista emp_dep30 un empleado cuyo departamento será el 20:

```
INSERT INTO emple_dep30
VALUES (8999, 'LUCAS', 'ANALISTA', 7566, '20/02/99', 380000, NULL, 20);
```

Después de esta orden el sistema devolverá el mensaje:

```
Una fila creada.
```

Sin embargo, al intentar recuperar la información insertada nos encontraremos lo siguiente:

```
SELECT * FROM emple_dep30 WHERE emp_no = 8999;
Ninguna fila seleccionada.
```

En realidad la fila se ha insertado en la tabla *empleados* a través de la vista *emple_dep30* (podemos comprobarlo mediante una consulta a la tabla empleados). Pero la vista no puede acceder a la información que se acaba de introducir ya que no cumple la condición especificada al crear la vista.

Este comportamiento puede parecer incoherente y resulta desconcertante para el usuario.

Se puede evitar indicando al crear la tabla que se compruebe que cualquier inserción o modificación satisface la condición establecida añadiendo la cláusula **WHITH CHECK OPTION** al final de la instrucción de creación.

El formato de creación de vistas ampliado con esta opción será:

```
CREATE VIEW nombredevista [( listadecolumnas)]
AS consulta [WITH CHECK OPTION];
```

En nuestro ejemplo:

```
CREATE VIEW emple_dep30 AS
SELECT * FROM EMPLEADOS
WHERE DEP_NO = 30
WITH CHECK OPTION
```

Si la fila insertada o modificada no satisface la condición de creación de la vista, la sentencia INSERT o UPDATE falla, y no se realiza la operación.

También podemos usar la sentencia **WITH READ ONLY**, que especifica que sólo se puede hacer SELECT de las filas de la vista creada.

```
CREATE VIEW nombredevista [( listadecolumnas)]
AS consulta [WITH CHECK OPTION | READ ONLY];
```

En nuestro ejemplo:

```
CREATE VIEW emple_dep30 AS
SELECT * FROM EMPLEADOS
```

```
WHERE DEP_NO = 30
WITH READ ONLY
```

Cualquier operación INSERT, UPDATE o DELETE sobre la vista dep30 fallará. Esto no lo soporta MySQL.

1.7 Mostrar la lista de vistas

La vista del diccionario de datos de Oracle USER_VIEWS permite mostrar una lista de todas las vistas que posee el usuario actual. Es decir, para saber qué vistas hay disponibles se usa:

```
SELECT * FROM USER_VIEWS;
```

La columna **TEXT** de esa vista contiene la sentencia SQL que se utilizó para crear la vista (sentencia que es ejecutada cada vez que se invoca a la vista).

En MySQL se pueden consultar con el comando SHOW TABLES;

1.8 Eliminación de vistas

La sentencia **DROP VIEW** permite eliminar la definición de una vista.

```
DROP VIEW nombredevista
```

El siguiente ejemplo borrará la vista emple_dep30:

```
drop view emple_dep30
```

2 SINÓNIMOS

Un sinónimo es un nuevo nombre que se puede dar a una tabla o vista. Con los sinónimos se pueden utilizar dos nombres diferentes para referirse a un mismo objeto. Resultan interesantes cuando se tiene acceso a tablas de otros usuarios, se pueden crear sinónimos para hacer referencia a a esas tablas y, así, no hay que escribir el nombre del usuario propietario de la tabla delante de la tabla a la que tenemos acceso cada vez que queramos consultarla. En MySQL no se pueden crear sinónimos.

2.1 Creación de sinónimos

La sintaxis es:

```
CREATE [PUBLIC] SYNONYM nombre FOR objeto;
```

Dónde:

- objeto es el objeto al que se referirá el sinónimo
- PUBLIC, hace que el sinónimo esté disponible para cualquier usuario.

2.2 Lista de sinónimos

En Oracle, la vista **USER_SYNONYMS** permite observar la lista de sinónimos del usuario, la vista **ALL_SYNONYMS** permite mostrar la lista completa de sinónimos de todas las bases de datos a las que tenemos acceso.

2.3 Borrar sinónimos

La sintaxis es:

```
DROP SYNONYM nombre;
```

3 ÍNDICES

Los índices son objetos que forman parte del esquema que hacen que las bases de datos aceleren las operaciones de consulta y ordenación sobre los campos a los que el índice hace referencia. El propósito principal es acelerar las consultas

Se almacenan aparte de la tabla a la que hace referencia, lo que permite crearlos y borrarlos en cualquier momento. Por ejemplo: Se supone que Hacienda dispone de una tabla de CONTRIBUYENTES en la que cada fila se identifica por el NIF del contribuyente. Para consultar un NIF determinado realizamos la siguiente consulta: `SELECT * FROM CONTRIBUYENTES WHERE NIF='7866978-A';`

Si la columna NIF no está indexada, SQL recorre la tabla de CONTRIBUYENTES secuencialmente, desde la primera fila hasta el final de la tabla, tanto si encuentra como si no encuentra el dato que se busca. Si la columna NIF está indexada, SQL realizará una búsqueda binaria en el índice hasta encontrar el dato buscado, mediante un único acceso sobre la tabla CONTRIBUYENTES.

Lo que realizan es una lista ordenada por la que SQL puede acceder para facilitar la búsqueda de los datos. Cada vez que se añade un nuevo registro, los índices involucrados se actualizan a fin de que su información esté al día. De ahí que cuantos más índices haya, más le cuesta a SQL añadir registros, pero más rápidas se realizan las instrucciones de consulta.

La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY. Estos son índices obligatorios, por lo que los crea el propio SGBD.

En MySQL podemos encontrar estos tipos de índices:

- **Índice B-tree (o simplemente INDEX):** Es el tipo de índice más común y predeterminado en MySQL (para motores de almacenamiento como InnoDB y MyISAM). Son eficientes para:
  - Búsquedas por igualdad (=).
  - Búsquedas por rango (>, <, >=, <=, BETWEEN).
  - Búsquedas con el prefijo de cadenas (LIKE 'prefijo%').
  - Ordenación (ORDER BY) y agrupación (GROUP BY) si el índice cubre las columnas necesarias.
- **Índice Hash (HASH):** Utiliza una función hash para crear un índice. Son extremadamente rápidos para búsquedas por igualdad (=) pero no son eficientes para búsquedas por rango o prefijo. Solo están disponibles para algunos motores de almacenamiento, como MEMORY.
- **Índice FULLTEXT (FULLTEXT):** Diseñado para realizar búsquedas de texto completo en columnas de tipo TEXT o VARCHAR. Permiten buscar palabras o frases dentro del texto.
- **Índices Espaciales (SPATIAL):** Utilizados para indexar datos espaciales (tipos de datos GEOMETRY, POINT, LINESTRING, POLYGON, etc.) y optimizar consultas espaciales. Solo están disponibles para motores de almacenamiento que soportan tipos de datos espaciales (como MyISAM y InnoDB).
- **Índices de Prefijo:** Se pueden crear índices que solo incluyan los primeros caracteres de una columna VARCHAR o TEXT. Esto puede reducir el tamaño del índice pero también puede disminuir su selectividad.

3.1 Creación de índices

La sintaxis es:

```
CREATE INDEX nombre ON tabla (columna 1, [columna 2 …])
```

En MySQL la sintaxis completa la podéis encontrar en https://dev.mysql.com/doc/refman/8.4/en/create-index.html

Ejemplo:

```
CREATE INDEX nombre_completo ON clientes (apellido1, apellido2, nombre)
```

El ejemplo crea un índice para los campos apellido1, apellido2 y nombre. Esto no es lo mismo que crear un índice para cada campo, este índice es efectivo cuando se buscan u ordenan clientes usando los tres campos (apellido1, apelllido2 y nombre) a la vez.

Se aconseja crear índices en campos que:

- Contengan una gran cantidad de valores.
- Contengan una gran cantidad de nulos.
- Sean parte habitual de cláusulas WHERE, GROUP BY u ORDER BY.

No se aconseja crear índices en campos que:

- Pertenezcan a tablas pequeñas
- No se usen a menudo en las consultas
- Pertenecen a tablas que se actualizan frecuentemente
- Se utilizan en expresiones

3.2 Lista de índices

En Oracle

Para ver la lista de índices en Oracle se utiliza la vista USER_INDEXES. Mientras que la vista USER_IND_COLUMNS Muestra la lista de columnas que son utilizadas por índices.

