USUARIS, PROGRAMACIÓ BBDD

Contingut

- 2.8.4.3 SHOW TRIGGERS

Gestió d'usuaris

Creació, modificació i eliminació d'usuaris

Creació d'usuaris

Per la creació d'usuaris utilitzem la sentència **CREATE USER**. La sintaxi és la següent:

```
CREATE USER usuari [IDENTIFIED BY [PASSWORD] 'contrasenya']
   [, usuari [IDENTIFIED BY [PASSWORD] 'contrasenya']] ...;
```

La clàusula opcional IDENTIFIED BY fa que es pugui assignar una contrasenya, i la opció PASSWORD serveix per especificar la contrasenya com el valor *hash* retornat.

Exemples:
Crea un usuari que s'anomena Jordi a l'ordinador local:

```
mysql> CREATE USER Jordi@localhost;
```

Crea un usuari que s'anomena Laura i que tingui per contrasenya Terrassa:

```
mysql> CREATE USER Laura IDENTIFIED BY 'Terrassa';
```

Crea un usuari que s'anomena Miquel a l'ordinador amb IP 192.168.0.55 i que envia contrasenya Copernic en valor *hash*:

```
mysql> SELECT PASSWORD('Copernic');
+-------------------------------------------+
| PASSWORD('Copernic')                      |
+-------------------------------------------+
| *0B1D3655498209F8EA0B513CAEAB18520FA55537 |
+-------------------------------------------+
```

```
mysql> CREATE USER Miquel@'192.168.0.55' IDENTIFIED BY PASSWORD '*0B1D3655498209F8EA0B513CAEAB18
```

Per poder utilitzar-la s'ha de tenir permís de CREATE USER o permís INSERT per la base de dades *mysql*. Per cada compte crea un registre a la taula *mysql.user*.

En versions anteriors s'utilitzava l'ordre GRANT per usuaris. Així, per crear un usuari sense permisos utilitzaríem l'ordre:

```
GRANT USAGE ON *.* TO usuari IDENTIFIED BY 'contrasenya';
```

Eliminació d'usuaris

Per la creació d'usuaris utilitzem la sentència DROP USER. La sintaxi és la següent:

```
DROP USER usuari [, usuari] ...;
```

Cal tenir en compte que aquesta sentència no tanca automàticament la sessió, això vol dir que fins que l'usuari no surti de la sessió aquest no s'eliminarà. Una vegada tanqui la sessió l'usuari s'eliminarà, i la següent vegada que vulgui accedir a la sessió li donarà error de no existència d'usuari.

Exemple:
Elimina l'usuari que s'anomena Jordi de l'ordinador local i Miquel de l'ordinador amb IP 192.168.0.55:

```
mysql> DROP USER Jordi@localhost,Miquel@'192.168.0.55';
```

Per poder utilitzar-la s'ha de tenir permís de CREATE USER o permís DELETE per la base de dades *mysql*.

A partir de la versió de MySQL 5.0.2 aquesta sentència esborra els usuaris i els seus permisos. Per les versions 5.0.0 i 5.0.1 primer s'han d'eliminar els seus permisos (mitjançant l'ordre REVOKE) i continuació es pot eliminar l'usuari (amb l'ordre DROP USER).

Reanomenar usuaris

Per reanomenar usuaris existents utilitzem la sentència RENAME USER. La sintaxi és la següent:

```
RENAME USER usuari_vell TO usuari_nou
   [, usuari_vell TO usuari_nou] ...;
```

Exemple:
Reanomena l'usuari Laura i anomena'l ara Marta:

```
mysql> RENAME USER Laura TO Marta;
```

Aquest comandament es va afegir a partir de la versió de MySQL 5.0.2. Per utilitzar-lo s'ha de tenir permís CREATE USER global o el permís UPDATE per la base de dades *mysql*.

