

Tratamiento de los datos

Contenidos

- ☞ Herramientas gráficas proporcionadas por el SGBD
- ☞ Sentencia INSERT
- ☞ INSERT y SELECT
- ☞ Sentencia UPDATE
- ☞ Sentencia DELETE
- ☞ UPDATE y DELETE con subconsultas
- ☞ Borrado y modificación de registros con relaciones
- ☞ Transacciones
- ☞ Acceso concurrente a los datos
- ☞ Vistas, usuarios y privilegios

Objetivos

- ☞ Identificar herramientas y sentencias para modificar el contenido de la base de datos
- ☞ Insertar, borrar y actualizar datos en las tablas
- ☞ Incluir en una tabla información de una consulta
- ☞ Adoptar medidas para mantener la integridad y consistencia de la información
- ☞ Reconocer el funcionamiento de transacciones y anular parcial o totalmente cambios producidos por una transacción
- ☞ Identificar efectos de las políticas de bloqueo de registros
- ☞ Crear vistas y usuarios y asignar privilegios

En este capítulo se detalla la sintaxis de las sentencias INSERT, UPDATE y DELETE. Se expone el tratamiento de las transacciones y los problemas del acceso concurrente a los datos. Además, se introducen las principales herramientas gráficas de edición de datos y se explica cómo crear vistas, usuarios y otorgar y revocar permisos.

5.1. Herramientas gráficas para la edición de los datos

Existen multitud de herramientas gráficas para la edición de datos, algunas, incorporadas como parte del software del gestor de base de datos, por ejemplo, el entorno gráfico de Access; otras herramientas se distribuyen como paquetes a añadir al SGBD, como phpMyAdmin de MySQL; y el software de terceros, programas por los que hay que pagar una licencia aparte como *TOAD* o *Aqua Data Studio*. Otros gestores como Oracle, no incorporan expresamente una herramienta de edición de datos como tal (se pueden consultar, pero no se puede editar datos desde Enterprise Manager), pero se aprovechan de herramientas de terceros para esta labor.

5.1.1. Edición con phpMyAdmin

Una de las muchas utilidades de phpMyAdmin es la inserción de datos a través de formularios web, donde, de forma muy sencilla, se escriben los valores para cada campo de la tabla deseada. Tan solo hay que seleccionar la pestaña *Insertar* y seleccionar la tabla donde se va a insertar los registros. Se llenan los valores y se pulsa continuar.

Campo	Tipo	Función	Nulo	Valor
Código	int(11)		<input type="checkbox"/>	1
Nombre	varchar(50)		<input type="checkbox"/>	Leonard Nimoy
Fecha	date		<input checked="" type="checkbox"/>	1931-03-26
Nacionalidad	varchar(20)		<input type="checkbox"/>	EEUU

Campo	Tipo	Función	Nulo	Valor
Código	int(11)		<input type="checkbox"/>	2
Nombre	varchar(50)		<input type="checkbox"/>	William Shatner
Fecha	date		<input type="checkbox"/>	1931-03-22
Nacionalidad	varchar(20)		<input checked="" type="checkbox"/>	EEUU

Figura 5.1: Inserción de datos a través de phpmyadmin (paso 1).

De esta forma, se genera el código SQL automáticamente para insertar los valores.

¿Sabías que ... ? En MySQL existe la sintaxis **extended insert**, que permite insertar varios registros con un solo INSERT. Véase Figura. 5.2

The screenshot shows the phpMyAdmin interface after executing an SQL query. At the top, a message says "2 filas(s) fueron insertadas. La Id de la fila insertada es: 3". Below this is the SQL query:

```
INSERT INTO `startrek`.`Actores` (
  `Codigo`,
  `Nombre`,
  `Fecha`,
  `Nacionalidad`
)
VALUES
('1', 'Leonard Nimoy', '1931-03-26', 'EEUU'),
('2', 'William Shatner', '1931-03-22', 'EEUU');
```

On the right, there is a "Campos" (Fields) sidebar with "Codigo", "Nombre", "Fecha", and "Nacionalidad" listed. At the bottom, there are buttons for "[Editar]" and "[Crear código PHP]".

Ejecutar la(s) consulta(s) SQL en la base de datos startrek:

SQL code:

```
INSERT INTO `startrek`.`Actores` ('Codigo', 'Nombre', 'Fecha', 'Nacionalidad') VALUES
('1', 'Leonard Nimoy', '1931-03-26', 'EEUU'),
('2', 'William Shatner', '1931-03-22', 'EEUU');
```

Fields sidebar:

- Código
- Nombre
- Fecha
- Nacionalidad

Buttons at the bottom:

- Guardar esta consulta en favoritos: [checkbox]
- Permitir que todo usuario pueda acceder a este favorito: [checkbox]
- Reemplazar el favorito existente que tenga el mismo nombre: [checkbox]
- [Delimitador:]
- [Mostrar esta consulta otra vez]
- [Continuar]

Figura 5.2: Inserción a través de phpmyadmin.(paso 2).

Para eliminar o modificar registros, se utiliza la pestaña *Examinar*:

The screenshot shows the "Examinar" (Browse) tab of phpMyAdmin. At the top, it displays the result of a SELECT query:

```
Mostrando registros 0 - 1 (2 total, La consulta tardó 0.00004 seg)
SELECT *
FROM `Actores`
LIMIT 0 , 30
```

Below the results, there are options to change the display mode (horizontal or vertical), sort by key (None), and other settings like "Mostrar: 30 filas empezando de 0".

	Código	Nombre	Fecha	Nacionalidad
<input type="checkbox"/>	1	Leonard Nimoy	1931-03-26	EEUU
<input type="checkbox"/>	2	William Shatner	1931-03-22	EEUU

At the bottom, there are buttons for "Operaciones sobre los resultados de la consulta" (Operations on the query results) including "Vista de impresión" (Print View), "Previsualización para imprimir (documento completo)" (Preview for printing (full document)), "Exportar" (Export), and "CREATE VIEW".

Figura 5.3: Modificación y eliminación de datos a través de phpMyAdmin.

5.1.2. Access como entorno gráfico para otros gestores

Si se conoce el entorno Access es muy sencillo conectarlo con otros gestores, y de esta manera, aprovecharse del conocimiento de la interfaz de Microsoft para administrar otros gestores de bases de datos. Esto es posible gracias a la utilización de conectores ODBC. ODBC son las siglas de *Open Database Connectivity*. Es una

herramienta que, de forma estándar, permite conectar aplicaciones a cualquier gestor de bases de datos. Muchas aplicaciones ofimáticas, como Excel, Word y el propio Access permite el acceso remoto a datos de un SGBD. Para esto, tan solo es necesario instalar el driver ODBC para ese SGBD. De esta manera, se puede crear una DSN o *Data Source Name*, es decir, una referencia a cierta base de datos.

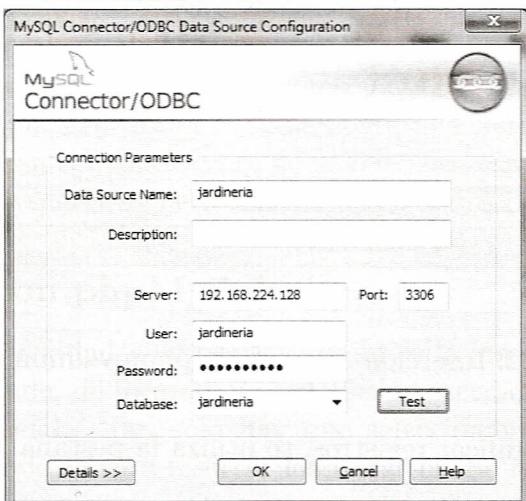


Figura 5.4: Creación de un origen de datos ODBC para MySQL.

Esta referencia se crea en Windows a través del panel de Control, Herramientas Administrativas, y la opción *Orígenes de Datos ODBC*. Cada origen de datos requiere, además de un nombre para el propio origen, el nombre del usuario, la contraseña, el nombre de la base de datos y los datos de conexión al servidor (Dirección IP y puerto TCP/UDP por donde se conecta).

A través de un origen de datos ODBC, se pueden enlazar a Access las tablas de un SGDB para el cual se ha configurado la DSN. Para ello, a través de la pestaña de Access *Datos Externos*, opción *más*, se elige la opción *bases de datos de ODBC* y a continuación se siguen los pasos del asistente. Despues, se elige la opción de vincular al origen de datos creando una tabla vinculada, y finalmente, se selecciona la DSN creada anteriormente desde la pestaña *Origen de datos de equipo*.

-
- ◊ **Actividad 5.1:** Descarga desde la página web de mysql www.mysql.org, apartado **Downloads, connectors**, el conector ODBC para MySQL. Instálalo en tu ordenador y conéctate a la base de datos **jardinería** de la máquina virtual de prácticas disponible en www.garceta.es. Si lo prefieres, conéctate a algún otro servidor que tengas disponible. (Asegúrate, en ese caso, de tener desactivada la opción

bind-address del servidor mysql en el fichero /etc/mysql/my.cnf para poder conectar desde fuera de la propia máquina virtual).

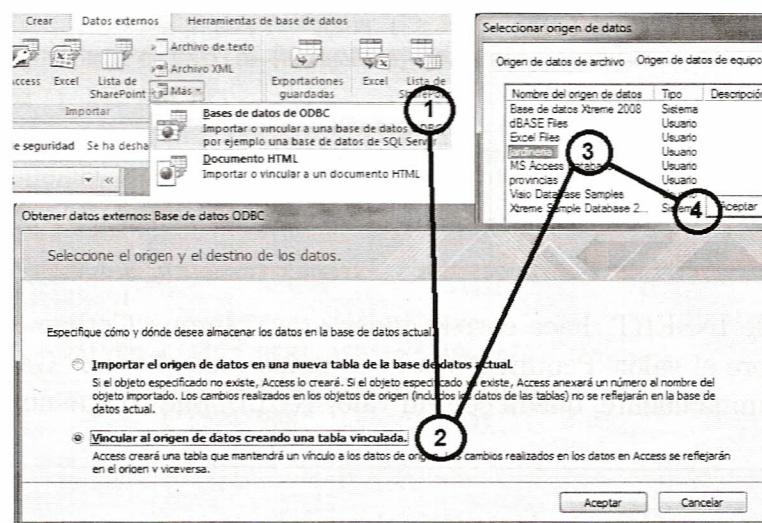


Figura 5.5: Vínculo de MySQL a Access.