En MySQL

Para ver los índices de una tabla se muestran con:

```
SHOW INDEXES FROM nombre_tabla;
```

3.3 Borrar índices

La instrucción `DROP INDEX` seguida del nombre del índice permite eliminar el índice en cuestión.

4 SECUENCIAS

Una secuencia sirve para generar automáticamente números distintos. Se utilizan para generar valores para campos que se utilizan como clave forzada +- subrogada o sustituta - (claves cuyo valor no interesa, solo sirven para identificar los registros de una tabla) Es decir, se utilizan en los identificadores de las tablas, siempre y cuando no importe qué número se asigna a cada fila.

Es una rutina interna de la base de datos la que realiza la función de generar un número distinto cada vez. Las secuencias se almacenan independientemente de la tabla, por lo que una misma secuencia se puede utilizar para diversas tablas.

MySQL no tiene secuencias. Como tal, podemos usar AUTO_INCREMENT.

4.1 Creación de secuencias

Una secuencia es un objeto de base de datos que sirve para generar números enteros únicos. Es muy útil para generar automáticamente valores para claves primarias. Para crear una secuencia en el esquema propio es necesario tener el privilegio CREATE SEQUENCE. Se crea una secuencia en cualquier otro esquema con el privilegio CREATE ANY SEQUENCE.

El formato para crear una secuencia es éste:

```
CREATE SEQUENCE secuencia
[INCREMENT BY n]
  [START WITH n]
[{MAXVALUE n|NOMAXVALUE}]
[{MINVALUE n|NOMINVALUE}]
[{CYCLE|NOCYCLE}]
```

Donde:

- secuencia. Es el nombre que se le da al objeto de secuencia
- INCREMENT BY. Indica cuánto se incrementa la secuencia cada vez que se usa. Por defecto se incrementa de uno en uno
- START WITH. Indica el valor inicial de la secuencia (por defecto 1)
- MAXVALUE. Máximo valor que puede tomar la secuencia. Este número entero debe ser mayor o igual que el entero especificado en MINVALUE.
- NOMAXVALUE señala que el valor máximo para una secuencia ascendente es 1027 y para una secuencia descendente –1
- MINVALUE. Mínimo valor que puede tomar la secuencia. Por defecto -1026
- CYCLE. Hace que la secuencia vuelva a empezar si se ha llegado al máximo valor.

Ejemplo:

```
CREATE SEQUENCE numeroPlanta
INCREMENT BY 100
```

```
START WITH 100
MAXVALUE 2000;
```

4.2 Ver lista de secuencias

La vista del diccionario de datos de Oracle USER_SEQUENCES muestra la lista de secuencias actuales. La columna LAST_NUMBER muestra cual será el siguiente número de secuencia disponible.

4.3 Uso de las secuencias

Los métodos NEXTVAL y CURRVAL se utilizan para obtener el siguiente número y el valor actual de la secuencia respectivamente.

Ejemplo de uso (Oracle):

```
SELECT numeroPlanta.NEXTVAL FROM DUAL;
```

En SQL estándar:

```
SELECT nextval('numeroPlanta');
```

Eso muestra en pantalla el siguiente valor de la secuencia. Realmente NEXTVAL incrementa la secuencia y devuelve el valor actual. CURRVAL devuelve el valor de la secuencia, pero sin incrementar la misma.

Ambas funciones pueden ser utilizadas en:

- Una consulta SELECT que no lleve DISTINCT, ni grupos, ni sea parte de una vista, ni sea subconsulta de otro SELECT, UPDATE o DELETE
- Una subconsulta SELECT en una instrucción INSERT
- La cláusula VALUES de la instrucción INSERT
- La cláusula SET de la instrucción UPDATE

No se puede utilizar (y siempre hay tentaciones para ello) como valor para la cláusula DEFAULT de un campo de tabla.

Su uso más habitual es como apoyo al comando INSERT (en Oracle):

```
INSERT INTO plantas(num, uso)
VALUES(numeroPlanta.NEXTVAL, 'Suites');
```

4.4 Modificar secuencias

Se pueden modificar las secuencias, pero la modificación sólo puede afectar a los futuros valores de la secuencia, no a los ya utilizados. Sintaxis:

```
ALTER SEQUENCE secuencia
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n|NOMAXVALUE}]
[{MINVALUE n|NOMINVALUE}]
[{CYCLE|NOCYCLE}]
```

4.5 Borrar secuencias

Lo hace el comando DROP SEQUENCE seguido del nombre de la secuencia a borrar.

4.6 Listar secuencias

En SQL estándar, a través de INFORMATION_SCHEMA.SEQUENCES podemos acceder a la información sobre todas

las secuencias creadas.
En Oracle se hace mediante la vista USER_SEQUENCES, permite observar la lista de secuencias del usuario

5 OPTIMIZACIÓN DE CONSULTAS, CONSULTA DEL PLAN DE EJECUCIÓN

Comando EXPLAIN

El comando `EXPLAIN` en MySQL es una herramienta muy poderosa que te permite analizar cómo MySQL ejecuta una consulta SQL, lo cual es especialmente útil para optimizar el rendimiento de las consultas, identificando cuellos de botella o pasos innecesarios en la ejecución.

Cuando utilizas `EXPLAIN` antes de una consulta, MySQL devuelve información sobre cómo se llevará a cabo la ejecución de la consulta. Esta información incluye detalles como el orden de las tablas que se van a leer, el tipo de unión que se realizará entre las tablas, si se está utilizando un índice, entre otros aspectos.

Sintaxis básica de `EXPLAIN`

```
EXPLAIN SELECT * FROM nombre_de_tabla WHERE condiciones;
```

¿Qué devuelve el comando EXPLAIN?

El comando `EXPLAIN` muestra una tabla con varias columnas, las cuales proporcionan información detallada sobre cómo MySQL ejecuta la consulta. Las principales columnas son:

1. **id**: Identificador de la consulta (útil en consultas complejas con subconsultas).
2. **select_type**: Indica el tipo de la consulta. Algunos valores posibles son:
    - **SIMPLE**: Una consulta sin subconsultas ni uniones.
    - **PRIMARY**: La consulta principal.
    - **UNION**: Una consulta que se une con otra consulta.
    - **SUBQUERY**: Una subconsulta en la cláusula WHERE, FROM o SELECT.
3. **table**: Muestra la tabla que está siendo referenciada en ese paso de la ejecución.
4. **type**: Tipo de acceso a la tabla, uno de los más importantes para identificar cuellos de botella:
    - **ALL**: Escaneo completo de la tabla (es el menos eficiente).
    - **index**: Escaneo por índice, pero lee toda la tabla (más eficiente que ALL).
    - **range**: Escaneo de un rango de índices.
    - **ref**: Busca una fila usando un valor de índice único.
    - **eq_ref**: Se utiliza cuando una fila se puede encontrar en un índice único con una búsqueda exacta.
    - **const**: Se utiliza cuando la consulta busca una constante (es muy eficiente).
    - **NULL**: No se refiere a una tabla en ese paso (puede ocurrir en una subconsulta).
5. **possible_keys**: Los índices que MySQL podría usar para resolver la consulta.
6. **key**: El índice que MySQL realmente utilizará.
7. **key_len**: La longitud del índice que se utilizará.

8. **ref**: Muestra qué columnas o constantes se utilizan para buscar en el índice.
9. **rows**: El número estimado de filas que MySQL cree que tiene que leer para ejecutar la consulta.
10. **Extra**: Información adicional sobre cómo se ejecutará la consulta, como si se está usando un "filesort", si se está realizando un "join" de tablas, si se está utilizando un índice de cobertura (covering index), entre otros.

Ejemplo de resultado de EXPLAIN:

Supón que tienes una consulta como la siguiente:

```
EXPLAIN SELECT * FROM empleados WHERE salario > 50000;
```

Y el resultado podría ser algo como esto:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	empleados	range	salario_idx	salario_idx	4	NULL	1000	Using where; Using index

Análisis del resultado:

- **id**: Es una consulta simple, por lo que tiene el valor 1.
- **select_type**: La consulta es de tipo SIMPLE, ya que no tiene subconsultas.
- **table**: La tabla que se está usando es empleados.
- **type**: Se está utilizando el tipo range, lo que significa que MySQL está utilizando un índice para realizar una búsqueda en un rango de valores.
- **possible_keys**: El índice que podría usarse es salario_idx, que es el índice sobre la columna salario.
- **key**: El índice que MySQL está utilizando realmente es salario_idx.
- **key_len**: La longitud del índice es 4, lo que indica que MySQL está utilizando el índice completo.
- **ref**: No se está haciendo referencia a ninguna columna en el índice.
- **rows**: Se estima que MySQL leerá unas 1000 filas para ejecutar la consulta.
- **Extra**: El Extra muestra que se está utilizando el filtro WHERE y el índice.

