 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------


**2253 CFGS Desenvolupament d'Aplicacions Web**  
**Mòdul 2 – Bases de dades**  
**UF3 – Llenguatges SQL: DCL i extensió procedimental**  
**NF3 – Procediments emmagatzemats, triggers i funcions**  
**APUNTS**

## Índex

1. Els procediments emmagatzemats i funcions .....pag 2
2. Els disparadors ..... pag 5

## Breu introducció

En aquest nucli formatiu veurem l'extensió procedimental que aporta el llenguatge SQL per a possibilitar la programació en els entorns de bases de dades. Concretarem en els procediments, les funcions i els disparadors o triggers.

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------


## 1. Procediments i funcions

Un procediment és un conjunt de comandes SQL que poden emmagatzemar-se en un servidor. Des dels clients es posen executar els procediments emmagatzemats al servidor.

Els procediments s'emmagatzemen a la taula **proc** de la base de dades **mysql**. Per a poder utilitzar els procediments cal que l'usuari tingui els següents permisos: **CREATE ROUTINE**, **ALTER ROUTINE** i **EXECUTE**.

La sintaxi per a la gestió de procediments:

CREATE PROCEDURE	Crea un procediment (rutina)
CREATE FUNCTION	Crea una funció (rutina)
CALL	Per a cridar un procediment o funció. Els procediments només es poden cridar amb CALL i en canvi les funcions es poden invocar dins una expressió com les funcions internes. Les funcions poden retornar un valor escalar. Els procediments no retornen cap valor. Les rutines s'associen a una base de dades. Per a invocar el procediment p() o f() definits a la base de dades test cal fer: CALL test.p() CALL test.f()
ALTER PROCEDURE	Modifica la definició d'un procediment
ALTER FUNCTION	Modifica la definició d'una funció
DROP PROCEDURE	Esborra procediment.
DROP FUNCTION	Esborra funció.
SHOW CREATE PROCEDURE	Per a veure els procediments creats.
SHOW CREATE FUNCTION	Per a veure les funcions creades.
SHOW PROCEDURE STATUS	Per a veure els procediments emmagatzemats.
SHOW FUNCTION STATUS	Per a veure les funcions emmagatzemades. Podem afegir un patró: LIKE 'part_dun_nom'
BEGIN END	Inici i final del contingut d'un procediment o funció.
DECLARE	Per a declarar variables.
SET	Assignar valor a variable.
OPEN, FETCH, CLOSE	Per als cursors.
IF	Condicional.
CASE	Condicional.

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------

LOOP	Bucle.
LEAVE	Sortida del bucle.
ITERATE	Continuar al bucle.
REPEAT	Bucle (fins que es compleix condició)
WHILE	Bucle (mentre es compleix condició)

Les funcions i procediments s'esborren a l'esborrar la base de dades.

MySQL permet utilitzar comandes **SELECT** dins dels procediments i funcions.

La sintaxi de creació d'un procediment o funció:


```
CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body
CREATE FUNCTION sp_name ([parameter[,...]])
RETURNS type
[characteristic ...] routine_body
parameter:
[ IN | OUT | INOUT ] param_name type
type:
Any valid MySQL data type
characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
routine_body:
procedimientos almacenados o comandos SQL válidos
```

Un exemple de procediment:

```
delimiter //
CREATE PROCEDURE simpleproc (OUT param1 INT)
BEGIN
SELECT COUNT(*) INTO param1 FROM t;
END
//

delimiter ;
CALL simpleproc(@a);
SELECT @a;
```


Un altre exemple:

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------

```

delimiter //
CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
RETURN CONCAT('Hello, ',s,'!');
//
delimiter ;
SELECT hello('world');

```

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------

## 2. Disparadors (triggers)

Un disparador o trigger és un objecte amb nom dins d'una base de dades el qual s'associa amb una taula i s'activa quan succeeix en aquests un event.

L'ús més habituals dels disparadors és per a verificar valors a inserir o realitzar càlculs sobre valors involucrats a una actualització.

S'associa a una taula i es defineix per a què s'activi abans o després produir-se una sentència **INSERT**, **DELETE** o **UPDATE** en la taula.

Sintaxi:

```
CREATE TRIGGER nom_disp moment_disp event_disp
ON nom_taula FOR EACH ROW sentencia_disp
```


El disparador queda associat a una taula. Amb `moment_disp` indiquem el moment en què el disparador entra en acció potser abans (**BEFORE**) o després (**AFTER**) de la sentència que l'activa. Amb `event_disp` indiquem la classe de sentència que activa el disparador. Per exemple, una disparador **BEFORE** per a sentències **INSERT** podria utilitzar-se per a validar els valors a inserir. No pot haver dos disparadors a la mateixa taula que corresponguin al mateix moment que la sentència. Amb `sentencia_disp` indiquem la sentència que s'executa quan s'activa el disparador. Si es desitgen executar múltiples sentències es posen entre **BEGIN** i **END**.

Per exemple, les següents sentències creen una taula i un disparador associat a la taula anterior, per a sentències **INSERT** dins d'una taula. Els disparador suma els valors inserits en una de les columnes de la taula.