5.2. La sentencia INSERT

La sentencia INSERT de SQL permite insertar una fila en una tabla, es decir, añadir un registro de información a una tabla.

El formato de uso es muy sencillo:

```
INSERT [INTO] nombre_tabla [(nombre_columna,...)]
VALUES ({expr | DEFAULT},...)
```



nombre_tabla es el nombre de la tabla donde se quiere insertar la fila. Después del nombre de la tabla, de forma optativa, se pueden indicar las columnas donde se va a insertar la información. Si se especifican las columnas, la lista de valores (VALUES) a insertar se asociará correlativamente con los valores a las columnas indicadas. Si no se especifican las columnas, la lista de valores se escribirá conforme al orden de las columnas en la definición de la tabla. A continuación se muestran unos cuantos ejemplos:

```
desc mascotas;
+-----+-----+-----+
| Field | Type   | Null | Key | Default |
|       |         |      |     |        |
```

codigo	int(11)	NO	PRI	NULL	
nombre	varchar(50)	YES		NULL	
raza	varchar(50)	YES		NULL	
cliente	varchar(9)	YES		NULL	

```
#INSERT especificando la lista de columnas
INSERT INTO mascotas (Codigo, Nombre, Raza)
VALUES
(1,'Pequitas','Gato Común Europeo')
```

Este tipo de INSERT, hace corresponder a la columna Código el valor 1, a la columna Nombre el valor 'Pequitas' y a la columna raza el valor 'Gato Común Europeo'. La columna cliente, queda con un valor NULL, puesto que no se ha indicado un valor.

```
#INSERT sin especificar la lista de columnas.
INSERT INTO mascotas VALUES
(2, 'Calcetines', 'Gato Común Europeo', '59932387L')
```

En este caso, al no especificarse la lista de columnas, hay que indicar todos los valores para todas las columnas en el orden en que están definidas las columnas en la tabla.

```
#INSERT con columnas con valores por defecto
INSERT INTO vehiculos VALUES ('1215 BCD','Toledo TDI', DEFAULT);
```

Aquí, se ha usado el valor DEFAULT para asignar el valor por defecto a la tercera columna de la tabla vehículos, es decir, la columna marca tiene definida la asignación por defecto del valor 'Seat'.

Si se construye una sentencia INSERT con más campos en la lista de valores que el número de columnas especificadas (o número de columnas de la tabla) el SGBD informará del error.

```
#INSERT Errónea
insert into vehiculos (Matricula,Modelo,Marca)
    VALUES ('4123 BFH','Ibiza');
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

5.3. La sentencia INSERT extendida

La sintaxis extendida de INSERT para gestores tipo MySQL es la siguiente:

```
INSERT [INTO] nombre_tabla [(nombre_columna,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

Los puntos suspensivos del final indican que se puede repetir varias veces la lista de valores. Así, MySQL admitiría algo del estilo:

```
insert into vehiculos (Matricula,Modelo,Marca)
VALUES ('4123 BFH','Ibiza','Seat'),
       ('1314 FHD','Toledo','Seat'),
       ('3923 GJS','León','Seat');
```

5.4. INSERT y SELECT

Una variante de la sentencia INSERT consiste en una utilizar la sentencia SELECT para obtener un conjunto de datos y, posteriormente, insertarlos en la tabla.

```
INSERT
[INTO] nombre_tabla [(nombre_columna,...)]
SELECT ... FROM ...
```

Se puede ejecutar la siguiente consulta:

```
#Inserta en una tabla Backup todos los vehículos
INSERT INTO BackupVehiculos
SELECT * FROM vehiculos;
```

La sentencia SELECT debe devolver tantas columnas como columnas tenga la tabla donde se introduce la información. En el ejemplo anterior, la tabla BackupVehiculos tiene una estructura idéntica a la tabla vehículos.

Se puede ver, además, que la sentencia SELECT puede ser tan compleja como se desee, usando filtros, agrupaciones, ordenaciones, etc.

5.5. La sentencia UPDATE

La sentencia UPDATE de SQL permite modificar el contenido de cualquier columna y de cualquier fila de una tabla. Su sintaxis es la siguiente:

```
UPDATE nombre_tabla  
SET nombre_col1=expr1 [, nombre_col2=expr2 ] ...  
[WHERE filtro]
```

La actualización se realiza dando a las columnas nombre_col1, nombre_col2... los valores expr1, expr2,... Se actualizan todas las filas seleccionadas por el filtro indicado mediante la cláusula WHERE. Esta cláusula WHERE, es idéntica a la que se ha utilizado para el filtrado de registros en la sentencia SELECT.

Por ejemplo, si se desea actualizar el equipo de 'Pau Gasol' porque ha fichado por otro equipo, por ejemplo, los 'Knicks', habría que ejecutar la siguiente sentencia:

```
UPDATE jugadores SET Nombre_equipo='Knicks'  
WHERE Nombre='Pau Gasol';
```

Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

El gestor informa de que el filtro seleccionó una fila, y que, por tanto, cambió 1 fila, en este caso, la columna Nombre_equipo de esa fila seleccionada.

Es posible cambiar más de una columna a la vez:

```
UPDATE jugadores SET Nombre_equipo='Knicks', Peso=210  
WHERE Nombre='Pau Gasol';
```

Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Si se omite el filtro, el resultado es la modificación de todos los registros de la tabla, por ejemplo, para cambiar el peso de los jugadores de la NBA de libras a kilos:

```
UPDATE jugadores SET Peso=Peso*0.4535;
```

5.6. La sentencia DELETE

En SQL se utiliza la sentencia DELETE para eliminar filas de una tabla. Su sintaxis es:

```
DELETE FROM nombre_tabla
  [WHERE filtro]
```

El comando **DELETE** borra los registros seleccionados por el filtro **WHERE**, que es idéntico al de la sentencia **SELECT**.

Si se desea borrar al jugador 'Jorge Garbajosa' de la base de datos, habría que escribir la siguiente sentencia:

```
DELETE FROM jugadores WHERE Nombre='Jorge Garbajosa';
Query OK, 1 row affected (0.01 sec)
```

Si se omite el filtro, el resultado es el borrado de todos los registros de la tabla:

```
DELETE FROM jugadores;
Query OK, 432 rows affected (2.42 sec)
```

5.7. La sentencias UPDATE y DELETE con subconsultas

Es posible actualizar o borrar registros de una tabla filtrando a través de una subconsulta. La única limitación es que hay gestores que no permiten realizar cambios en la tabla que se está leyendo a través de la subconsulta.

Por ejemplo, si se desea eliminar los empleados 'Representante Ventas' que no tengan clientes se podría codificar:

```
DELETE FROM Empleados
WHERE CodigoEmpleado Not in
  (SELECT CodigoEmpleadoRepVentas
   FROM Clientes)
AND Puesto='Representante Ventas';
```

No sería posible, sin embargo, escribir una sentencia de este tipo para borrar los clientes con LimiteCredito=0, puesto que se leen datos de la misma tabla que se borran:

```
DELETE FROM Clientes
WHERECodigoCliente in
(SELECTCodigoCliente
FROM Clientes WHERE LimiteCredito=0);
ERROR 1093 (HY000): You can't specify target table 'Clientes' for update
in FROM clause
```

5.8. Borrado y modificación de registros con relaciones

Hay que tener en cuenta que no siempre se pueden borrar o modificar datos: Considérese por ejemplo, que un cliente llama a una empresa pidiendo darse de baja como cliente, pero el cliente tiene algunos pagos pendientes. Si el operador de la BBDD intenta eliminar el registro (DELETE), el SGBD debería informar de que no es posible eliminar ese registro puesto que hay registros relacionados.

O por ejemplo, se desea cambiar (UPDATE) el nombre de un equipo de la NBA (que es su clave primaria), ¿qué sucede con los jugadores? También habrá que cambiar el nombre del equipo de los jugadores, puesto que el campo Nombre_Equipo es una clave foránea.

En este punto, hay que recordar las cláusulas REFERENCES de la sentencia CREATE TABLE para crear las relaciones de clave foránea-clave primaria de alguna columna de una tabla:

definición_referencia:

```
REFERENCES nombre_tabla[(nombre_columna,...)]
[ON DELETE opción_referencia]
[ON UPDATE opción_referencia]
```

opción_referencia:

```
CASCADE | SET NULL | NO ACTION
```

Las cláusulas ON DELETE y ON UPDATE personalizan el comportamiento de estos dos casos. Si por ejemplo, se intenta eliminar un registro con otros registros

relacionados, y se ha seleccionado la opción ON DELETE NO ACTION y ON UPDATE NO ACTION el comportamiento sería el siguiente:

```
#dos tablas relacionadas en mysql
#han de ser innodb para soportar FOREIGN KEYS
CREATE TABLE clientes (
    dni varchar(15) PRIMARY KEY,
    nombre varchar(50),
    direccion varchar(50)
) engine=innodb;

CREATE TABLE pagos_pendientes(
    dni varchar(15),
    importe double,
    FOREIGN KEY(dni) REFERENCES clientes(dni)
        on delete NO ACTION
        on update NO ACTION
) engine=innodb;

#un cliente y dos pagos pendientes
INSERT INTO clientes
VALUES ('55555672L','Pepe Cifuentes','C/Los almendros,23');
INSERT INTO pagos_pendientes VALUES ('55555672L',500);
INSERT INTO pagos_pendientes VALUES ('55555672L',234.5);

#Se intenta borrar el cliente y no es posible
DELETE FROM clientes WHERE dni='55555672L';
ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails ('gestion/pagos_pendientes',
CONSTRAINT 'pagos_pendientes_ibfk_1'
FOREIGN KEY ('dni') REFERENCES 'clientes' ('dni'))

#Se intenta modificar el dni del cliente y no lo permite
UPDATE clientes set dni='55555555L' WHERE dni='55555672L';
ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails ('gestion/pagos_pendientes',
CONSTRAINT 'pagos_pendientes_ibfk_1'
FOREIGN KEY ('dni') REFERENCES 'clientes' ('dni'))

#de igual modo si se intenta borrar la tabla clientes,
#tampoco podemos
DROP TABLE clientes;
ERROR 1217 (23000): Cannot delete or update
a parent row: a foreign key constraint fails
```