¿Cómo optimizar una consulta con EXPLAIN?

La información que proporciona EXPLAIN puede ayudarte a identificar problemas en el rendimiento de tus consultas. Algunos consejos para optimizar las consultas son:

1. **Evitar** ALL **en la columna** type: Si ves que MySQL está usando un escaneo completo de la tabla (ALL), es un indicador de que no se está utilizando un índice eficiente. Asegúrate de que las columnas de búsqueda estén indexadas.
2. **Usar índices adecuados**: Asegúrate de que las columnas que aparecen en las cláusulas WHERE, JOIN y ORDER BY estén indexadas adecuadamente.
3. **Revisar la columna** Extra: Si ves términos como "Using filesort" o "Using temporary", significa que MySQL está realizando operaciones adicionales que podrían ralentizar la consulta. Esto puede deberse a la falta de un índice adecuado o a una consulta compleja.
4. **Analizar las subconsultas**: Si tienes subconsultas en tu consulta principal, asegúrate de que MySQL

las esté manejando eficientemente, ya que las subconsultas pueden ser costosas.

Comando ANALYZE

ANALYZE **ejecuta la consulta** y proporciona información sobre el **rendimiento real** de cada etapa del plan de ejecución. A diferencia de EXPLAIN, ANALYZE sí interactúa con los datos. La salida de ANALYZE incluye información adicional sobre el tiempo de ejecución y el costo de cada etapa.

- **rows:** Número real de filas examinadas.
- **filtered:** Porcentaje real de filas filtradas.
- **Duration:** El tiempo que tardó en ejecutarse este paso.
- **Cost:** Una medida del costo estimado de este paso.

SQL

```
ANALYZE SELECT * FROM clientes WHERE nombre = 'Juan';

ANALYZE SELECT c.nombre, p.precio FROM clientes c JOIN pedidos p ON c.id =
p.cliente_id WHERE p.fecha > '2024-01-01';
```

6 GESTIÓN DE SEGURIDAD EN ORACLE

La gestión de la seguridad tiene que ver con la gestión de los usuarios y con la concesión y supresión de privilegios a los usuarios. El administrador de la base de datos es el encargado de permitir o denegar el acceso a determinados objetos o recursos de la base de datos.

Tenemos dos tipos de seguridad en una base de datos:

- **Seguridad del sistema:** incluye los mecanismos que controlan el acceso y uso de la base de datos: quién se conecta.
- **Seguridad de los datos:** incluye mecanismos que controlan el acceso y uso de la base de datos a nivel de objetos: quién puede acceder a una tabla, quién puede insertar datos, etc.

USUARIOS

Un usuario es un nombre definido en nuestro sistema que se puede conectar y acceder a diferentes objetos de las bases de datos que tenemos en el sistema.

Asociado con cada usuario de la base de datos hay un esquema con el mismo nombre. Un esquema es una colección lógica de objetos (tablas, vistas, secuencias, sinónimos,...). Por defecto, cada usuario tiene acceso a todos los objetos de su esquema correspondiente, y puede acceder a los objetos de otro usuario siempre y cuando tenga los privilegios necesarios.

Creación de Usuarios

Al instalar el sistema gestor de base de datos ORACLE se crean automáticamente dos usuarios con privilegios de administrador de la base de datos (DBA): SYS y SYSTEM.

Para crear usuario se necesita el privilegio CREATE USER.

La orden para crear usuarios es:

```
CREATE USER nombre_usuario
IDENTIFIED BY contraseña
[DEFAULT TABLESPACE espacio_tabla]
[TEMPORARY TABLESPACE espacio_tabla]
[QUOTA {entero {K|M} | UNLIMITED} ON espacio_tabla
[PROFILE perfil];
```

Donde:

- CREATE USER nombre_usuario, crea un nombre de usuario que será identificado por el sistema.
- IDENTIFIED BY permite dar una clave de acceso.
- DEFAULT TABLESPACE, asigna a un usuario una tablespace por defecto para almacenar los objetos que cree. Si no se asigna ninguno, el tablespace por defecto es USERR.
- TEMPORARY TABLESPACE específica el nombre de tablespace para trabajos temporales. Si no se especifica ninguno, el tablespace por defecto es TEMP.
- QUOTA: asigna un espacio en Mega o Kilobytes en el tablespace asignado. Si no se especifica, el usuario no tiene cuota asignada y no podrá crear objetos en el tablespace.
- PROFILE: asigna un perfil al usuario.

Ejemplo1: Crear un usuario de nombre USUARIO1. La clave es la misma. El tablespace por defecto es USERS (ya que no se indica) al cual se le han asignado 500 kilobytes. El tablespace para trabajos temporales es TEMP (ya que no se indica en la orden):

```
CREATE USER USUARIO1 IDENTIFIED BY USUARIO1 QUOTA 500K ON USERS;
```

Ejemplo2: Crear un usuario de nombre USUARIO2. La clave es la misma. El tablespace por omisión es TRABAJO al cual se le han asignado 1 megabyte. El tablespace para trabajos temporales es TEMPORAL al cual se han asignado 500 kilobytes.

```
CREATE USER USUARIO2 IDENTIFIED BY USUARIO2 DEFAULT TABLESPACE TRABAJO
TEMPORARY TABLESPACE TEMPORAL QUOTA 1M ON TRABAJOS QUOTA 500K ON TEMPORAL;
```

Aquí las tablas TRABAJOS y TEMPORAL deberían haber sido creadas previamente.

Nota: Vistas con información de usuarios

- **USER_USERS:** Obtiene información del usuario actual: fecha de creación, los tablespace asignados, el identificador, etc.
- **ALL_USERS**: Obtiene información acerca de todos los usuarios creados en la base de datos: el nombre, la fecha de creación y su identificador.

Modificación de usuarios

Las opciones dadas al usuario con CREATE USER pueden ser modificadas con la orden ALTER USER. La sintaxis es la siguiente:

```
ALTER USER nombre_usuario
IDENTIFIED BY contraseña
[DEFAULT TABLESPACE espacio_tabla]
[TEMPORARY TABLESPACE espacio_tabla]
[QUOTA {entero {K|M} | UNLIMITED} ON espacio_tabla
[PROFILE perfil];
```

Cada usuario puede cambiar su clave de acceso. No puede cambiar el tablespace por defecto, la cuota ni el perfil, a no ser que se tenga el privilegio ALTER USER.

Borrado de usuarios

Podemos borrar un usuario de la base de datos, incluido los objetos que contiene. La sintaxis es:

```
DROP USER usuario [CASCADE];
```

La opción CASCADE suprime todos los objetos del usuario antes de borrar el usuario.

Para borrar un usuario es necesario tener el privilegio DROP USER.

Por ejemplo, borramos el usuario1: `DROP USER USUARIO1`. Si el usuario tiene objetos creados se visualizará un mensaje de error: ORA-01922.

Se debe especificar CASCADE para borrar USUARIO1 `DROP USER USUARIO1 CASCADE`;

6.1 PRIVILEGIOS

Un privilegio es la capacidad de un usuario dentro de la BD a realizar determinadas operaciones o acceder a determinados objetos de otros usuarios.

Cuando se crea un usuario es necesario darle privilegios para que pueda hacer algo. Oracle ofrece varios roles o funciones. Un rol o función está formado por un conjunto de privilegios.

Roles que se aplican al entorno de desarrollo.

Roles (funciones)	Privilegios
CONNECT	ALTER_SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE Y CREATE VIEW.
RESOURCE	CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE TABLE, CREATE SEQUENCE, CREATE TRIGGER Y CREATE TYPE.
DBA	Posee todos los privilegios del sistema.