Assignar contrasenya

La comanda SET PASSWORD assigna una contrasenya a un compte d'usuari MySQL que existeixi. Si volem assignar una contrasenya a l'usuari actual fer servir la sintaxi és la següent:

```
SET PASSWORD = PASSWORD('some password');
```

Si volem assignar una contrasenya a un altre compte d'usuari fer servir la sentència:

```
SET PASSWORD FOR user = PASSWORD('some password');
```

Exemple:
Assigna la contrasenya 'Nicolau' a l'usuari Marta:

```
mysql> SET PASSWORD FOR Marta = PASSWORD('Nicolau');
```

Gestió de permisos

Nivells de permisos

A MySQL existeixen diferents nivells de permisos:

- **Nivell global**: Els permisos globals s'apliquen a totes les bases de dades d'un servidor. Per indicar l'objecte s'utilitza *.* .
- **Nivell de base de dades**: Els permisos de base de dades s'apliquen a tots els objectes a una base de dades donada. Per indicar l'objecte s'utilitza *nom_base_dades.** .
- **Nivell de taula**: Els permisos de taula s'apliquen a totes les columnes d'una taula. Per indicar l'objecte s'utilitza *nom_base_dades.nom_taula* .
- **Nivell de columna**: Els permisos de columna s'apliquen a les columnes d'una taula. S'ha d'especificar les columnes que se li assignen o treuen permisos.
- **Nivell de rutina**: Els permisos de rutina s'apliquen a procediments emmagatzemats.

Tipus de permisos

Els principals permisos que podem assignar són:

Permís	Descripció
ALL [PRIVILEGES]	Dona tots els permisos
ALTER *	Permet l'ús d'*ALTER TABLE*
CREATE *	Permet l'ús de *CREATE TABLE*
CREATE USER	Permet l'ús de *CREATE USER*, *DROP USER*, *RENAME USER* i *REVOKE ALL PRIVILEGES*
DELETE *	Permet l'ús de *DELETE*
DROP *	Permet l'ús d'*DROP TABLE*
GRANT OPTION *	Permet donar permisos
INDEX *	Permet l'ús de *CREATE INDEX* i *DROP INDEX*
INSERT * **	Permet l'ús d'*INSERT*
SELECT * **	Permet l'ús de *SELECT*
UPDATE * **	Permet l'ús d'*UPDATE*
USAGE	És equivalent a *no privileges*

* Aquest permisos són els únics que poden especificar-se a nivell de taula.
** Aquest permisos són els únics que poden especificar-se a nivell de columna.

Altres permisos que podem asignar són:

Permís	Descripció
ALTER ROUTINE	Modifica o esborra rutines emmagatzemades
CREATE ROUTINE	Crea rutines emmagatzemades
CREATE TEMPORARY TABLES	Permet l'ús de *CREATE TEMPORARY TABLE*
CREATE VIEW	Permet l'ús de *CREATE VIEW*
EXECUTE	Permet a l'usuari executar rutines emmagatzemades
FILE *	Permet l'ús de *SELECT ... INTO OUTFILE* i *LOAD DATA INFILE*
LOCK TABLES	Permet l'ús de *LOCK TABLES*
PROCESS *	Permet l'ús de *SHOW FULL PROCESSLIST*
REFERENCES	No està implementat
RELOAD *	Permet l'ús de *FLUSH*
REPLICATION CLIENT	Permet a l'usuari preguntar on estan els servidors
REPLICATION SLAVE *	Necessari pels esclaus de replicació
SHOW DATABASES *	Permet l'ús de *SHOW DATABASES*
SHOW VIEW	Permet l'ús de *SHOW CREATE VIEW*
SHUTDOWN *	Permet l'ús de *mysqladmin shutdown*
SUPER *	Permet l'ús de *CHANGE MASTER,KILL,PURGE MASTER LOGS* i *SET GLOBAL*

* Aquest permisos només poden donar-se globalment (utilitzant *ON* *.*).

Assignar permisos

La comanda GRANT permet assignar permisos als comptes d'usuaris:

```
GRANT tipus_permis [(columna)] [, tipus_permis [(columna)]] ...
   ON [tipus_objecte] {nom_taula | * | *.* | nom_base_dades.*}
   TO usuari [IDENTIFIED BY [PASSWORD] 'contrasenya']
       [, usuari [IDENTIFIED BY [PASSWORD] 'contrasenya']] ...
   [WITH GRANT OPTION];
```

En el tipus_objecte podem indicar TABLE, PROCEDURE o FUNCTION.