Sin embargo, si la creación de la relación estuviese personalizada con las opciones ON UPDATE CASCADE y ON DELETE CASCADE, el comportamiento sería:

```
#dos tablas relacionadas en mysql
create table clientes (
    dni varchar(15) primary key,
    nombre varchar(50),
    direccion varchar(50)
) engine=innodb;
create table pagos_pendientes(
    dni varchar(15),
    importe double,
    foreign key (dni) references clientes(dni)
        on delete CASCADE on update CASCADE
) engine=innodb;

#un cliente y dos pagos pendientes
INSERT INTO clientes
    values ('5555672L','Pepe Cifuentes','C/Los almendros,23');
INSERT INTO pagos_pendientes VALUES ('5555672L',500);
INSERT INTO pagos_pendientes VALUES ('5555672L',234.5);

#se borra el cliente...
DELETE FROM clientes WHERE dni='5555672L';
Query OK, 1 row affected (0.00 sec)

#además, se verifica que ha borrado en cascada sus pagos pendientes.
SELECT * FROM pagos_pendientes;
Empty set (0.00 sec)

#si en lugar de borrar el cliente, se hubiera cambiado el dni:
UPDATE clientes set dni='55555555L' WHERE dni='5555672L';
Query OK, 1 row affected (0.02 sec)

#ha cambiado el dni de los pagos en cascada.
SELECT * FROM pagos_pendientes;
+-----+-----+
| dni      | importe |
+-----+-----+
| 55555555L |      500 |
| 55555555L |    234.5 |
+-----+-----+
```

Recuerda. En Oracle, solo existe la cláusula ON DELETE. Para implementar ON UPDATE, se debe generar un TRIGGER (Véase Capítulo 6)

◊ **Actividad 5.2:** Crea las tablas de mascotas y clientes con la opción ON UPDATE y ON DELETE a SET NULL y comprueba el resultado de ejecutar las inserciones, actualizaciones y borrados de los ejemplos anteriores.

5.9. Transacciones

Un SGBD actualiza múltiples datos a través de una transacción. Una transacción es un conjunto de sentencias SQL que se tratan como una sola instrucción (atómica). Una transacción puede ser confirmada (commit), si todas las operaciones individuales se ejecutaron correctamente, o, abortada (rollback) a la mitad de su ejecución si hubo algún problema (por ejemplo, el producto pedido no está en stock, por tanto no se puede generar el envío). Trabajar con transacciones puede ser esencial para mantener la integridad de los datos. Por ejemplo, se puede dar el caso de que se descuenta el stock de un producto antes de proceder a su envío, pero cuando se va a generar la cabecera del pedido, la aplicación cliente sufre un corte en las comunicaciones y no da tiempo a generarla. Esto supone una pérdida de stock. La transacción garantiza la atomicidad de la operación: O se hacen todas las operaciones, o no se hace ninguna.

Generalmente, cuando se conecta con un cliente a un SGBD, por defecto está activado el modo AUTOCOMMIT=ON, es decir, cada comando SQL que se ejecute, será considerado como una transacción independiente. Para activar las transacciones de múltiples sentencias hay que establecer el modo AUTOCOMMIT=OFF. A partir de ese momento, todos los comandos SQL enviados al SGBD tendrán que terminarse con una orden COMMIT o una orden ROLLBACK. De este modo, se asegura la integridad de los datos a un nivel más alto. Muchos SGBD requieren de una orden START TRANSACTION o START WORK para comenzar una transacción y otros lo hacen de forma implícita al establecer el modo autocommit=off.

```
#MySQL, 3 formas para comenzar una transacción:  
SET AUTOCOMMIT=0; #6  
START TRANSACTION; #6  
BEGIN WORK;  
--Oracle:  
SET AUTOCOMMIT OFF
```

Para terminar una transacción, tanto en MySQL como en ORACLE, hay que aceptar, o rechazar los cambios mediante:

```
#La palabra clave WORK es opcional  
COMMIT WORK; #Acepta los cambios.  
ROLLBACK WORK; #Cancela los cambios
```

Cualquier conjunto de sentencias SQL se considera cancelado si termina abruptamente la sesión de un usuario sin hacer COMMIT. Un ejemplo de transacción en MySQL sería la siguiente:

```
SET AUTOCOMMIT 0;  
#se actualiza el stock  
UPDATE Productos SET Stock=Stock-2 WHERE CodigoProducto='AAAF102';  
#se inserta la cabecera del pedido  
INSERT INTO Pedidos VALUES  
    (25,now(),'Francisco Garcia','Pendiente de Entrega');  
#se inserta el detalle del pedido  
INSERT INTO DetallePedidos  
(CodigoPedido,CodigoProducto,Unidades) VALUES  
    (25,'AAAF102',2);  
#aceptar transacción  
COMMIT WORK;
```

5.10. Acceso concurrente a los datos

Cuando se utilizan transacciones, pueden suceder problemas de concurrencia en el acceso a los datos, es decir, problemas ocasionados por el acceso al mismo dato de dos transacciones distintas. Estos problemas están descritos por SQL estándar y son los siguientes:

Dirty Read (Lectura Sucia). Una transacción lee datos escritos por una transacción que no ha hecho COMMIT.

Nonrepeatable Read (Lectura No Repetible). Una transacción vuelve a leer datos que leyó previamente y encuentra que han sido modificados por otra transacción.

Phantom Read (Lectura Fantasma). Una transacción lee unos datos que no existían cuando se inició la transacción.

Cuando se trabaja con transacciones, el SGBD puede bloquear conjuntos de datos para evitar o permitir que sucedan estos problemas. Según el nivel de concurrencia

que se desee, es posible solicitar al SGBD cuatro niveles de aislamiento. Un nivel de aislamiento define cómo los cambios hechos por una transacción son visibles a otras transacciones:

Read Uncommitted (Lectura no acometida). Permite que sucedan los tres problemas. Las sentencias SELECT son efectuadas sin realizar bloqueos, por tanto, todos los cambios hechos por una transacción pueden verlos las otras transacciones.

Read Committed (Lectura acometida). Los datos leídos por una transacción pueden ser modificados por otras transacciones. Se pueden dar los problemas de Phantom Read y Non Repeatable Read.

Repeatable Read (Lectura Repetible). Tan solo se permite el problema del Phantom Read. Consiste en que ningún registro leído con un SELECT se puede cambiar en otra transacción.

Serializable. Las transacciones ocurren de forma totalmente aislada a otras transacciones. Se bloquean las transacciones de tal manera que ocurren unas detrás de otras, sin capacidad de concurrencia. El SGBD las ejecuta concurrentemente si puede asegurar que no hay conflicto con el acceso a los datos.

En MySQL, las tablas innodb tienen el nivel de aislamiento por defecto establecido en REPETEABLE READ, y se puede alterar cambiándolo en el fichero de configuración my.cnf o ejecutando:

```
SET TRANSACTION ISOLATION
LEVEL {READ UNCOMMITTED | READ COMMITTED
        | REPEATABLE READ | SERIALIZABLE}
```

En Oracle, el nivel por defecto es *READ COMMITTED* y, además de este, solo permite *SERIALIZABLE*. Se puede cambiar ejecutando el comando:

```
SET TRANSACTION ISOLATION LEVEL {READ COMMITTED|SERIALIZABLE};
```

¿Sabías que . . . ? Algunas transacciones pueden quedar interbloqueadas dependiendo del nivel de aislamiento seleccionado. Esta situación de *interbloqueo*, o *deadlock* entre dos transacciones, consiste en que ninguna de ellas puede seguir ejecutándose porque la primera intenta acceder a un bloque de datos que tiene bloqueada la segunda, y la segunda, intenta acceder a un bloque de datos que tiene bloqueada la primera. En esta situación de interbloqueo, el SGBD elimina a una de las dos transacciones, siendo la *victima* del interbloqueo la que hace rollback de sus operaciones.

5.10.1. Ejemplo de problemas en el acceso concurrente

A continuación se muestran, mediante una serie de consultas en MySQL cómo reproducir los problemas típicos de la concurrencia en bases de datos. Primero, para consultar el tipo de aislamiento y cambiarlo a *read uncommitted*, que permite que se produzcan los tres problemas, se puede ejecutar los comandos:

```
#consulta tipo de aislamiento
mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
1 row in set (0.00 sec)

#establecer el tipo de aislamiento a read uncommitted
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00 sec)
```

Para el siguiente ejemplo se utilizará la tabla *coches* creada con el motor innodb:

```
create table coches (nombre varchar(20)) engine=innodb;
insert into coches values ('toyota'),('audi'),('seat');
```

La lectura no repetible

A continuación, dos sesiones distintas, de dos usuarios distintos, consultan la base de datos en el siguiente orden para producir una lectura no repetible:

```
Lectura no repetible-1

#sesión 1
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from coches;
+-----+
| nombre |
+-----+
| toyota |
| audi   |
| seat   |
+-----+
3 rows in set (0.00 sec)
```

La sesión 1 inició una transacción para hacer operaciones y segundos después una segunda sesión realiza cambios en la tabla mientras la transacción continúa en la sesión 1:

Lectura no repetible-2

```
#sesión 2
mysql> update coches set nombre='volvo' where nombre='audi';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

La sesión 2 ha realizado un cambio en los datos, por tanto, la transacción de la primera sesión no será capaz de reproducir la situación anterior puesto que se ha producido una *lectura no repetible*:

Lectura no repetible-3

```
#sesión 1
mysql> select * from coches;
+-----+
| nombre |
+-----+
| toyota |
| volvo  |
| seat   |
+-----+
3 rows in set (0.00 sec)
```

La lectura fantasma

Si a continuación, el cliente 2 inserta un nuevo registro en la tabla, la sesión 1 podrá inmediatamente ver los cambios. Esto es lo que se llama una *lectura fantasma*:

Lectura fantasma-1