Privilegio sobre los objetos

Estos privilegios nos permiten acceder y realizar cambios en los datos de los objetos de otros usuarios. Se dispone de los siguientes privilegios sobre los objetos tablas, vistas, secuencias y procedures:

Privilegio sobre objetos	Tabla	Vista	Secuencia	Procedure
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X			
INSERT	X	X		
REFERENCES	X			
SELECT	X	X	X	
UPDATE	X	X		

La orden para dar privilegios sobre los objetos es GRANT, con la siguiente sintaxis:

```
GRANT      {priv_objeto [,priv_objeto] … | ALL [PRIVILEGES]}
             [(columna [,columna]...)]
ON       [usuario.]objeto
TO          {usuario | rol | PUBLIC} [,{usuario | rol | PUBLIC} … ]
[WITH GRANT OPTION];
```

donde:

- ON especifica el objeto sobre el que se dan privilegios
- TO identifica a los usuarios o roles a los que se conceden los privilegios.
- ALL concede todos los privilegios sobre el objeto especificado
- WITH GRANT OPTION permite que el receptor del privilegio o rol se lo asigne a otros usuarios o roles.
- PUBLIC asigna los privilegios a todos los usuarios actuales y futuros. El propósito es garantizar el acceso a determinados objetos de todos los usuarios de la BD.

Con la orden GRANT se pueden conceder privilegios INSERT, UPDATE o REFERENCES sobre determinadas columnas de una tabla.

Ejemplos:

Conceder a María todos los privilegios sobre TABLA1: `GRANT ALL ON TABLA1 TO María;`

Conceder a María el privilegio de modificar solo la columna apellido de TABLA1:

```
GRANT UPDATE (apellido) ON TABLA1 TO María;
```

Conceder a María el privilegio de insertar en TABLA1, y además, para que ella pueda pasar este privilegio a otros usuarios.

```
GRANT INSERT ON TABLA1 TO María WITH GRANT OPTION;
```

Ahora María puede conceder el permiso de insertar a otros usuarios sobre la tabla TABLA1

Retirada de privilegios

Para retirar privilegios está la orden REVOKE.

La sintaxis para retirar privilegios de objetos es:

```
REVOKE      {priv_objeto [, priv_objeto] … | ALL  [PRIVILEGES]}
ON        [usuario.]objeto
FROM        {usuario|rol|PUBLIC} [,{usuario|rol|PUBLIC} …];
```

La sintaxis para retirar privilegios de sistema o roles a los usuarios es:

```
REVOKE      {priv_sistema | rol} [, {priv_sistema | rol}] … |
FROM        {usuario|rol|PUBLIC} [,{usuario|rol|PUBLIC} …];
```

Ejemplo: Quitar los privilegio SELECT y UPDATE sobre TABLA1 a María

```
REVOKE SELECT, UPDATE ON TABLA1 FROM María;
```

6.2 ROLES

Un conjunto de privilegios puede agruparse en un rol, para facilitar la tarea de asignación de ese conjunto e privilegios a otros usuarios o a otros roles. Los privilegios de un rol pueden ser de sistema y a nivel de objeto.

Primero se debe crear un rol con la orden:

```
CREATE ROLE Nombre_Rol;
```

Conceder privilegios a los roles

Una vez creado se le conceden privilegios con la orden GRANT. Ejemplo: Asignamos privilegios al rol Acceso: `GRANT SELECT, INSERT ON EMPLEADOS TO Acceso;`

Supresión de privilegios en los roles

Con la orden REVOKE se puede suprimir los privilegios dados a los roles. Ejemplo: Retirar el privilegio INSERT en la tabla empleados al rol Acceso

```
REVOKE INSERT ON EMPLEADOS TO Acceso;
```

Supresión de un rol

```
DROP ROLE NombreRol;
```

6.3 PERFILES

Un perfil es un conjunto de límites a los recursos de la BD. Se pueden utilizar perfiles para poner límites a la cantidad de recursos del sistema y de la BD disponibles para un usuario y para gestionar las restricciones de contraseña.

La sintaxis para crear un perfil es:

CREATE PROFILE nombrePerfil LIMIT

{parámetros_recursos | parámetros_contraseña}

{ Entero [K|M] | UNLIMITED | DEFAULT};

Donde:

- parámetros_recursos: SESSIONS_PER_USER; CPU_PER_SESION, CPU_PER_CALL, CONNECT_TIME, …
- parámetros_contraseña: FAILED_LOGIN_ATTEMPTS, PASSWORD_LIFE_TIME, ….

Para activar el uso de perfiles en el sistema, el administrador ha de ejecutar la orden

```
ALTER SYSTEM SET RESOURCE_LIMIT=TRUE;
```

Para asignar un perfil a un usuario se puede utilizar la orden ALTER USER:

```
ALTER USER USUARIO PROFILE nombreperfil;
```

Borrado de un perfil

```
DROP PROFILE nombreperfil [CASCADE];
```

7 GESTIÓN DE LA SEGURIDAD EN MYSQL

7.1 EL SISTEMA DE PERMISOS EN MYSQL

El sistema de privilegios de MySQL se encarga de establecer quien puede conectar al servidor y de asegurar que cada usuario conectado ejecute solamente las operaciones que tenga permitidas.

El servidor trabaja con cuenta (equipo + usuario + contraseña) que tienen ciertos permisos o privilegios. Dichas cuentas y permisos se encuentran organizados en las base de datos llamada mysql.

El acceso al servidor tiene lugar en dos etapas:

- se comprueba nuestra identidad
- se comprueban nuestros permisos para hacer operaciones sobre objetos del SGBD

Tablas de permisos

MySQL almacena la información de sus usuarios y privilegios en una base de datos, cuyo nombre es mysql. Podemos ver su contenido con show tables;

El servidor usa las tablas user, db y host en la base de datos mysql en ambas etapas del control del acceso. Las columnas en estas tablas de permisos se muestran a continuación:

Nombre tabla	user	db	host
Alcance columnas	Host	Host	Host
	User	Db	Db
	Password	User	
Columnas privilegios	Select_priv	Select_priv	Select_priv
	Insert_priv	Insert_priv	Insert_priv
	Update_priv	Update_priv	Update_priv

	Delete_priv	Delete_priv	Delete_priv
	Index_priv	Index_priv	Index_priv
	Alter_priv	Alter_priv	Alter_priv
	Create_priv	Create_priv	Create_priv
	Drop_priv	Drop_priv	Drop_priv
	Grant_priv	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv	
	Alter_routine_priv	Alter_routine_priv	
	References_priv	References_priv	References_priv
	Reload_priv		
	Shutdown_priv		
	Process_priv		
	File_priv		
	Show_db_priv		
	Super_priv		
	Create_tmp_table_priv	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv	Lock_tables_priv
	Execute_priv		
	Repl_slave_priv		
	Repl_client_priv		
Columnas seguridad	ssl_type		
	ssl_cipher		
	x509_issuer		
	x509_subject		
Columnas recursos control	max_questions		
	max_updates		
	max_connections		
	max_user_connections		

Durante la segunda etapa de control de acceso, el servidor efectúa una verificación de petición para asegurar que cada cliente tiene suficientes privilegios para cada petición que recibe. Además de las tablas de permisos `user`, `db`, y `host` , el servidor puede consultar las tablas `tables_priv` y `columns_priv` para peticiones que impliquen tablas. Las tablas `tables_priv` y `columns_priv` proveen de un control de privilegios más fino a nivel de tabla y columna. Tienen las siguientes columnas:

Nombre tabla	**tables_priv**	**columns_priv**
Alcance de columnas	Host	Host
	Db	Db
	User	User
	Table_name	Table_name

		Column_name
Columnas privilegios	Table_priv	Column_priv
	Column_priv	
Otras columnas	Timestamp	Timestamp
	Grantor	

Los permisos de la tabla user son globales, es decir se aplican a todas las bases de datos. Si tenemos un permiso a "Y" en un permiso de dicha tabla podremos hacer dicha operación sobre cualquier tabla de cualquier base de datos de nuestro sistema. Si por el contrario está a "N" pasaremos al siguiente nivel, es decir a la tabla db y así sucesivamente con el resto de tablas tables_priv y columns_priv.

Cada tabla de permisos contiene columnas de alcance y columnas de privilegios:

Las **columnas de alcance** determinan el alcance de cada entrada (registro) en las tablas, es decir, el contexto en que el registro se aplica. Ejemplo, un registro de la tabla user con los valores Host (miempresa.es) y User (pepito) se usaría para autenticar conexiones hechas al servidor desde el equipo miempresa.com por el cliente pepito. De forma similar, un registro de la tabla db con la columnas Host, User y Db con los valores miempresa.es, pepito y empresa se usaría cuando pepito conectase desde el equipo miempresa.es para acceder a la base de datos empresa. Las tablas `tables_priv` y `columns_priv` contienen columnas de alcance indicando tablas o combinaciones de tabla/columna sobre las que tiene permiso cierta cuenta.