Per donar (i treure) permisos, s'ha de tenir permisos GRANT OPTION.

Podem utilitzar els caràcters comodí '_' i '%' per especificar noms de bases de dades, però no al nom d'usuari.

Exemples:
Crea l'usuari p_admin, que tingui per contrasenya el seu mateix nom, que tingui privilegis totals sobre la base de dades bd_permisos:

```
mysql> GRANT ALL PRIVILEGES
mysql> ON bd_permisos.*
mysql> TO p_admin IDENTIFIED BY 'p_admin';
```

Crea l'usuari p_actual, que tingui per contrasenya el seu mateix nom, que tingui privilegis per visualitzar, modificar, crear i eliminar les dades de la base de dades bd_permisos, però no l'estructura de les taules:

```
mysql> GRANT SELECT, UPDATE, INSERT, DELETE
mysql> ON bd_permisos.*
mysql> TO p_actual IDENTIFIED BY 'p_actual';
```

Crea l'usuari p_disseny, que tingui per contrasenya el seu mateix nom, que tingui privilegis per modificar els camps de la base de dades bd_permisos, però no les dades, ni crear ni afegir taules:

```
mysql> GRANT ALTER
mysql> ON bd_permisos.*
mysql> TO p_disseny IDENTIFIED BY 'p_disseny';
```

Crea l'usuari p_selec, que tingui per contrasenya el seu mateix nom, que tingui privilegis per veure el nom i el cognom de la taula tbl_usuari:

```
mysql> GRANT SELECT (nom,cognom)
mysql> ON bd_permisos.tbl_usuari
mysql> TO p_selec IDENTIFIED BY 'p_selec';
```

La clàusula WITH GRANT OPTION permet que l'usuari que rep privilegis se'ls pugui reassignar a altres.

Exemple:
Crea l'usuari p_delegat, que tingui per contrasenya el seu mateix nom, que tingui privilegis sobre la base de dades bd_permisos per mostrar les dades i permís per donar-li aquest permís a altres usuaris:

```
mysql> GRANT SELECT
mysql> ON bd_permisos.*
mysql> TO p_delegat IDENTIFIED BY 'p_delegat'
mysql> WITH GRANT OPTION;
```

Per recarregar les taules de permisos utilitzem la comanda:

```
FLUSH PRIVILEGES;
```

Treure permisos

La comanda REVOKE permet treure permisos als comptes d'usuaris:

```
REVOKE tipus_permis [(columna)] [, tipus_permis [(columna)]] ...
   ON [tipus_objecte] {nom_taula | * | *.* | nom_base_dades.*}
   FROM usuari [, usuari] ...;
```

Exemple:
A l'usuari p_actual treu el privilegi que té sobre la base de dades bd_permisos d'esborrar registres:

```
mysql> REVOKE DELETE
mysql> ON bd_permisos.*
mysql> FROM p_actual;
```

MySQL per fer més fàcil treure tots els permisos globals de nivell de base de dades i de nivell de taula, utilitza la sentència:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM usuari [, usuari] ...
```

Exemple:
A l'usuari p_actual treu tots el permisos que té sobre la base de dades:

```
mysql>  REVOKE ALL PRIVILEGES, GRANT OPTION FROM p_actual;
```

Mostrar els permisos

Per mostrar els permisos d'un usuari s'utilitza la instrucció:

```
SHOW GRANTS FOR usuari;
```

Para llistar els permisos de la sessió actual, es pot utilitzar la següent ordre:

```
SHOW GRANTS;
```

Programació en bases de dades

SQL (Structured Query Language) és un llenguatge per realitzar consultes en bases de dades relacional. No és com els llenguatges de programació que coneixem (que tenen variables, estructures de control de fluxe, bucles, ...). És per això que existeix una eina de programació que ens ofereixen els sistemes gestors de bases de dades que s'anomena PL/SQL (Procedural Language/Structured Query Language).

Tipus de dades

Els tipus de dades que utilitza són basicament les mateixes que per SQL. Així podem definir tres categories:

- **Numèrics**: per exemple SMALLINT, INT o INTEGER, FLOAT, DOUBLE o REAL, NUMERIC o DECIMAL, ...
- **Text**: CHAR, VARCHAR, TEXT, ENUM, ...
- **Temps**: DATE, TIME, DATETIME, TIMESTAMP, ...

Operadors

Operador d'assignació	:=	Assigna un valor a la variable
Operadors aritmètics	+	Suma
	-	Resta
	*	Producte

	/	Quocient
	**	Potència
Operadors de comparación	=	igual
	<> o !=	diferent
	<	menor
	>	major
	>=	menor o igual
	<=	major o igual
Operador de concatenació	||	

Variables

Les variables són valors per processar al programa. La forma de declarar-ho és:

```
nom_variable TIPUS [NOT NULL] [:=valor | DEFAULT valor]
```

Per iniciar els valors podem utilitzar **:=** o **DEFAULT** (fan el mateix), i si escribim **NOT NULL** fa que sigui obligatori iniciar la variable.

Exemple:
Volem definir dues variables:

- **data**, que serà obligatori que tingui un valor.
- **hora**, que tindrà per defecte el valor de l'hora actual.

```
data     DATE     NOT NULL
hora     HOUR     DEFAULT NOW()
```

Constants

Són variables que no pot ser modificat el seu valor. Es defineixen així:

```
nom_contant CONSTANT TIPUS :=valor
```

Exemple:
Volem definir el màxim de files que podem tenir i serà 20, així definirem:

```
max_files CONSTANT INT := 20
```

Comentaris

S'utilitza per fer aclariments al codi. Podem escriure'ls de dues maneres:

- Si utilitzem /* */ és per escriure comentaris en més d'una línia.
- Si utilitzem -- és per escriure comentaris en una sola línia.

Exemple:

```
/** Això és un comentari
en 2 línies **/


-- Això és un comentari
-- en 2 línies
```

Estructures de control

Control de flux

PL/SQL només disposa de la estructura condicional IF. La seva estructura és:

```
IF (expressió) THEN
      instrucció
[ELSEIF (expressió) THEN
     instrucció]
[ELSE
     instrucció]
END IF;
```

Exemple:
Donat una variable numèrica *var* volem definir *var2* en forma de text dient si el nombre és positiu, negatiu o zero:

```
 IF (var>0) THEN
     var2:='positiu'
 ELSEIF (var<0) THEN
     var2:='negatiu'
 ELSEIF (var=0) THEN
     var2:='zero'
 END IF;
```

CASE

Quan tenim vàries opcions, els *IF-ELSE* generen un codi massa complex, aleshores s'utilitzem el *CASE*:

```
CASE expressió
   WHEN valor1 THEN
      instruccio
   WHEN valor2 THEN
      instruccio
   ...
   ELSE instruccio
END CASE;
```

Exemple:
Donat una variable numèrica *var* volem definir *var2* en forma de text (u,dos,tres,...):

```
CASE var
WHEN 1 THEN
    var2:='u'
WHEN 2 THEN
    var2:='dos'
WHEN 3 THEN
    var2:='tres'
END CASE;
```

Bucle *LOOP*

El bucle *LOOP* es repeteix fins que es força la sortida amb la instrucció LEAVE. La seva sintaxi és:

```
etiqueta: LOOP
   instruccions
   IF (expressió) THEN
      instruccions
      LEAVE etiqueta;
   END IF;
END LOOP etiqueta;
```

Exemple:
Donat una variable numèrica *var* suma 1 fins que valgui 10:

```
bucle: LOOP
    var:=var+1
    IF (var=10) THEN
        LEAVE bucle;
    END IF;
END LOOP bucle;
```

Bucle *WHILE*

L'ordre *WHILE* es repeteix mentre es compleix la condició. La seva sintaxi és:

```
WHILE (expressió) DO
   instruccions
END WHILE;
```

Exemple:
Donat una variable numèrica *var* suma 1 fins que valgui 10:

```
WHILE (var<=10) DO
    var:=var+1
END WHILE;
```

Bucle *REPEAT*

El bucle *REPEAT* es repeteix fins que la condició és certa. La seva sintaxi és:

```
REPEAT
   instruccions
UNTIL condició
END REPEAT;
```

Exemple:
Donat una variable numèrica *var* suma 1 fins que valgui 10:

```
REPEAT
   var:=var+1
UNTIL (var=10)
END REPEAT;
```

Si utilitzem la condició REVERSE el bucle es recorre en sentit invers.

GOTO

La sentència *GOTO* desvia el fluxe a una determinada etiqueta, que s'indiquen <<etiqueta>>. La sintaxi és:

```
GOTO etiqueta:
...
<<etiqueta>>
```

Exemple:
Donat una variable numèrica *var* suma 1 fins que valgui 10:

```
<<etiq>>
var:=var+1
IF (var<10) THEN
    GOTO etiq;
END IF;
```

Estructura d'un bloc

Els blocs en PL/SQL presenten una estructura dividida en 3 parts:

- La part declarativa **DECLARE**, on es defineixen totes les constants i variables que s'utilitzaran.
- La part d'execució **BEGIN**, on es defineixen les accions que s'executaran.
- La part d'excepcions **EXCEPTION**, on es defineixen els errors que suportarà.

La seva sintaxi és:

```
[DECLARE]
   ...
BEGIN
   ...
[EXCEPTION]
   ...
END;
```

Tipus de blocs

Introducció

A MySQL és possible crear rutines. Aquestes rutines (que poden ser funcions o procediments) queden emmagatzemades al servidor de bases de dades i poden ser executades posteriorment. Tenim que:

- Les funcions retornen un valor. Existeixen múltiples funcions predefinides d'SQL, per exemples la funció YEAR().
- Els procediments no retornen un valor, si no que executen una acció.

Els procediments emmagatzemats poden millorar el rendiment donat que envien menys informació entre el servidor i el client.

Necessiten la taula proc de la base de dades mysql, que és on queden emmagatzemats i que es creat durant la instal.lació de MySQL. Els permissos necessaris per executar rutines són:

- Permís CREATE ROUTINE: necessari per crear procediments emmagatzemats.
- Permís ALTER ROUTINE: necessari per modificar i esborrar procediments emmagatzemats. Es dóna automàticament al creador d'una rutina.
- Permís EXECUTE: necessari per executar procediments emmagatzemats. Es dóna automàticament al creador d'una rutina.

Procediments

CREATE PROCEDURE

S'utilitza per crear procediments. La seva sintaxi és:

```
CREATE PROCEDURE nom_procediment ([[IN|OUT|INOUT] nom_parametre tipus], ...)
cos del procediment;
```

DROP PROCEDURE

S'utilitza per eliminar procediments. La seva sintaxi és:

```
DROP PROCEDURE [IF EXISTS] nom_procediment;
```

SHOW CREATE PROCEDURE

Retorna la cadena per executar un procediment. La seva sintaxi és:

```
SHOW CREATE PROCEDURE nom_procediment;
```

CALL

La comanda CALL fa la crida d'un procediment creat.

```
CALL nom_procediment;
```

Exemple:
Volem crear un procediment anomenat *buscar* que donat un paràmetre ens cerqui tots els usuaris que contenen aquest paràmetre de la taula *user* de la base de dades *mysql*:

```
DELIMITER //
CREATE FUNCTION buscar(IN valor VARCHAR(20))
BEGIN
   SELECT User FROM user WHERE User LIKE valor;
END //
DELIMITER ;
```

Utilitzem la sentència DELIMITER per canviar el delimitador de final de sentència i no es confongui el de final de sentència amb l'intermig.

Funcions

CREATE FUNCTION

Al llenguatge SQL tenim unes funcions definides pel sistema, per exemple ABS() o CONCAT(). També es permet que l'usuari pugui definir funcions, el que s'anomenen **funcions definides per l'usuari (UDF)**.

La seva sintaxi és:

```
CREATE [AGGREGATE] FUNCTION nom_funcio ([nom_parametre TIPUS][, ...])
RETURNS {STRING|INTEGER|REAL}
cos del procediment;
```

Amb la funció AGGREGATE funciona com una funció d'agregació, és a dir, com ho fa el COUNT() i el SUM().

Exemple:
Volem crear una funció anomenada *qualificacio* que si posem 'apte' torni una 'A' i si posem 'no apte' torni 'NA':

```
CREATE FUNCTION qualificacio (qualificacio VARCHAR(10))
RETURNS VARCHAR(5)
IF (qualificacio='apte')
   THEN RETURN 'A';
ELSEIF (qualificacio='no apte')
   THEN RETURN 'NA';
ELSE RETURN 'NS/NC';
END IF;
```

DROP FUNCTION

S'utilitza per eliminar funcions. La seva sintaxi és:

```
DROP FUNCTION nom_funcio;
```

Exemple:
Eliminem una funció anomenada *qualificacio*:

```
DROP FUNCTION qualificacio;
```

Disparadors (*triggers*)

Un disparador (en anglès *trigger*) és una rutina emmagatzemada que s'executarà quan en una taula esdevenen events del tipus INSERT, DELETE o UPDATE. Per exemple quan en una taula inserim un nou registre, en aquesta taula podem fer que s'executi una rutina. El mateix podem fer quan s'esborra o bé quan s'actualitza.

CREATE TRIGGER

Al llenguatge SQL la sentència per crear un disparador (en anglès *trigger*) és:

```
CREATE TRIGGER nom_disparador temporalitzacio_disparador esdeveniment_disparador
   ON nom_taula FOR EACH ROW
   cos_disparador;
```

Cal tenir en compte les següents coses:

- La temporalitzacio_disparador pot ser BEFORE o AFTER, segons si volem executar-ho abans o després que la fila sigui modificada.
- L'esdeveniment_disparador és el tipus d'esdeveniment que activarà el disparador i pot ser INSERT, UPDATE o DELETE.

Cal destacar que no pot haver definit sobre una taula un disparador amb la mateixa temporalitzacio_disparador i esdeveniment_disparador. Per exemple, no podrien existir dos disparadors amb BEFORE INSERT.

Exemple:
Crea un disparador que per cada inserció que fem a la taula *prova* compti els registres a la taula *suma*:

```
CREATE DATABASE exemple_disparadors;
USE exemple_disparadors;
CREATE TABLE prova (codi INT);
CREATE TABLE suma (suma INT);
INSERT INTO suma VALUES (0);
DELIMITER //
CREATE TRIGGER actual AFTER INSERT
   ON prova FOR EACH ROW
   BEGIN
   UPDATE suma SET suma=suma+1;
   END //
DELIMITER ;
```

Es pot fer referència a les columnes de la taula subjecte (la taula associada amb el disparador) mitjançant l'ús de:

- **OLD**: OLD.nom_columna es refereix a un camp d'una fila existent abans que s'actualitza o s'elimina.
- **NEW**: NEW.nom_columna es refereix a un camp d'una fila existent després que s'actualitza.

Exemple:
Crea un disparador que per cada inserció que fem a la taula *prova* sumi el camp codi dels registres a la taula *suma*:

```
DELIMITER //
CREATE TRIGGER suma AFTER INSERT
   ON prova FOR EACH ROW
   BEGIN
   UPDATE suma SET suma=suma+NEW.codi;
   END //
DELIMITER ;
```

Exemple:
Crea un disparador que quan s'esborra un registre de *prova* es copia en una altra taula de seguretat anomenada *prova_seg*:

```
CREATE TABLE prova_seg (codi INT);
DELIMITER //
CREATE TRIGGER copia_seg AFTER DELETE
   ON prova FOR EACH ROW
   BEGIN
   INSERT INTO prova_seg VALUES (OLD.codi);
   END //
DELIMITER ;
```

DROP TRIGGER

Per eliminar un disparador utilitzarem la sentència DROP TRIGGER. La sentència per eliminar-lo és:

```
DROP TRIGGER [IF EXISTS] nom_del_trigger;
```

SHOW TRIGGERS

Retorna la cadena de creació del disparador. La seva sintaxi és:

```
SHOW TRIGGERS;
```