```
#cliente 2
mysql> insert into coches values ('renault');
Query OK, 1 row affected (0.00 sec)
```

Lectura fantasma-2

```
#cliente 1
mysql> select * from coches;
+-----+
| nombre |
+-----+
| toyota |
| volvo  |
| seat   |
| renault |
+-----+
4 rows in set (0.00 sec)
```

La potencial situación problemática tanto de la lectura no repetible como de la lectura fantasma es que datos que participan en una transacción cambian o aparecen sin previo aviso, pudiendo provocar que la transacción genere información que no es del todo íntegra.

La lectura sucia

El último problema, y más grave, es el de la lectura sucia. A continuación, siguiendo con las dos sesiones de los ejemplos anteriores, se reproduce esta situación. En este caso, la sesión 2 inicia una transacción y realiza operaciones DML sobre los datos:

Lectura sucia-1

```
#sesión 2
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into coches values ('alfa');
Query OK, 1 row affected (0.00 sec)

mysql> update coches set nombre='honda' where nombre='toyota';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Después, la sesión 1 lee los datos que hay en la tabla:

Lectura sucia-2

```
#sesión 1
mysql> select * from coches;
+-----+
| nombre |
+-----+
| honda  |
| volvo  |
| seat   |
| renault|
| alfa   |
+-----+
5 rows in set (0.00 sec)
```

Se puede observar, que gracias a esa lectura la sesión 1 puede generar nueva información o realizar algún trabajo que implique trabajar con la información modificada por la sesión 2. Pero, supóngase que ahora, la sesión 2 realiza un *rollback* de sus operaciones:

Lectura sucia-3

```
#sesión 2
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)
```

Se puede observar que la sesión 1, aunque ya no puede acceder a la información que fue insertada por la sesión 2, puesto que se ha efectuado un rollback, puede haber estado trabajando con datos, que, en realidad, nunca han estado en la base de datos. Esto, es una *lectura sucia*:

Lectura sucia-4

```
#sesión 1
mysql> select * from coches;
+-----+
| nombre |
+-----+
| honda  |
| volvo  |
| seat   |
| renault|
| alfa   |
+-----+
5 rows in set (0.00 sec)
```

5.11. El acceso a la información

Cuando se administra la seguridad en el acceso a 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 (nativa). En la seguridad integrada, se suele contar con los usuarios de un sistema de dominio o un servicio de directorio (LDAP) para proporcionar el acceso a determinados recursos del gestor de base de datos. 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 bases de datos.

El alcance de este libro es tratar la seguridad nativa de un SGBD a través de SQL explicando el funcionamiento básico de la seguridad en MySQL, Oracle y DB2.

5.12. Las vistas

Una vista **es una tabla sin contenido, totalmente virtual, que devuelve las filas resultado de ejecutar una consulta SQL. La diferencia con una consulta ejecutada directamente es que, mientras cada sentencia SQL enviada al SGBD tiene que pasar**

por un proceso de compilación, la vista es una consulta cuya definición ha sido almacenada previamente y que ya ha sido compilada, siendo por tanto el tiempo de ejecución bastante menor. También tiene una implicación importante en el hecho de que un usuario podría no tener acceso a la información de varias tablas y, sin embargo, sí tener acceso a la vista que consulta esas tablas, proporcionando de esta manera un acceso controlado solo a determinadas filas y columnas de esas tablas.

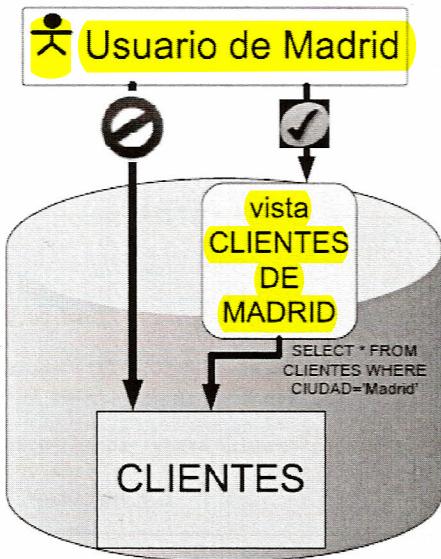


Figura 5.6: Vistas.

Por ejemplo, en una tabla de clientes, un usuario de una oficina de Madrid podría tener solo acceso a la información de los clientes de Madrid, y tan solo a ciertos campos. De esta manera, no tendría acceso a ningún campo de la tabla de clientes y, sin embargo, podría tener acceso a una vista que consulte aquellos clientes cuya provincia sea Madrid.

La sintaxis para crear una vista es la siguiente:

```
CREATE [OR REPLACE] VIEW [esquema.]nombre_vista [(lista_columnas)] AS  
sentencia_select
```



La ejecución del **CREATE VIEW** provoca que se compile la sentencia select y que se almacene con el nombre nombre_vista. Los nombres de las columnas de la vista se pueden especificar mediante lista_columnas. Si se especifica la lista de columnas, cada columna tendrá el alias correspondiente, si no, se obtendrá el nombre devuelto por la consulta. Si la vista ya existe, se puede reemplazar con **OR REPLACE**.

Un ejemplo de CREATE VIEW es el siguiente:

```
CREATE VIEW nba.jugadoresMiami AS  
    SELECT Nombre, Posicion FROM nba.jugadores WHERE Nombre_equipo='HEAT';  
SELECT * from nba.jugadoresMiami;
```

Además, se pueden crear vistas para que los usuarios no expertos puedan acceder de forma fácil a la información, proporcionándoles, a través de una vista, información obtenida a través de una sentencia SQL compleja:

```
CREATE VIEW VistaPedidos (CodigoPedido,Cliente,Total) AS  
    SELECT CodigoPedido, NombreCliente, SUM(Cantidad*PrecioUnidad)  
    FROM Clientes NATURAL JOIN Pedidos NATURAL JOIN DetallePedidos  
    GROUP BY CodigoPedido;
```

Para eliminar una vista se hace uso del comando DROP VIEW:

```
DROP VIEW [esquema.]nombre_vista;
```

Hay pequeñas variaciones en la sintaxis de los comandos CREATE VIEW y DROP VIEW dependiendo del SGBD que se utilice. Además, se dispone también de un comando ALTER VIEW para hacer modificaciones a la definición de la vista. Para más información sobre estas variaciones, consultar los manuales de cada gestor.

5.13. Los usuarios

Para crear cuentas de usuario que permitan a los usuarios acceder a ciertos objetos con un nivel determinado de privilegios hay que hacer uso del comando CREATE USER.

```
CREATE USER nombre_usuario IDENTIFIED BY 'password' [opciones];
```

Esta sencilla sentencia crea una cuenta usuario que permite la autenticación de un usuario en el SGBD a través de la password identificada mediante la opción IDENTIFIED BY. Esta sintaxis de create user es válida tanto para MySQL como

para Oracle, aunque a Oracle se le pueden incluir multitud de opciones extras para dar características adicionales, como la asignación de cuota para añadir información a un tablespace o bloquear la cuenta temporalmente. En DB2 no se utiliza el comando CREATE USER puesto que la mayor parte de la **gestión de usuarios** se hace de forma integrada con el Sistema Operativo. A continuación se muestran ejemplos de creación de usuarios:

```
#creación de usuario en MySQL
CREATE USER paco IDENTIFIED BY 'o99238kjkA';

--creación de usuario en Oracle
CREATE USER paco
    IDENTIFIED BY 'o99238kjkA'
    DEFAULT TABLESPACE 'Nominas'
    QUOTA UNLIMITED ON 'Nominas'
    ACCOUNT LOCK;
```

-
- ◊ **Actividad 5.3:** Crea los usuarios Pedro y Javier en los SGBD mysql y Oracle con passwords apropiadas. En Oracle, asigna quota de 1MB en algún tablespace.
-

Para eliminar usuarios, se puede utilizar la sentencia DROP USER:

```
DROP USER nombre_usuario [CASCADE];
```

En Oracle, se puede incluir el token CASCADE para indicar que junto con el usuario se borren todos los objetos de su esquema. En MySQL, se pueden borrar a la vez varios usuarios, separando los nombres de los usuarios mediante comas.

Para modificar usuarios, Oracle utiliza el comando ALTER USER, que no está disponible en MySQL. Este comando ALTER USER usa las mismas opciones que CREATE USER.

```
--Modificación de usuario en Oracle
ALTER USER Paco
    IDENTIFIED BY 'nueva_pass' DEFAULT Tablespace 'Facturas';
```

En MySQL hay que modificar los usuarios actualizando sus datos en la propia tabla mysql.user del sistema. Por ejemplo, para cambiar el host desde el que el

usuario Javier se puede conectar, habría que ejecutar la siguiente sentencia. Cuando en MySQL se modifica algún permiso modificando el contenido de las tablas del sistema, hay que ejecutar además el comando FLUSH PRIVILEGES para forzar al gestor a volver leer las tablas de permisos y que los cambios en los permisos sean efectivos desde ese momento.

```
#Modificación de usuario en mysql.
UPDATE mysql.user
SET host='192.168.3.1' where user='JAVIER';
FLUSH PRIVILEGES;
```

En MySQL también se puede renombrar un usuario conservando todos sus privilegios utilizando el comando RENAME USER y cambiar la password mediante el comando SET PASSWORD:

```
#Modificación de usuarios en MySQL. Cambiar nombre y password al usuario Paco
RENAME USER Paco@localhost to PacoSanchez@localhost;
SET PASSWORD for Paco@localhost = PASSWORD('nueva_pass');
```



◊ **Actividad 5.4:** Ejecuta el siguiente comando para cambiar la password de un usuario en mysql:

```
UPDATE mysql.user
SET Password = PASSWORD("nueva_pwd")
WHERE user='paco' AND host='localhost';
```

◊ **Actividad 5.5:** Cambia la password de los usuarios Pedro y Javier creados en Oracle y bloquea sus cuentas mediante la opción ACCOUNT UNLOCK.

5.14. Los privilegios

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



comando REVOKE. Estos comandos varian en su sintaxis dependiendo del SGBD que se está usando puesto que el sistema de seguridad de cada uno es distinto. Además, influye también la forma en que el subsistema de permisos del SGBD se integra con el sistema operativo.