Las **columnas de privilegios** indican que privilegios se otorgan a una cuenta sobre ciertas bases, tablas y/o columnas y otros objetos del SGBD, es decir que operaciones pueden ejecutarse. El servidor combina la información de diversas tablas de permisos para tener una descripción completa de lo permisos de un usuario.

En las tablas `user`, `db` y `host`, cada privilegio se lista en una columna separada que se declara como ENUM('N','Y') DEFAULT 'N'. Es decir, cada privilegio puede estar desactivado o activado, estando desactivados por defecto.

En las tablas tables_priv, columns_priv, y procs_priv, las columnas de privilegios se declaran como columnas de tipo SET. Los valores de estas columnas pueden contener cualquier combinación de los privilegos que afectan a tablas, columnas y procesos respectivamente, como se ve en la siguiente figura:

Nombre de tabla	**Nombre de columna**	**Posible conjunto de elementos**
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

El servidor usa la tabla de permisos como sigue:

- Las columnas de alcance (user y host) de la tabla user determinan si se rechazan o permiten conexiones entrantes. Para conexiones permitidas, cualquier privilegio otorgado en la tabla user indica los privilegios globales del usuario (superusuario). Estos privilegios se aplican a *todas* las bases de datos en el servidor.
- Las columnas de alcance de la tabla db determinan qué usuarios pueden acceder a qué bases de datos desde qué equipo. La columna de privilegios determina qué operaciones se permiten. Un privilegio otorgado a nivel de base de datos se aplica a la base de datos y a todas sus tablas.
- La tabla host se usa en conjunción con la tabla db cuando desea que un registro de la tabla db se aplique a varios equipos. Por ejemplo, si queremos que un usuario sea capaz de usar una base de datos desde varios equipos en su red, dejamos el valor Host vacío en el registro de usuario de la tabla `db` , luego rellenamos

la tabla host con un registro para cada uno de estos equipos.

Nota: La tabla host no se ve afectada por los comandos GRANT ni REVOKE . La mayoría de instalaciones MySQSL no necesitan usar esta tabla en absoluto.

- Las tablas tables_priv y columns_priv son similares a la tabla db, pero son más detalladas: se aplican a nivel de tabla y de columna en lugar de a nivel de base de datos. Un privilegio otorgado a nivel de tabla se aplica a la tabla y a todas sus columnas. Un privilegio otorgado a nivel de columna se aplica sólo a la columna especificada.
- La tabla procs_priv se aplica a rutinas almacenadas. Un privilegio otorgado a nivel de rutina se aplica sólo a una única rutina.

El servidor **mysqld** lee los contenidos de las tablas de permisos en memoria cuando arranca. Puede decirle que las vuelva a leer mediante el comando `FLUSH PRIVILEGES` o ejecutando los comandos **mysqladmin flush-privileges** o **mysqladmin reload** .

Cuando modifica los contenidos de las tablas de permisos, es una buena idea asegurarse que sus cambios configuran permisos tal y como desea. Para consultar los permisos de una cuenta dada, usamos el comando `SHOW GRANTS` . Por ejemplo, para determinar los permisos que se otorgan a una cuenta con valores Host y User de pc84.example.com y antonio, usamos este comando:

```
mysql> SHOW GRANTS FOR 'antonio'@'pc84.example.com';
```

Privilegios en MySQL

La información sobre los privilegios de las cuentas está almacenada en las tablas `user`, `db`, `host`, `tables_priv`, `columns_priv`, y `procs_priv` de la base de datos `mysql`.

Los nombres utilizados en las sentencias GRANT y REVOKE para referirse a privilegios se muestran en la siguiente tabla, junto al nombre de columna asociado con cada privilegio en las tablas grant y el contexto en que el privilegio se aplica.

Table 8.2 Permissible Static Privileges for GRANT and REVOKE

Privilege	Grant Table Column	Context
ALL [PRIVILEGES]	Synonym for “all privileges”	Server administration
ALTER	Alter_priv	Tables
ALTER ROUTINE	Alter_routine_priv	Stored routines
CREATE	Create_priv	Databases, tables, or indexes
CREATE ROLE	Create_role_priv	Server administration
CREATE ROUTINE	Create_routine_priv	Stored routines
CREATE TABLESPACE	Create_tablespace_priv	Server administration
CREATE TEMPORARY TABLES	Create_tmp_table_priv	Tables
CREATE USER	Create_user_priv	Server administration
CREATE VIEW	Create_view_priv	Views
DELETE	Delete_priv	Tables
DROP	Drop_priv	Databases, tables, or views
DROP ROLE	Drop_role_priv	Server administration
EVENT	Event_priv	Databases
EXECUTE	Execute_priv	Stored routines
FILE	File_priv	File access on server host
GRANT OPTION	Grant_priv	Databases, tables, or stored routines
INDEX	Index_priv	Tables
INSERT	Insert_priv	Tables or columns

Privilege	Grant Table Column	Context
LOCK TABLES	Lock_tables_priv	Databases
PROCESS	Process_priv	Server administration
PROXY	See proxies_priv table	Server administration
REFERENCES	References_priv	Databases or tables
RELOAD	Reload_priv	Server administration
REPLICATION CLIENT	Repl_client_priv	Server administration
REPLICATION SLAVE	Repl_slave_priv	Server administration
SELECT	Select_priv	Tables or columns
SHOW DATABASES	Show_db_priv	Server administration
SHOW VIEW	Show_view_priv	Views
SHUTDOWN	Shutdown_priv	Server administration
SUPER	Super_priv	Server administration
TRIGGER	Trigger_priv	Tables
UPDATE	Update_priv	Tables or columns
USAGE	Synonym for “no privileges”	Server administration

Explicación más detallada de estos privilegios se puede encontrar en el manual de referencia de MySQL.

Control de acceso detallado

Veamos el proceso detallado desde que accedemos al sistema con las credenciales correspondientes hasta que intentamos ejecutar cualquier tipo de comando sql.

Control de acceso, nivel 1: Comprobación de la conexión

Cuando intente conectar a un servidor MySQL, el servidor aceptará o rechazará la conexión basándose en su identidad y si usted puede identificar su identidad proporcionando la clave correcta. Si no es así, el servidor le denegará el acceso completamente. En caso contrario, el servidor acepta la conexión, y entra en el estado o nivel 2, y espera peticiones.

Nuestra identidad se basa en dos elementos de información:

- El nombre de máquina cliente (o IP) desde donde nos conectamos
- Nuestro nombre de usuario MySQL

La comprobación de la identidad se realiza utilizando las tres columnas de la tabla *user (Host, User,* y *Password*). El servidor sólo acepta la conexión si las columnas *Host* y *User* de alguna de las tablas *user* es coincidente con el nombre de máquina y usuario del cliente, y además el cliente proporciona la clave especificada en ese registro.

Los valores de *Host* en la tabla *user* pueden ser especificados de las siguientes maneras:

- Un valor de *Host* debe ser un nombre de máquina o un número IP, o 'localhost' para indicar la máquina local. Para valores de *Host* especificados como números IP, se puede especificar una máscara de red indicando cuantos bits de la dirección utilizar para el número de red. Por ejemplo:

```
mysql> GRANT ALL PRIVILEGES ON db.* TO david@'192.58.197.0/255.255.255.0';
```

Esto permite a david conectarse desde cualquier cliente que tenga un número IP dentro de la red 192.58.197.0

La máscara de red solo puede ser utilizada para decirle al servidor que use 8,16, 24 o 32 bits para la dirección.

- Puede utilizar los caracteres comodín '%' y '_' en los valores de las columnas Host. Estos tienen el mismo significado que en las operaciones de búsqueda de patrones realizadas mediante el operador LIKE. Por ejemplo, un valor de Host igual a '%' retorna cualquier nombre de máquina, así como un valor de '%.mysql.com' retorna cualquier nombre de máquina en el dominio mysql.com.
- Un valor vacío de *Host* en un registro de la tabla *db* significa que los privilegios de dicho registro deben ser combinados con aquellos que se encuentren en el registro de la tabla host que concuerde con el nombre del cliente. Los privilegios se combinan utilizando operaciones AND (intersección), no OR (union).