5.14.1. El sistema de privilegios de MySQL

La sintaxis del comando GRANT para MySQL es la siguiente:

```
GRANT tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
ON {nombre_tabla | * | *.* | base_datos.* | base_datos.nombre_tabla}
    TO usuario [IDENTIFIED BY [PASSWORD] 'password']
    [, usuario [IDENTIFIED BY [PASSWORD] 'password']] ...
    [WITH opcion [opcion] ...]

opcion =
    GRANT OPTION
    | MAX_QUERIES_PER_HOUR count
    | MAX_UPDATES_PER_HOUR count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS count
```



En MySQL se puede otorgar a un usuario permisos para hacer cualquier operación a nivel de host, de base de datos, de tabla o de columna. Así, es posible asignar, por ejemplo, permisos de SELECT sobre las columnas NombreCliente, Dirección y Telefono de la tabla clientes:

```
GRANT SELECT (NombreCliente,Telefono,Ciudad)
ON Clientes TO paco@localhost;
```

Con esta sentencia el usuario *paco@localhost* solo podrá seleccionar las columnas *NombreCliente*, *Telefono* y *Ciudad* de la tabla clientes, siéndole denegada una consulta del tipo *select * from clientes*.

tipo_privilegio es la clase de permiso que puede ser otorgado, típicamente pueden ser *select*, *insert*, *update*, ... Pueden ser de los más variados. En el Cuadro 5.3 de la página 210 hay algunos ejemplos de los permisos disponibles en MySQL.

Estos tipos de privilegio se pueden aplicar a las siguientes expresiones:

Expresión	Se aplica el permiso a
nombre_tabla	La tabla nombre_tabla
*	Todas las tablas de la base de datos que se está usando
**	Todas las tablas de todas las bases de datos
base_datos.*	Todas las tablas de la base de datos db_name
base_datos.nombre_tabla	Solo la tabla nombre_tabla de la base de datos db_name

Cuadro 5.1: Tipos de objetos a los que se puede otorgar permisos.

Finalmente con 'TO usuario' se indica el usuario al que se quiere otorgar el permiso. Si el usuario user no existe, se crea, opcionalmente con la password indicada mediante la cláusula identified by.

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

Expresión	Función
GRANT OPTION	Permite conceder a otros usuarios los permisos que tiene el usuario, por tanto, el administrador debe ser muy cauto a la hora de conceder esta opción a los usuarios de la base de datos.
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 (logins) 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 el valor 0, significa ilimitado.

Cuadro 5.2: Opciones adicionales a los permisos.

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	Crea rutinas almacenadas
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLE
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER, y REVOKE.
CREATE VIEW	Permite el uso de CREATE VIEW
DELETE	Permite el uso de DELETE
DROP	Permite el uso de DROP TABLE
EXECUTE	Permite al usuario ejecutar rutinas almacenadas
FILE	Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE
INDEX	Permite el uso de CREATE INDEX y DROP INDEX
INSERT	Permite el uso de INSERT
LOCK TABLES	Permite el uso de LOCK TABLES en tablas para las que tenga el permiso SELECT
PROCESS	Permite el uso de SHOW FULL PROCESSLIST
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	SHOW DATABASES muestra todas las bases de datos
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 al gestor
GRANT OPTION	Permite dar permisos

Cuadro 5.3: Tipos de privilegios en mysql.

Algunos ejemplos de consultas para asignación de permisos en MySQL son los siguientes:

```
#Otorga permisos de select e insert a todas las tablas de nba
GRANT SELECT, INSERT on nba.* TO paco@localhost;

#Otorga todos los privilegios a la tabla Clientes de jardineria
GRANT ALL PRIVILEGES on jardineria.Clientes TO paco@localhost;

#Otorga permisos de select a todas las tablas de todas las bases de datos
#permitiendo al usuario ceder esos permisos a otros usuarios
GRANT SELECT on *.* to paco@localhost WITH GRANT OPTION;

#Otorga permisos de SELECT, INSERT, UPDATE y DELETE con un límite de 10
#consultas a la hora en la tabla jugadores de la nba
GRANT SELECT,INSERT,UPDATE,DELETE on nba.jugadores to paco@localhost
WITH MAX_QUERIES_PER_HOUR 10 MAX_UPDATES_PER_HOUR 10
```



La sentencia **REVOKE** deniega permisos a un usuario sobre un objeto. A continuación se describe la sintaxis:

```
REVOKE tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
ON {nombre_tabla | * | *.* | base_datos.* | base_datos.nombre_tabla}
FROM usuario [, usuario] ...
```

La sintaxis es muy parecida a la de la sentencia grant, a continuación se muestran unos cuantos ejemplos:

```
#quita el permiso de select en la tabla jardineria.Cliente
revoke select on jardineria.Clientes from paco@localhost;

#elimina el permiso ALL PRIVILEGES de todas las tablas
revoke all privileges on *.* from paco@localhost;

#quita los permisos de select e insert de todas las tablas de jardineria
revoke select,insert on jardineria.* from paco@localhost;
```

Otra forma de asignar y eliminar permisos en MySQL es utilizando las tablas del catálogo de metadatos, es decir, las tablas de la base de datos *mysql* creada en toda instalación del SGBD y que almacena toda la información sobre todos los

objetos manejados por el gestor. En esta base de datos existen 5 tablas relacionadas con el sistema de permisos de MySQL, user, db, host, tables_priv y columns_priv. Estas tablas se pueden manipular manualmente con inserts y deletes para otorgar y denegar permisos a nivel de usuario, base de datos, equipo, tablas y columnas respectivamente.

A continuación se ilustra cómo MySQL procesa una consulta. Al recibir la instrucción SQL, se comprueba si el acceso a los diversos objetos están autorizados en cualquiera de estas tablas. Si ninguno de los niveles permite el acceso, la consulta es denegada por falta de permisos.

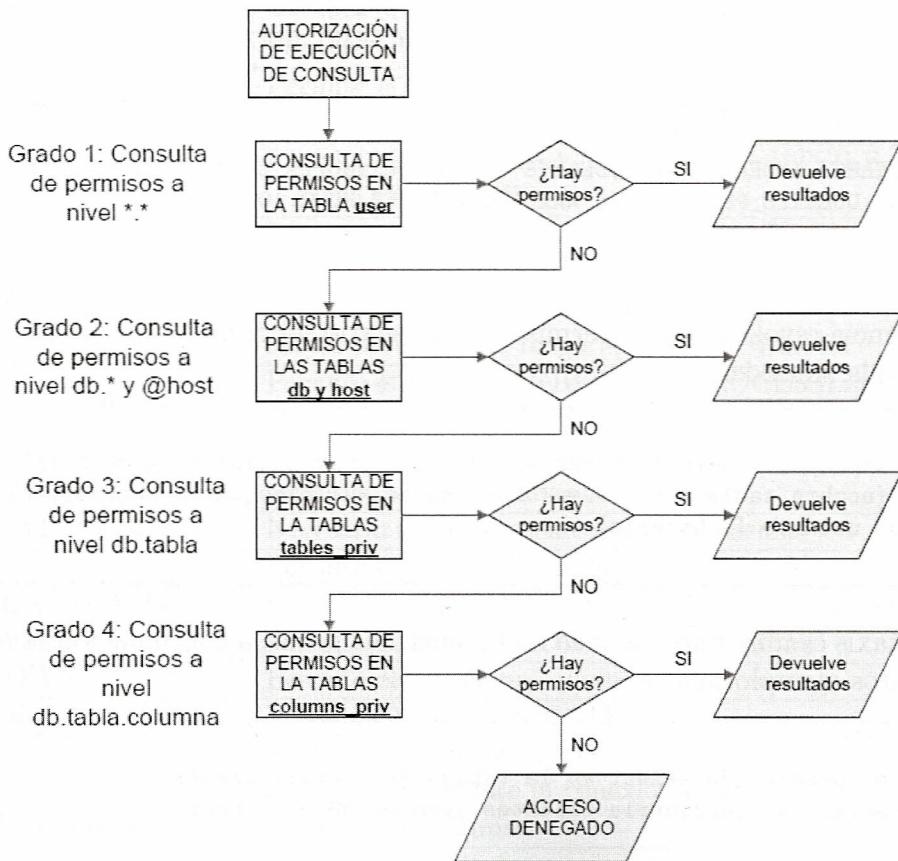


Figura 5.7: Estructura de permisos en mysql.

5.14.2. El sistema de privilegios de Oracle

El sistema de privilegios de Oracle clasifica los permisos en dos tipos, privilegios del sistema y privilegios de objetos. Además define los siguientes conceptos:

- **ROLES** Conjunto de privilegios que se pueden asignar a un determinado usuario. Un usuario puede pertenecer a múltiples roles.

- **PERFILES** Un perfil es un conjunto de restricciones sobre el uso de recursos. Cada usuario puede pertenecer a un único perfil.

De esta manera, Oracle permite asignar a un usuario privilegios del sistema, de objetos, roles y perfiles, todo ello a través de la sentencia GRANT:

```
GRANT privilegio [ON [esquema.]objeto] TO {usuario | rol | PUBLIC}
    [WITH {GRANT | ADMIN} OPTION]
privilegio =
    tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
    | privilegio_sistema
    | rol
```

El privilegio a otorgar puede ser de varios tipos:

- Un privilegio sobre un objeto a un usuario o rol. Si el privilegio es INSERT o UPDATE se puede indicar sobre qué columnas se tiene el permiso. En este caso, se debe incluir la cláusula ON para indicar sobre qué objeto se va a aplicar el permiso.

```
-- permiso de select y update en dos columnas de la tabla Clientes
GRANT SELECT, UPDATE (NombreCliente, Telefono)
ON Jardineria.Clientes To Javier;
-- todos los permisos sobre Pedidos al rol ADMINISTRACION
GRANT ALL PRIVILEGES ON Jardineria.Pedidos to ADMINISTRACION;
```

- Un privilegio de sistema sobre un usuario o rol.

```
--permiso para crear tablas
GRANT CREATE TABLE to Javier;
--permiso para crear tablas al rol ADMINISTRACION
GRANT CREATE TABLE to ADMINISTRACION;
```

- Un rol sobre un usuario u otro rol. El rol puede estar predefinido (rol de sistema) o puede ser definido por el usuario.

```
--asignar Rol de acceso y consumo a otro Rol
GRANT CONNECT, RESOURCE to ADMINISTRACION;
```

```
--asignar el rol ADMINISTRACION a Javier
GRANT ADMINISTRACION to Javier;
```

- Un privilegio de objeto o sistema a todo el mundo (PUBLIC).

```
--permiso de acceso y consumo de recurso a todos
GRANT CONNECT, RESOURCE to PUBLIC;
--permiso de update sobre las columnas Nombre y email de empleados
GRANT UPDATE (Nombre, email) ON Jardineria.Empleados to Public;
```

Privilegio	Significado
ALL [PRIVILEGES]	Da todos los permisos simples incluido GRANT OPTION
DELETE	Permite el uso de DELETE
INSERT	Permite el uso de INSERT
SELECT	Permite el uso de SELECT
UPDATE	Permite el uso de UPDATE
REFERENCES	Permite crear claves foráneas
EXECUTE	Permite ejecutar un procedimiento, función, paquete...
ALTER	Permite cambiar la definición de un objeto
INDEX	Permite crear índices de una tabla

Cuadro 5.4: Tipos de privilegios de objeto en Oracle.