En la columna *User*, los caracteres comodín no están permitidos, pero puede especificar un valor en blanco, que será válido para cualquier nombre. Si el registro de la tabla *user* que concuerda con una conexión entrante tiene un valor vacío, el usuario es considerado anónimo, sin nombre de usuario, no un usuario con el nombre que el cliente especificó realmente.

La columna *Password* puede estar vacía. Esto no es un comodín que permite que cualquier clave sea permitida. Significa que el usuario debe conectarse sin especificar una clave.

Los valores que no están vacíos de *Password* en la tabla *user* representan claves cifradas. MySQL no almacena las claves en forma de texto llano para que cualquiera pueda verlo. En vez de esto, la clave suministrada por un usuario que se está intentando conectar es cifrada (utilizando la función PASSWORD()). La clave cifrada se utiliza entonces durante el proceso de conexión en el momento de comprobar si es correcta. (Esto se realiza sin que la clave cifrada viaje nunca sobre la conexión.) Desde el punto de vista de MySQL, la clave cifrada es la clave REAL, así que no debería darse acceso a ella a nadie. En concreto, no de acceso de lectura a las tablas de la base de datos mysql a usuarios no-administrativos.

Es posible que el nombre del cliente y del usuario de una conexión entrante concuerde con más de un registro en la tabla *user*.

Cuando hay la posibilidad de múltiples concordancias, el servidor debe determinar cual de ellas utilizar. El problema se resuelve de la siguiente manera:

- Siempre que el servidor lee la tabla *user* a memoria, ordena los registros.
- Cuando un cliente intenta conectar, el servidor mira a través de los registros en el orden establecido.
- El servidor utiliza el primer registro que concuerda con el nombre y usuario del cliente.

Para ver como esto ocurre, supongamos que la tabla *user* es como esta:

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| %         | root     | …
| %         | jeffrey  | …
| localhost | root     | …
| localhost |          | …
+-----------+----------+-
```

Cuando el servidor lee la tabla, ordena las entradas con los valores de *Host* más específicos primero. Los nombres de cliente y números de IP son los más específicos. El comodín '%' significa ``cualquier cliente'' y es menos específico. Registros con el mismo valor de *Host* se ordenan con el valor de *User* más específico (un valor de *User* en blanco significa ``cualquier usuario'' y es menos específico). En la tabla *user* recién mostrada, el resultado después de ordenar sería el siguiente:

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| localhost | root     | … ...
| localhost |          | … ...
| %         | jeffrey  | … ...
| %         | root     | … ...
+-----------+----------+-
```

Cuando un cliente intenta conectar, el servidor mira los registros ordenados y utiliza la primera concordancia. Para una conexión desde `localhost` por `jeffrey`, dos de las entradas de la tabla concuerdan: la primera con los valores `'localhost'` y '' de *Host* y *User* respectivamente, y el registro con los valores `'%'` y `'jeffrey'`. El registro con valor `'localhost'` aparece primero en la tabla ordenada, así que es el que el servidor utiliza.

Si puedes conectar al servidor, pero tus privilegios no son los que esperas, probablemente estás siendo identificado como algún otro usuario. Para averiguar qué cuenta de usuario utilizó el servidor para identificarte, usa la función

`CURRENT_USER()` que devuelve un valor en formato *`user_name`*`@`*`host_name`* que indica los valores de *User* y *Host* del registro concordante en la tabla *user*. Supongamos que `jeffrey` conecta y ejecuta la siguiente sentencia:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

El resultado mostrado indica que el registro que concuerda en la tabla *user* tiene un valor vacío en la columna *User*. En otras palabras, el servidor trata a jeffrey como a un usuario anónimo., lo que puede dar lugar a errores.

Nivel 2: Verificación de permisos

Una vez establecida una conexión, el servidor entra en el estado 2 del control de acceso. Por cada petición que viene en la conexión, el servidor determina que operación realizar, y entonces comprueba si tiene suficientes privilegios para hacerlo. Aquí es donde las columnas de privilegios de las tablas grant entran en juego. Estos privilegios puede venir de cualquiera de las tablas *user, db*, *host*, *tables_priv*, o *columns_priv*.

La tabla *user* otorga privilegios que se asignan de manera global, y que se aplican sin importar sobre qué base de datos trabajamos. Por ejemplo, si la tabla *user* nos otorga el privilegio DELETE, podremos borrar registros de cualquier tabla en cualquier base de datos en todo el servidor. En otras palabras, los privilegios de la tabla *user* son privilegios de superusuario. Es aconsejable otorgar privilegios en la tabla *user* sólo a superusuarios tales como administradores de base de datos. Para otros usuarios, deberíamos dejar los privilegios de la tabla *user* con el valor 'N' y otorgar los privilegios únicamente a niveles más específicos. Podemos otorgar privilegios para bases de datos, tablas o columnas particulares.

Las tablas *db* y *host* otorgan privilegios específicos para una base de datos. Los valores en las columnas de estas tablas pueden tener los siguientes formatos:

- Un valor en blanco de la columna *User* en cualquiera de las tablas concuerda con el usuario anónimo.
- Los caracteres comodín '%' y '_' pueden utilizarse en las columnas *Db* de cualquiera de las tablas.
- Un valor de '%' en la columna *Host* de la tabla *db* significa ``cualquier host.'' Un valor vacío en la columna *Host* de la tabla *db* significa ``consulta la tabla host para más información''.
- Un valor de '%' o en blanco en la columna *Host* de la tabla *host* significa ``cualquier host.''
- Un valor de '%' o en blanco de la columna *Db* en cualquiera de las dos tablas, significa ``cualquier base de datos.''

El servidor lee y ordena las tablas *db* y *host* al mismo tiempo que lee la tabla *user*. El servidor ordena la tabla *db* basándose en el rango de las columnas *Host*, *Db* y *User*, y ordena la tabla *host* basándose en el rango de las columnas *Host* y *Db*. Igual que con la tabla *user*, la ordenación coloca los valores menos específicos en última posición, y cuando el servidor busca correspondencias, utiliza la primera que encuentra.

Las tablas *tables_priv* y *columns_priv* otorgan privilegios específicos para tablas y columnas respectivamente. Los valores en las columnas de rango de estas tablas pueden tener los siguientes formatos:

- Los caracteres comodín '%' y '_' pueden ser utilizados en la columna *Host* de cualquiera de las tablas.
- Un valor de '%' o vacío en la columna *Host* de cualquiera de las tablas significa ``cualquier host.''
- Las columnas *Db*, *Table_name* y *Column_name* no pueden contener caracteres comodín ni estar en blanco en ninguna de las tablas.

El servidor ordena las tablas *tables_priv* y *columns_priv* basándose en las columnas *Host*, *Db*, y *User*. Esto es similar a la ordenación de la tabla *db*, pero más simple, porque únicamente la columna *Host* puede contener comodines.

El proceso de verificación de peticiones se describe aquí:

Para peticiones que requieran privilegios de administrador, como SHUTDOWN o RELOAD, el servidor comprueba

únicamente el registro de la tabla *user* porque es la única tabla que especifica los privilegios administrativos. El acceso se otorga si el registro permita la operación demandada, y es denegado en caso contrario. Por ejemplo, si usted quisiera ejecutar **mysqladmin shutdown**, pero su registro de la tabla *user* no le otorga el privilegio SHUTDOWN, el servidor deniega el acceso sin ni siquiera consultar las tablas *db* o *host*.

Para peticiones sobre bases de datos (INSERT, UPDATE, etc.), el servidor primero comprueba los privilegios globales del usuario (superuser) mirando el registro de la tabla *user*. Si el registro permite la operación demandada, se otorga el acceso. Si los privilegios globales de la tabla *user* son insuficientes, el servidor determina los privilegios específicos sobre la base de datos comprobando las tablas *db* y *host*:

1. El servidor busca en la tabla *db* una concordancia en las columnas *Host*, *Db* y *User*. Las columnas *Host* y *User* se hacen concordar con el nombre de *host* y de usuario MySQL. La columna *Db* se hace concordar con la base de datos a la que el usuario quiere acceder. Si no hay ningún registro para *Host* y *User*, se deniega el acceso.
2. Si hay un registro que concuerda en el registro de la tabla *db* y su columna *Host* no está vacía, ese registro define los privilegios específicos del usuario en la base de datos.
3. Si la columna *Host* del registro concordante de la tabla *db* se encuentra vacía, significa que la tabla *host* enumera qué hosts pueden tener acceso a la base de datos. En este caso, una comprobación más se realiza en la tabla *host* para encontrar una concordancia en las columnas *Host* y *Db.* Si ningún registro de la tabla *host* concuerda, se deniega el acceso. Si hay una concordancia, los privilegios específicos sobre la base de datos del usuario son calculados como la intersección (¡*no* unión!) de los privilegios en los registros de las tablas *db* y *host*; es decir, los privilegios que tienen valor 'Y' en ambos registros. (De esta manera puede otorgar privilegios en el registro de la tabla *db* y entonces restringir selectivamente host a host utilizando los registros de la tabla *host*.)

Tras determinar los privilegios específicos de la base de datos otorgados por los registros de las tablas *db* y host, el servidor los añade a los privilegios globales otorgados por la tabla *user*. Si el resultado permite la operación demandada, se otorga el acceso. En caso contrario, el servidor comprueba sucesivamente la tabla del usuario y los privilegios de las columnas en las tablas tables_priv y columns_priv, los añade a los privilegios del usuario, y permite o deniega el acceso basándose en el resultado.

También se puede utilizar la tabla *host* para indicar hosts que *no* son seguros. Supongamos que se tiene una máquina public.su.dominio que está situada en un lugar público que no se considera seguro. Se puede permitir el acceso a todos los hosts de la red excepto a esa máquina utilizando registros de la tabla *host* como este:

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

Naturalmente, se debe siempre comprobar las entradas en las tablas grant (por ejemplo, utilizando SHOW GRANTS o **mysqlaccess**) para estar seguro de que los privilegios de acceso son realmente los que pensamos que son.

Para consultar los privilegios usamos `SHOW PRIVILEGES`

Cuándo tienen efecto los cambios de privilegios

Cuando **mysqld** se inicia, todos los contenidos de las tablas *grant* se leen a memoria y se hacen efectivas para el control de acceso en ese punto.

Cuando el servidor recarga las tablas *grant*, los privilegios para los conexiones de clientes existentes se ven afectadas de la siguiente manera:

- Los cambios en los privilegios de tabla y columna toman efecto en la siguiente petición del cliente.
- Los cambios en privilegios de base de datos toman efecto en la siguiente sentencia USE *db_name*.
- Los cambios a los privilegios globales y las claves de acceso toman efecto la próxima vez que el cliente se

conecte.

Si se modifica las tablas *grant* utilizando GRANT, REVOKE, o SET PASSWORD, el servidor se da cuenta de estos cambios y recarga las tablas *grant* en la memoria inmediatamente.

Si se modifica las tablas *grant* directamente utilizando sentencias como INSERT, UPDATE, o DELETE, los cambios no tendrán efecto en la comprobación de privilegios hasta que se reinicie el servidor, o bien se le comunique a éste que debe recargar las tablas. Para recargar las tablas manualmente, se ejecuta la sentencia `FLUSH PRIVILEGES` o los comandos **mysqladmin flush-privileges** o **mysqladmin reload**.

Si modificamos las tablas *grant* directamente pero olvidamos recargarlas, los cambios *no tienen efecto* hasta que reinicie el servidor.

7.2 USUARIOS Y PRIVILEGIOS

El acceso al servidor MySQL está controlado por usuarios y privilegios. El usuario administrador del sistema MySQL se llama root. Además de este usuario, se incluye el usuario anónimo, que tiene permisos sobre la base de datos *test.*

La administración de usuarios y privilegios en MySQL se realiza a través de las siguientes sentencias:

- **`CREATE USER`** → Crea un usuario
- **`GRANT`** → Otorga privilegios a un usuario, en caso de no existir se crea el usuario.
- **`REVOKE`** → elimina los privilegios de un usuario existente.
- **`SET PASSWORD`** → asigna una contraseña.
- **`DROP USER`** → elimina un usuario

CREAR USUARIOS

MySQL es un sistema de gestión de bases de datos claramente orientado a la web, y una de los síntomas en su arquitectura ha venido siendo que la creación de los usuarios se realiza en la misma sentencia que el permiso (grant) de acceso a una o varias bases de datos. La orientación de MySQL va cambiando con el tiempo y el uso que se le da a las bases de datos cada vez trasciende más el entorno web,.

La sentencia `CREATE USER`

A partir de la versión MySQL 5.0.2 existe la posibilidad de crear usuarios sin necesidad de asignarles privilegios, utilizando la sentencia **`CREATE USER`**.

Por ejemplo, para crear el usuario eva:

```
CREATE USER 'eva'@'localhost' IDENTIFIED BY 'fer_pass';
```

Al igual que con la sentencia GRANT, el contexto `'localhost'` define que el usuario solamente se puede conectar desde el servidor de MySQL, y el `IDENTIFIED BY` define el password del usuario, se puede omitir, para un usuario sin password, siempre que el modo SQL no sea NO_AUTO_CREATE_USER.

Los privilegios necesarios para ejecutar la sentencia **`CREATE USER`** son `CREATE USER`

LA SENTENCIA GRANT

La sintaxis básica de la sentencia GRANT consta de tres secciones obligatorias:

```
GRANT lista_privilegios
ON base_de_datos.tabla
TO usuario
```

Es la misma sintaxis que vimos en ORACLE. Veamos algunos ejemplos de como se puede usar `GRANT`:

Aquí otorgamos los permisos `UPDATE, INSERT, SELECT` sobre la tabla precios de la base de datos demo al usuario victor que se conecta desde la máquina local.

```
GRANT UPDATE, INSERT, SELECT ON demo.precios TO 'victor'@'localhost';
```

Podemos otorgar permisos a más de un usuario:

```
GRANT UPDATE, INSERT, SELECT ON demo.precios TO 'victor'@'localhost', 'pedro'@'equ.es';
```

Ahora se permite el acceso del usuario victor desde cualquier equipo del dominio 'empresa.com'

```
GRANT UPDATE, INSERT, SELECT ON demo.precios TO 'victor'@'%.empresa.com';
```

En el siguiente ejemplo otorgamos privilegios sobre todas las tablas de de la base de datos demo

```
GRANT ALL ON demo.* TO 'victor'@'localhost';
```

En la siguiente tabla vemos las opciones resumidas para la clausula ON del comando GRANT

Opción	Significado
.	Todas las bases de datos y todas las tablas
Base.*	Todas las tablas de la base de datos especificada
tabla	Tabla especificada de la base de datos en uso
*	Todas la tablas de la base de datos en uso

En el siguiente ejemplo podemos especificar las columnas sobre las que se otorgan privilegios con el comando `GRANT`

```
GRANT UPDATE(precio), SELECT(precio,empresa) ON demos.precios TO 'victor'@'localhost';
```

LIMITAR RECURSOS DE CUENTAS

Una forma de limitar los recursos de los servidores MySQL es asignar a la variable de sistema `max_user_connections` un valor distinto de cero. Sin embargo,este método es estrictamente global, y no está permitido para la administración de cuentas individuales. Además, limita sólo el número de conexiones simultáneas hechas usando una sola cuenta, y no lo que un cliente puede hacer una vez conectado.

En MySQL se puede limitar los siguientes recursos de servidor para cuentas individuales:

- El número de consultas que una cuenta puede realizar por hora
- El número de actualizaciones que una cuenta puede hacer por hora
- El número de veces que una cuenta puede conectar con el servidor por hora

Cualquier comando que un cliente puede realizar cuenta en el límite de consultas. Sólo los comandos que modifiquen la base de datos o las tablas cuentan en el límite de actualizaciones.

Una cuenta en este contexto es un registro en la tabla `user` . Cada cuenta se identifica unívocamente por los valores de las columnas `User` y `Host`.

Como prerrequisito para usar esta característica, la tabla `user` en la base de datos mysql debe contener las columnas relacionadas con el recurso. Los límites de recursos se guardan en las columnas `max_questions`, `max_updates`, `max_connections`, y `max_user_connections`.