Privilegio	Significado
ALTER DATABASE	Permite ejecutar el comando ALTER DATABASE
ALTER SYSTEM	Permite el uso de ALTER SYSTEM
AUDIT SYSTEM	Permite la auditoría mediante la sentencia AUDIT
CREATE [ANY] objeto	Permite el uso de CREATE para un tipo de objeto (TABLE, VIEW, PROCEDURE, USER, PROFILE, etc...)
ALTER [ANY] objeto	Permite el uso de ALTER para un tipo de objeto (TABLE, VIEW, PROCEDURE, USER, PROFILE, etc...)
SYSDBA	Permite ejecutar operaciones de parada y puesta en marcha de la BBDD
ANALYZE [ANY] objeto	Permite analizar un tipo de objeto

Cuadro 5.5: Tipos de privilegios de sistema en Oracle.

Las cláusulas WITH GRANT OPTION o WITH ADMIN OPTION transmiten la autorización de conceder ese privilegio de objeto o de sistema a otro usuario o rol. Para denegar privilegios sobre un objeto Oracle utiliza la sentencia *REVOKE*

```
REVOKE privilegio  
[ON [esquema.]objeto]  
FROM {usuario | rol | PUBLIC}  
  
privilegio =  
    tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...  
    | privilegio_sistema  
    | rol
```

Algunos ejemplos de esta sentencia son los siguientes:

```
--Denegar permiso de select en Clientes al rol ADMINISTRACION  
REVOKE SELECT ON Jardineria.Clientes FROM ADMINISTRACION;  
--Denegar rol CONNECT a todos  
REVOKE CONNECT FROM PUBLIC;  
--Denegar los roles ALMACEN Y CONTABILIDAD A Javier;  
REVOKE ALMACEN,CONTABILIDAD FROM Javier;  
--Denegar todos los privilegios a Javier en la tabla Pedidos  
REVOKE ALL PRIVILEGES ON Pedidos FROM Javier;
```

Cuando se elimina un privilegio de un usuario, el SGBD elimina el permiso de ese usuario de forma inmediata a él y a todos los usuarios a los que se les haya concedido el privilegio con la opción WITH GRANT o WITH ADMIN.

Los roles

Para simplificar la tarea de organizar los permisos, Oracle permite la creación de Roles con la sentencia CREATE ROLE:

```
CREATE ROLE nombre_rol [opciones_identificacion];  
  
opciones_identificacion:  
    NOT IDENTIFIED  
    | IDENTIFIED  
        { BY password | EXTERNALLY | GLOBALLY | USING [esquema].package}
```

Al crear un rol, se puede especificar el tipo de autentificación. Si se especifica NOT IDENTIFIED, el rol está autorizado por la base de datos y no es necesaria ninguna contraseña para activarlo, sin embargo si se especifica IDENTIFIED, el

usuario debe ser autorizado antes de activar el rol. Dentro de IDENTIFIED existen varias posibilidades:

- BY password, donde el usuario debe indicar la contraseña antes de activar el papel.
- EXTERNALLY, para ser autorizado mediante un servicio externo (como el sistema operativo).
- GLOBALLY, que cede la autorización al servicio Oracle Security Service.
- Si se elige USING, se elige crear un rol de aplicación segura, es decir, un rol que solo puede ser activado por las aplicaciones que usen el paquete.

Cuando un usuario se conecta al SGBD, este otorga al usuario los permisos que se le hayan concedido explícitamente mediante la orden GRANT y los permisos que tengan los roles por defecto del usuario. Para asignar un rol por defecto a un usuario se utiliza la sentencia ALTER USER de la siguiente forma:

```
ALTER USER usuario DEFAULT ROLE [opciones_rol];
opciones_rol:
    rol [, rol] ...
    | ALL [EXCEPT rol]
    | NONE
```

Se puede anular el rol por defecto mediante la opción NONE, o habilitar todos los roles mediante ALL, o todos menos uno mediante ALL EXCEPT rol, o conceder varios separados por coma. Por ejemplo, para asignar los roles ADMINISTRACION y CONTABILIDAD al usuario Javier:

```
GRANT Contabilidad,Administracion to Javier;
ALTER USER Javier DEFAULT ROLE ADMINISTRACION,CONTABILIDAD;
```

Finalmente, se puede utilizar también la sentencia SET ROLE para dotar momentáneamente de ciertos privilegios a un usuario que tenga autorizado su uso. Por ejemplo, se puede crear el rol ALMACEN, con todos los privilegios sobre la tabla Productos y asignarselo al usuario Javier. Si el usuario Javier no activa el rol ALMACEN mediante SET ROLE y su contraseña, no podrá disfrutar de sus privilegios

```
CREATE ROLE ALMACEN IDENTIFIED BY AlmacenPwd;
GRANT ALL Privileges ON Jardineria.Productos TO ALMACEN;
GRANT ALMACEN to Javier;

--conexión con el usuario Javier
SQL> select count(*) from Jardineria.productos;
select count(*) from Jardineria.productos
*
ERROR en línea 1:
ORA-00942: la tabla o vista no existe

SQL> SET ROLE ALMACEN identified by AlmacenPwd;
Rol definido.

SQL> select count(*) from Jardineria.productos;
  COUNT(*)
-----
      276
```

Creación de perfiles

Los perfiles se usan para limitar la cantidad de recursos del sistema o de la base de datos disponibles para un usuario. Existe un perfil llamado DEFAULT que concede recursos ilimitados a todos los usuarios. Para evitar esto, se puede hacer uso de la sentencia CREATE PROFILE para crear nuevos perfiles.

```
CREATE PROFILE perfil LIMIT recursos ...
recursos =
    recurso { cantidad | UNLIMITED | DEFAULT }
    | password
recurso =
    SESSIONS_PER_USER
    | CPU_PER_SESSION
    | CPU_PER_CALL
    | CONNECT_TIME
    | IDLE_TIME
    | LOGICAL_READS_PER_SESSION
    | LOGICAL_READS_PER_CALL
    | COMPOSITE_LIMIT
    | PRIVATE_SGA
password =
    parametro { cantidad | UNLIMITED | DEFAULT }
    | PASSWORD_VERIFY_FUNCTION { función | NULL | DEFAULT }
```

```
parametro =  
    FAILED_LOGIN_ATTEMPTS  
    | PASSWORD_LIFE_TIME  
    | PASSWORD_REUSE_TIME  
    | PASSWORD_REUSE_MAX  
    | PASSWORD_LOCK_TIME  
    | PASSWORD_GRACE_TIME
```

Los diferentes límites que se pueden utilizar están definidos en la siguiente tabla:

Límite	Función
SESSIONS_PER_USER	Número máximo de sesiones abiertas a la vez por un usuario
CPU_PER_SESSION	Tiempo límite para una sesión expresado en centésimas de segundos
CPU_PER_CALL	Tiempo límite de respuesta para una llamada (parse, fetch o ejecución) en centésimas de segundos
CONNECT_TIME	Tiempo límite total de una sesión, expresado en minutos
IDLE_TIME	Tiempo máximo de inactividad de una sesión expresados en minutos
LOGICAL_READS_PER_SESSION	Número máximo de bloques de datos leidos en una sesión
LOGICAL_READS_PER_CALL	Número máximo de bloques de datos leidos en una llamada
PRIVATE_SGA	Tamaño de memoria que una sesión puede reservar del espacio privado de memoria del SGA
COMPOSITE_LIMIT	Coste total en recursos de una sesión, expresado mediante el promedio de ciertos parámetros

Cuadro 5.6: Tipos de recursos.

Por ejemplo, se puede crear el perfil OPERADOR con los siguientes límites:

```
CREATE PROFILE OPERADOR LIMIT  
    SESSIONS_PER_USER 5      -- 5 sesiones concurrentes  
    CPU_PER_SESSION UNLIMITED --Tiempo de CPU ilimitado  
    IDLE_TIME 10            -- 10 minutos de inactividad
```

```
CONNECT_TIME 120      -- 2 horas totales de conexión  
;
```

También es posible alterar un perfil mediante la sentencia ALTER PROFILE:

```
--modificar el perfil DEFAULT para que la password no caduque  
alter profile DEFAULT LIMIT PASSWORD\_LIFE\_TIME Unlimited;  
  
--modificar el perfil OPERADOR para que su límite de sesión sea de 1 hora  
-- y un máximo de 10 sesiones concurrentes  
alter profile OPERADOR LIMIT CONNECT_TIME 60 SESSIONS_PER_USER 10;
```

Un perfil se puede asignar en el momento de crear o modificar el usuario, por ejemplo:

```
-- creación de un usuario con el perfil OPERADOR  
CREATE USER Francisco IDENTIFIED BY FrancisPwd  
    QUOTA UNLIMITED ON Users  
    PROFILE OPERADOR;
```

5.14.3. El sistema de privilegios de DB2

Desde el punto de vista de la seguridad, se puede decir que en DB2 existen tres grandes puntos de gestión de los privilegios de acceso:

1. Autenticación
2. Autorización
3. Privilegio

La autenticación busca responder a las siguientes preguntas: ¿quién puede acceder a la instancia o base de datos?, ¿dónde se verificará la password de usuario? En DB2 los usuarios de autenticación deben estar dados de alta en el sistema operativo, es decir, que para resolver la primera pregunta, bastaría con decir que solo los usuarios del sistema pueden acceder a la instancia o base de datos, pero jojo!, los usuarios pueden serlo del sistema local o de un sistema cliente remoto que esté intentando la conexión, esto sería así en función de cómo se defina la variada política de autenticación que dará respuesta a la segunda pregunta. La política de autenticación no solo permite o no la conexión desde clientes, sino que, entre otras cosas, indica si habrá encriptación o no del usuario y password, e incluso de los datos intercambiados durante la comunicación con el servidor.

La autorización hace referencia al nivel de permisos con que un usuario materializa el acceso a la instancia o base de datos. También determina los comandos que el usuario puede ejecutar, los datos que puede leer o modificar, o los tipos de objetos de base de datos que se le permite crear, modificar o borrar. Todas estas consideraciones están preprogramadas en siete grupos de autorizaciones que pueden ser concedidos, en función del caso, a un grupo de usuarios o a un usuario determinado.

- SYSADM: es el único al que se le permite modificar los parámetros de la instancia. Comparable a la autoridad del usuario *root* en Unix, puede ejecutar cualquier comando contra la instancia, contra todas las bases de datos de la instancia y contra todos los objetos de cualquier base de datos.
- SYSCTRL: puede hacer todas las actividades de administración (crear y borrar bases de datos y tablespaces) y mantenimiento (hacer backups, pasar estadísticas, actualizar parámetros de la base de datos), pero no podrá acceder a ningún dato de las bases de datos a no ser que le sean concedidos explícitamente los privilegios adecuados.
- SYSMAINT: solo puede ejecutar labores de mantenimiento (hacer backups, pasar estadísticas, actualizar parámetros de la base de datos).
- SYSMON: pueden obtener fotos estáticas de monitorización (en inglés *database system monitor snapshots*) de la instancia o sus bases de datos.
- DBADM: solo existe en el contexto de una base de datos concreta. Le está permitido crear y borrar tablas, ejecutar estadísticas, y conceder o quitar privilegios, pero en ningún caso podrá borrar la base de datos, ni crear tablespaces, ni hacer backups, o modificar parámetros de configuración.
- LOAD: permite ejecutar cargas de datos sobre las tablas y pasar estadísticas. Es un permiso a nivel de usuario.
- SECADM: es la autorización encargada de gestionar toda la implementación de las *LBAC* (del inglés *label-based access control=LBAC*), esto es el control de acceso basado en etiquetas.

Los privilegios propiamente dichos pueden ser a nivel de toda la base de datos o solo estar referidos a un objeto concreto. En esencia son prácticamente iguales que los permisos de Oracle, se conceden con el comando *GRANT* y se revocan con el *REVOKE*. Van desde el simple *CONNECT*, pasando por los DML (*SELECT, INSERT, DELETE, UPDATE*), hasta los típicos *ALTER, DROP...*

5.15. Prácticas Resueltas

Práctica 5.1: Inserciones, Actualizaciones y Borrados

Con la base de datos 'Jardinería', crea y ejecuta un script 'actualiza.sql' que realice las siguientes acciones:

1. Inserta una oficina con sede en Fuenlabrada.
2. Inserta un empleado para la oficina de Fuenlabrada que sea representante de ventas.
3. Inserta un cliente del representante de ventas insertado en el punto 2.
4. Inserta un pedido del cliente anterior (con su detalle) de al menos 2 productos con una transacción.
5. Actualiza el código del cliente insertado y averigua si hubo cambios en las tablas relacionadas.
6. Borra el cliente y verifica si hubo cambios.

actualiza.sql

```
#1
INSERT INTO Oficinas VALUES
('FUE-ES','Fuenlabrada','España','Madrid',
 '28941','918837627','C/Las suertes,27','Bajo A');

#2
INSERT INTO Empleados (CodigoEmpleado,Nombre,
    Apellido1,Email,CodigoOficina,Puesto) VALUES
(400,'Ismael','Sánchez','isanchez@jardineria.com','FUE-ES','Rep.Ventas');

#3
INSERT INTO Clientes(CodigoCliente, NombreCliente, Telefono,
    CodigoEmpleadoRepVentas)
VALUES (288,'Riegos Pérez','918882763',400);

#4
START TRANSACTION;
INSERT INTO Pedidos (CodigoPedido, FechaPedido, Estado, CodigoCliente)
    VALUES (1900,'2010-06-03','Pendiente',288);
INSERT INTO DetallePedidos (CodigoPedido, CodigoProducto,Cantidad,
    PrecioUnidad,NumeroLinea) VALUES (1900,'OR-99',1,15.99,1);
INSERT INTO DetallePedidos (CodigoPedido, CodigoProducto ,Cantidad,
    PrecioUnidad,NumeroLinea) VALUES (1900,'OR-251',3,168,2);
COMMIT WORK;

#5
UPDATE Clientes SET CodigoCliente=290 WHERE CodigoCliente=288;
#no permite la modificación, debería tener la FK con ON UPDATE CASCADE
```

```
#6
DELETE FROM Clientes WHERECodigoCliente=288;
#tampoco permite el borrado, debería tener la FK con ON DELETE CASCADE
```



Práctica 5.2: Actualizaciones y borrados con subconsultas

Usa subconsultas en los filtros y realiza las siguientes actualizaciones y borrados:

1. Borra los clientes que no tengan pedidos.
2. Incrementa en un 20 % el precio de los productos que no tengan pedidos.
3. Borra los pagos del cliente con menor límite de crédito.
4. Establece a 0 el límite de crédito del cliente que menos unidades pedidas tenga del producto 'OR-179'.

```
#1
delete from Clientes where CodigoCliente not in
(Select distinct CodigoCliente from Pedidos);
#2
update Productos set PrecioVenta=PrecioVenta*1.2 where not exists
(Select distinct CodigoProducto from DetallePedidos
where DetallePedidos.CodigoProducto=Productos.CodigoProducto);
#3
delete from Pagos where CodigoCliente=
(Select CodigoCliente from Clientes where LimiteCredito =
(Select min(LimiteCredito) from Clientes)
);
#4
update Clientes set LimiteCredito=0 where CodigoCliente=
(Select CodigoCliente from Pedidos natural join DetallePedidos
where Cantidad = (Select Min(Cantidad) From DetallePedidos where
CodigoProducto='OR-179') AND CodigoProducto='OR-179'
);
```



5.16. Prácticas Propuestas

Práctica 5.3: Vincular tablas a través de Access / ODBC

Mediante el driver ODBC para MySQL, enlaza a una BBDD Access las tablas de la NBA y las tablas de la BBDD jardinería. A continuación, realiza las siguientes acciones:

1. Exporta a Excel la tabla jugadores de la NBA.
2. Con Word, crea una carta modelo de felicitación de navidad a los clientes de la base de datos y combina la correspondencia para que, automáticamente, se genere una carta para cada cliente.
3. Inserta dos registros en la tabla Empleados de jardinería mediante un formulario creado en Access, y después, inserta dos jugadores de la NBA siguiendo el mismo procedimiento. Hay que asegurarse de que, efectivamente, están los registros insertados.

Se puede repetir esta misma operación con el driver ODBC para Oracle, pero hay que tener en cuenta que junto al driver, se debe instalar el software cliente de Oracle (sqlplus, tnsnames, etc.) para que funcione. ◇

Práctica 5.4: Actualizaciones y borrados variados

1. Modifica la tabla DetallePedido para insertar un campo numérico llamado IVA. Mediante una transacción, establece el valor de ese campo a 18 para aquellos registros cuyo pedido tenga fecha a partir de Julio de 2010. A continuación actualiza el resto de Pedidos estableciendo al 16 el IVA. 
2. Modifica la tabla DetallePedido para incorporar un campo numérico llamado TotalLinea, y actualiza todos sus registros para calcular su valor con la fórmula $\text{TotalLinea}=\text{PrecioUnidad}*\text{Cantidad}*\text{IVA}/100$. 
3. Borra el cliente que menor límite de crédito tenga. ¿Es posible borrarlo solo con una consulta? ¿Por qué? 
4. A través de phpMyAdmin o, mediante Access (vinculado vía ODBC), inserta dos clientes nuevos para un empleado a tu elección. A continuación, inserta un pedido con al menos 3 líneas de detalle. Después, ejecuta una consulta para rebajar en un 5 % el precio de los productos que sean más caros de 200 euros. 

Práctica 5.5: Inserciones, Actualizaciones y Borrados

En Oracle, con la BBDD 'Jardinería', codifica en SQL las siguientes acciones:

1. Inserta una oficina con sede en Leganés y dos empleados.
2. Inserta un cliente de cada empleado insertado.
3. Inserta dos pedidos de los clientes anteriores (con su detalle) de al menos 2 productos con una transacción.
4. Borra uno de los clientes y comprueba si hubo cambios en las tablas relacionadas. Si no hubo cambios, modifica las tablas necesarias estableciendo la clave foránea con la cláusula ON DELETE CASCADE.
5. Ejecuta el siguiente código para simular el ON UPDATE CASCADE de la tabla Pedidos y modifica el código de algún pedido. Comprueba que haya modificado los registros relacionados en la tabla DetallePedido:

```
CREATE OR REPLACE TRIGGER ActualizaPedidos
AFTER UPDATE ON Pedidos FOR EACH ROW
BEGIN
    UPDATE DetallePedidos SET CodigoPedido = :new.CodigoPedido
    WHERE CodigoPedido= :old.CodigoPedido;
END ActualizaClientes;
```

6. Crea ahora el siguiente disparador y prueba a cambiarle el código a un Empleado. ¿Qué sucede? Busca el concepto de tabla mutante y estudia el problema.

```
CREATE OR REPLACE TRIGGER ActualizaClientes
AFTER UPDATE ON Empleados FOR EACH ROW
BEGIN
    UPDATE Clientes SET CodigoEmpleadoRepVentas = :new.CodigoEmpleado
    WHERE CodigoEmpleadoRepVentas = :old.CodigoEmpleado;
    UPDATE Empleados SET CodigoJefe = :new.CodigoEmpleado
    WHERE CodigoJefe = :old.CodigoEmpleado;
END ActualizaClientes;
/
```



Práctica 5.6: MySQL: Crear usuarios y asignar permisos en local

Realiza las siguientes operaciones y almacena los comandos y los resultados en un fichero:

1. Crea un usuario llamado paco@localhost con la sintaxis create user con permisos de solo conexión y comprueba que se pueda conectar.
2. Crea un usuario llamado juan@localhost con la sintaxis grant con permisos de solo conexión y comprueba que se pueda conectar.
3. Otorga al usuario paco@localhost permisos de select en la tabla jardineria.Clientes y comprueba que se pueda consultar la tabla.
4. Otorga al usuario juan@localhost permisos de select, insert y update en las tablas de la base de datos jardineria con opcion GRANT.
5. Conéctate con el usuario juan y otorga permisos a paco de selección en la tabla jardineria.Empleados.
6. Quítale ahora los permisos a paco de selección sobre la tabla jardineria.Clientes.
7. Conéctate con root y elimina todos los permisos que has concedido a Paco y Juan.
8. Otorga a juan los permisos de SELECT sobre las columnas CodigoOficina y Ciudad de la tabla Oficinas de la base de datos jardineria.
9. Conéctate con juan y ejecuta la query 'SELECT * from jardineria.Oficinas' ¿Qué sucede?.
10. Borra el usuario paco@localhost.