Para cambiar el límite de recursos con un comando `GRANT` usamos la cláusula `WITH` que nombra cada recurso a ser limitado y un contador por hora indicando el valor límite. Por ejemplo, para crear una nueva cuenta que pueda acceder a la base de datos customer, pero sólo de forma limitada, utilizamos este comando:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     IDENTIFIED BY 'frank'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
```

```
    ->          MAX_USER_CONNECTIONS 2;
```

No todos los tipos de límites necesitan nombrarse en la cláusula `WITH`, pero los nombrados pueden presentarse en cualquier orden. El valor para cada límite por hora debe ser un entero representando el contador por hora. Si el comando `GRANT` no tiene cláusula `WITH`, los límites se inicializan con el valor por defecto de cero (o sea, sin límite). Para `MAX_USER_CONNECTIONS`, el límite es un entero indicando el máximo número de conexiones simultáneas que la cuenta puede hacer en cualquier momento. Si el límite asignado es el valor por defecto de cero, la variable de sistema `max_user_connections` determina el número de conexiones simultáneas para la cuenta.

Para inicializar o cambiar los límites de una cuenta existente, debemos usar un comando `GRANT USAGE` a nivel global (`ON *.*`). El siguiente comando cambia el límite de consultas para `francis` a 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

Este comando deja los permisos existentes de la cuenta inalterados y modifica sólo los valores cuyo límite se especifica.

Para eliminar un límite existente, ponga su valor a cero. Por ejemplo, para eliminar el límite de cuántas veces por hora puede conectar francis , usamos este comando:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

El conteo de recursos se hace por cuenta, no por cliente. Por ejemplo, si una cuenta tiene un límite de 50 consultas, no puede incrementar el límite a 100 haciendo dos conexiones simultáneas al servidor. Las consultas de ambas conexiones se cuentan juntas.

El contador actual por hora de uso de recursos puede reiniciarse globalmente para todas las cuentas, o individualmente para una cuenta dada:

- Para reiniciar los contadores actuales a cero para todas las cuentas, ejecute el comando `FLUSH USER_RESOURCES`. Los contadores también pueden reiniciarse recargando las tablas de permisos (por ejemplo, con un comando `FLUSH PRIVILEGES` o **`mysqladmin reload`**).
- Los contadores para una cuenta individual pueden ponerse a cero cambiando cualquiera de sus límites. Para hacerlo, use `GRANT USAGE` como se ha descrito anteriormente y especifique un valor límite igual al valor que tiene la cuenta en ese momento.

Los reinicios de contadores no afectan el límite `MAX_USER_CONNECTIONS` .

Todos los contadores empiezan a cero cuando el servidor arranca; los contadores no se guardan al reiniciar.

ASIGNAR CONTRASEÑAS A CUENTAS

Se pueden asignar contraseñas desde la línea de comandos usando el comando **mysqladmin** :

```
mysqladmin -u nombres_usuario -h equipo password "nuevacontr"
```

La cuenta para la que este comando cambia la contraseña es la que tiene un registro en la tabla user que coincida el user_name con la columna User y un equipo cliente desde el que se conecta en la columna Host.

Otra forma de asignar una contraseña en una cuenta es con el comando `SET PASSWORD`, cuya sintaxis es:

```
SET PASSWORD [FOR user] =
      {
            PASSWORD('some password') | 'encrypted password'
      }
```

Es decir, indicamos la cuenta y el password en texto claro usando la función password que lo encriptara o directamente encriptado.

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

Sólo los usuarios tales como root con acceso de modificación para la base de datos mysql pueden cambiar la contraseña de otro usuario. Si no estamos conectado como un usuario anónimo, podemos cambiar nuestra propia contraseña omitiendo la cláusula `FOR` :

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

Podemos usar el comando `GRANT USAGE` globalmente (`ON *.*`) para asignar una contraseña a una cuenta sin afectar los permisos actuales de la cuenta:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

Aunque generalmente es preferible asignar contraseñas usando uno de los métodos precedentes, puede hacerlo modificando la tabla user directamente:

- Para establecer una contraseña al crear una nueva cuenta, especifique un valor para la columna `Password`:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

- Para cambiar la contraseña en una cuenta existente, use UPDATE para especificar el valor de la columna Password:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('bagel')
    -> WHERE Host = '%' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```

Cuando especifique una contraseña en una cuenta mediante SET PASSWORD, INSERT, o UPDATE, debe usar la función `PASSWORD()` para cifrarla.

Si inicializa la contraseña usando el comando GRANT ... IDENTIFIED BY o **mysqladmin password** , ambos cifran la contraseña. En estos casos, el uso de la función PASSWORD() no es necesario.

ELIMINAR PRIVILEGIOS

El comando REVOKE puede eliminar privilegios otorgados con GRANT a los usuarios. Veamos un ejemplo:

```
REVOKE ALL ON *.* FROM visitante@localhost;
```

Al ejecutar este comando se le retiran al usuario visitante todos sus privilegios sobre todas las bases de datos, cuando se conecta desde localhost.

El comando anterior no retira todos los privilegios del usuario visitante, sólo se los retira cuando se conecta desde localhost. Si el usuario se conecta desde otra máquina y tenía permisos para hacerlo, sus privilegios quedarán intactos.

ELIMINAR CUENTAS DE USUARIO

Antes de proceder a la eliminación de usuarios, es necesario asegurarse de que se les ha quitado primero todos sus privilegios. Una vez asegurado este detalle, se procede a eliminarlo mediante el comando `DROP USER`.

```
DROP USER visitante
```

ROLES EN MYSQL

En MySQL, los **roles** son una característica que te permite gestionar permisos de manera más sencilla y eficiente, especialmente cuando tienes muchos usuarios con permisos similares. Los roles te permiten agrupar permisos y asignarlos a usuarios, evitando tener que asignar los mismos permisos de forma individual a cada uno.

Creación de roles

A partir de **MySQL 8.0**, se introdujo el soporte de roles. Para crear un rol, utilizamos el comando `CREATE ROLE`:

```
CREATE ROLE 'nombre_del_rol';
```

Asignar permisos a un rol

Una vez creado el rol, puedes asignar permisos a este rol utilizando el comando `GRANT`:

```
GRANT SELECT, INSERT, UPDATE ON base_de_datos.* TO 'nombre_del_rol';

GRANT SELECT, INSERT ON mi_base_de_datos.* TO 'admin_rol';
```

Asignar un rol a un usuario

Después de definir un rol, lo puedes asignar a un usuario:

```
GRANT 'nombre_del_rol' TO 'usuario'@'localhost';

GRANT 'admin_rol' TO 'usuario1'@'localhost';
```

Activar un rol para un usuario

Para que un usuario empiece a usar un rol, debes ejecutar el comando `SET ROLE`. Un usuario puede tener múltiples roles, y puedes activar uno o varios:

```
SET ROLE 'nombre_del_rol';

SET ROLE 'admin_rol';
```

Si quieres que el rol sea el predeterminado cuando el usuario inicie sesión, puedes configurar el rol con el comando `SET DEFAULT ROLE:`

```
SET DEFAULT ROLE 'nombre_del_rol' TO 'usuario'@'localhost';
```

Eliminar un rol

Si ya no necesitas un rol, puedes eliminarlo con el comando `DROP ROLE`:

```
DROP ROLE 'nombre_del_rol';

DROP ROLE 'admin_rol';
```

Roles Predeterminados

Algunos roles predeterminados en MySQL incluyen:

- **USAGE**: Este rol no otorga permisos, pero se puede usar como base para la creación de otros roles.
- **DB_ADMIN**: Puede realizar cualquier operación administrativa en las bases de datos.
- **DB_OWNER**: Este rol tiene control total sobre las bases de datos, similar al rol de administrador.

Visualizar Roles y Permisos

Puedes consultar los roles asignados a un usuario con la siguiente consulta:

```
SELECT * FROM information_schema.user_roles WHERE grantee = "'usuario'@'localhost'";
```

Para verificar los permisos de un rol:

```
SHOW GRANTS FOR 'nombre_del_rol';
```

7.3 CONEXIONES SEGURAS

MySQL puede establecer conexiones seguras encriptándolas mediante el protocolo SSL, de esta manera, los datos que se transmiten (tanto la consulta en un sentido, como el resultado, en el otro) entre el cliente y el servidor estarán protegidos contra intrusos. Para especificar que un usuario se debe conectar con este protocolo, se utiliza la cláusula `require`.

```
GRANT ALL ON *.* TO usuario@localhost REQUIRE SSL;
```

Las conexiones encriptadas ofrecen protección contra el robo de información, pero suponen una carga adicional para el servicio, que debe desencriptar la petición del cliente y encriptar la respuesta, por ello, merman el rendimiento del sistema. Si da tiempo os pasaré un Anexo con la forma de crear una conexión segura a nuestro servidor MySQL.