Práctica 5.7: MySQL: Crear usuarios y asignar permisos en remoto

Realiza las siguientes operaciones y almacena los comandos y los resultados en un fichero:

1. Crea un nuevo usuario llamado `usuario@direccion_ip` donde `direccion_ip` es una máquina de un compañero tuyo y `usuario` su nombre.
2. Otórgale permisos de selección en todas las tablas de la base de datos `jardineria`. Ten cuidado, es posible que tu servidor solo permita conexiones desde el ordenador local, para permitir conexiones remotas debes comentar la linea `bind-address` de tu fichero `my.cnf` que impide conexiones desde otros sitios que no sea el especificado (127.0.0.1). Asegúrate de reiniciar el servidor.
3. Pide a tu compañero que se conecte desde su máquina y que averigue qué permisos le has otorgado. El a tí te pedirá lo mismo, es decir, que te conectes a su máquina, indica qué instrucción sql ejecutas para conocer los permisos que tienes.
4. Revócale los permisos concedidos al usuario `usuario@direccion_ip`.
5. Concédele ahora permisos de creación de tablas en una nueva base de datos que has creado.
6. Solicítale que se conecte y que pruebe a crear una tabla. ¿Puede consultar la información?.
7. Borra ahora el usuario `usuario@direccion_ip`.
8. Con la bbdd `mysql` consulta qué privilegios tiene el usuario `juan@localhost` a nivel de servidor, a nivel de base de datos, a nivel de tablas y a nivel de columnas. Utiliza el comando `show grants for usuario`.



Práctica 5.8: Oracle: Creación de usuarios, roles y perfiles

1. Crea dos roles, uno llamado ADMINISTRACION con todos los privilegios sobre la tabla CLIENTES y PEDIDOS del esquema JARDINERIA y otro llamado CONTABILIDAD con todos los privilegios sobre la tabla PAGOS del mismo esquema. El rol CONTABILIDAD estará identificado por contraseña.
2. A continuación, crea un usuario Fernando en Oracle con las siguientes opciones: DEFAULT TABLESPACE Users, TEMPORARY TABLESPACE Temp, QUOTA 100 Megabytes en Users y otorga el rol CONNECT y RESOURCE.
3. Otorga como Rol por defecto ADMINISTRACION al usuario Fernando.
4. Con el usuario Fernando, prueba a ejecutar una consulta sobre la tabla Clientes y otra sobre la tabla PAGOS.
5. A continuación, activa el Rol CONTABILIDAD y repite la consulta sobre la tabla PAGOS con el usuario Fernando.
6. Desactiva todos los roles del usuario Fernando, incluyendo el rol por defecto.
7. Consulta la tabla Clientes.

◊

5.17. Resumen

Los conceptos clave de este capítulo son los siguientes:

- La sentencia INSERT se utiliza para insertar una fila o registro en una tabla. Algunos SGBD, como MySQL, permiten la inserción de más de una fila mediante la sintaxis *extended-insert*. Con INSERT se pueden insertar valores para todas las columnas o solo para algunas de ellas.
- La sentencia UPDATE se utiliza para actualizar uno o varios registros de una tabla. DELETE sirve para eliminar registros de una tabla. Se puede filtrar la información a modificar o borrar mediante un filtro WHERE. El filtro, puede tener todas las características del filtro de la SELECT. Si no se especifica filtro, se actualizan o borran todas las filas de la tabla. También se pueden actualizar o borrar filas con UPDATE y DELETE filtrando a través de una subconsulta. En este caso, la limitación consiste en no poder modificar o borrar registros de una tabla a la que se accede en la subconsulta.
- Se puede realizar la inserción de múltiples registros en una tabla con los resultados devueltos por una SELECT. La SELECT debe devolver tantas columnas como se especifiquen en la sentencia INSERT.
- No siempre es posible actualizar o borrar información de tablas puesto que existen restricciones. Además, algunas restricciones provocan cambios en cada una de las tablas relacionadas.
- Las transacciones son conjuntos de operaciones SQL que se ejecutan de forma atómica. Una vez iniciadas, se pueden *acometer* (COMMIT) o *cancelar* ROLLBACK.
- El tratamiento de transacciones, produce problemas de concurrencia, es decir, problemas que se presentan al haber varios usuarios actualizando, borrando y consultando un conjunto de datos. Estos problemas están clasificados por ANSI/ISO en 3 problemas típicos. Los SGBD permiten que ocurran o no, dependiendo del nivel de aislamiento que se seleccione.
- El comando CREATE VIEW sirve para crear vistas, personalizando la forma en la que el usuario obtiene la información.
- Se puede crear usuarios mediante el comando CREATE USER al cual se le puede asignar un perfil y varios roles. Mediante los comandos GRANT y REVOKE se otorgar y deniegan permisos.

5.18. Test de repaso

1. En la sentencia INSERT

- a) No se pueden omitir valores de columnas
- b) Se pueden especificar un número distinto de valores y columnas
- c) Se pueden omitir los nombres de columnas
- d) Ninguna de las anteriores

2. Con GRANT y REVOKE

- a) Se pueden otorgar permisos de select a una o varias tablas
- b) Se pueden otorgar permisos para borrar una tabla
- c) Se puede otorgar permisos para ejecutar comandos DDL
- d) Todas las anteriores

3. Para poner una columna al valor por defecto

- a) Se tiene que poner NULL
- b) Se tiene que poner DEFAULT
- c) No es posible hacerlo
- d) Ninguna de las anteriores

4. En sentencias UPDATE y DELETE

- a) Se puede especificar el mismo tipo de filtro que para WHERE
- b) Se puede especificar el mismo tipo de filtro que para HAVING
- c) Se puede filtrar mediante una subconsulta
- d) Todas las anteriores

5. Si se especifica ON DELETE CASCADE

- a) No se puede borrar el registro referenciado
- b) Se borra en cascada el registro referenciado
- c) No se puede borrar el registro que referencia
- d) Se borra en cascada el registro que referencia

6. En una sola sentencia UPDATE

- a) Sólo se puede modificar un campo de un registros
- b) Sólo se puede modificar un campo de varios registros
- c) Sólo se pueden modificar varios campos de un registro
- d) Se pueden modificar varios campos de varios registros

7. Una transacción

- a) Puede tener sentencias SELECT
- b) No puede tener sentencias INSERT
- c) No puede tener sentencias SELECT
- d) Sólo puede tener sentencias UPDATE

8. Para comenzar una transacción se usa

- a) START WORK
- b) START TRANSACTION
- c) SET AUTOCOMMIT=OFF
- d) Todas las anteriores

9. Los perfiles en Oracle

- a) Al igual que los roles, sirven para limitar los recursos
- b) No se pueden limitar recursos mediante perfiles
- c) Sirven para agrupar conjunto de permisos
- d) Ninguna de las anteriores

Soluciones: 1.c, 2.d, 3.b, 4.d, 5.b, 6.d, 7.a, 8.d, 9.d.

5.19. Comprueba tu aprendizaje

1. Escribe el formato de la sentencia INSERT y, ejemplifica su funcionamiento mediante dos inserciones, una especificando la lista de campos y otra indicando todos los valores para todas las columnas.
2. Transforma las dos sentencias INSERT anteriores en un único INSERT con sintaxis extendida.
3. Escribe el formato de la sentencia UPDATE y, ejemplifica su funcionamiento mediante tres querys, una que actualice una columna de una fila, una que actualice dos columnas de varias filas, y otra que actualice todas las filas de una tabla.
4. Escribe el formato de la sentencia DELETE y, ejemplifica su funcionamiento mediante dos querys, una que borre una sola fila y otra que borre todos los registros de una tabla.
5. ¿En qué consiste un INSERT-SELECT? Pon un ejemplo de su funcionamiento.
6. ¿Qué condiciones deben darse para que al hacer una modificación en una tabla, sus cambios se propaguen a la tabla que la referencia mediante una clave foránea?
7. Imagina una situación, y pon un ejemplo de borrado en cascada de registros.
8. Imagina otra situación distinta, y pon un ejemplo de actualización en cascada tanto en Oracle, como en MySQL.
9. ¿Qué quiere decir que una actualización o borrado se filtra mediante una consulta? ¿Existe alguna restricción al respecto?
10. Define el concepto de transacción indicando en qué situaciones son útiles y qué problemas puede ocasionar.
11. Define los tipos de problemas ocasionados por la concurrencia en el acceso a los datos:
 - a) Dirty Read o Lectura Sucia
 - b) Nonrepeatable Read o Lectura No Repetible
 - c) Phantom Read o Lectura Fantasma
12. ¿Qué significa que un SGBD tenga una política de aislamiento?
13. Enumera los tipos de aislamiento estándar.