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
```

La sentència **CREATE TRIGGER** crea un disparador anomenat `ins_sum` que s'associa amb la taula `account`. També s'inclouen clàusules que especifiquen el moment d'activació, l'event activador i què fer llavors després de l'activació.

- La paraula clau **BEFORE** indica el moment d'acció del disparador. En aquest cas, el disparador hauria d'activar-se abans que cada registre s'insereixi a la taula. L'altra paraula clau possible és **AFTER**.
- La paraula clau **INSERT** indica l'event que activarà el disparador. En l'exemple, la sentència **INSERT** causarà l'activació. També poden crear-se disparadors per a sentències

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------

#### DELETE i UPDATE.

- La sentència següent, **FOR EACH ROW**, defineix el que s'executarà cada vegada que el disparador s'activi, la qual cosa succeeix una vegada per a cada fila afectada per la sentència activadora. A l'exemple, la sentència activada és un senzill **SET** que acumula els valors inserits a la columna **amount**. La sentència es refereix a la columna com **NEW.amount**, la qual cosa significa "el valor de la columna amount que serà inserit en el nou registre".

I al fer les següents comandes, veurem el funcionament del trigger.

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98), (141,1937.50), (97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
```


Un altre exemple:

```
CREATE DATABASE test;
USE test;
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
  INSERT INTO test2 SET a2 = NEW.a1;
  DELETE FROM test3 WHERE a3 = NEW.a1;
  UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END
|
DELIMITER ;
INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
INSERT INTO test1 VALUES (1), (3), (1), (7), (1), (8), (4), (4);
```

Les columnes de la taula associada amb el disparador poden referenciar-se utilitzant els alies **OLD** i **NEW**. **OLD.nom\_col** fa referència a una columna d'una fila existent, abans de ser actualitzada o

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------

esborrada. **NEW.nom\_col** fa referència a una columna en una nova fila a punt de ser inserida, o en una fila existent després de ser actualitzada.

L'ús de **SET NEW.nom\_col=valor** necessita tenir el privilegi **UPDATE** sobre la columna. L'ús de **SET nom\_val=NEW.nom\_col** necessita el privilegi **SELECT** sobre la columna.

Per a esborrar un disparador:

```
DROP TRIGGER [nom_esq.] nom_disp
```

No es poden tenir diferents disparadors sobre una taula amb el mateix nom, ja que aquest identifica el disparador i aquests estan associats a taules. Tampoc es poden enis disparadors sobre la mateixa taula que siguin activats en el mateix moment i per al mateix event. Per exemple, no es pot definir dos **BEFORE INSERT** o dos **AFTER UPDATE** en la mateixa taula.

També hi ha limitacions en quant al que es pot utilitzar dins les sentències el disparador:

- No pot referir-se a taules directament pel seu nom, inclosa la mateixa taula a la qual està associat. Però es poden utilitzar les paraules clause **OLD** i **NEW**. **OLD** fa referència al registre existent que va a esborrar-se o actualitzar-se abans de que això succeeixi. **NEW** fa referència a un registre nou que s'insereix o a un registre modificat després de que succeeixi l'actualització.
- El disparador no pot invocar procediments utilitzant la sentència **CALL**.
- El disparador no pot utilitzar sentències que iniciïn o finalitzin transacció, tal com **START TRANSACTION**, **COMMIT** o **ROLLBACK**.


En un disparador per a **INSERT** només es pot utilitzar **NEW** perquè no hi ha versió anterior del registre. En un disparador per a **DELETE** només es pot utilitzar **OLD** perquè no hi ha nou registre.

Una columna precedida per **OLD** és només de lectura. Una columna precedida per **NEW** pot ser referenciada si té el privilegi **SELECT** en ella. En un disparador **BEFORE**, també és possible canviar el seu valor amb **SET NEW.nom\_col=val** si té el privilegi **UPDATE** en ella. Això significa que un disparador pot utilitzar-se per a modificar els valors abans que s'insereixin en un nou registre o s'utilitzin per a actualitzar un existent.

En un disparador **BEFORE**, el valor de **NEW** per a una columna **AUTO\_INCREMENT** és 0, no és el nombre seqüencial que es genera de manera automàtica quan el registre sigui realment inserit.

Entre **BEGIN** i **END** podem utilitzar les sintaxis permeses als procediments (bucles, condicionals ..).

Com succeeix amb les rutines emmagatzemades, quan es crea un disparador que executa sentències múltiples, es fa necessari redefinir el delimitador de sentències si s'introdueix el disparador mitjançant el programa mysql, de manera que es pugui utilitzar el caràcter ';' dins de la definició del disparador. El següent exemple mostra aquests aspectes. Es crea un disparador per a **UPDATE**, que verifica els valors utilitzats per a actualitzar cada columna, i modifica el valor per a que es trobi en un rang de 0 a 100. Això ha de fer-se mitjançant un disparador **BEFORE** perquè els valors an de verificar-se abans d'utilitzar-se per a actualitzar el registre:

 <b>INSTITUT AUSIÀS MARCH</b> Consorci d'Educació de Barcelona Generalitat de Catalunya Ajuntament de Barcelona	<b>APUNTS</b> <b>Desenvolupament d'Aplicacions Web</b> <b>Mòdul 2: Bases de Dades</b>	<b>FP_ICC0M02</b>
--	---	-------------------

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
  -> FOR EACH ROW
  -> BEGIN
  ->     IF NEW.amount < 0 THEN
  ->         SET NEW.amount = 0;
  ->     ELSEIF NEW.amount > 100 THEN
  ->         SET NEW.amount = 100;
  ->     END IF;
  -> END; //
mysql> delimiter ;
```

Una limitació dels disparadors és que no pot utilitzar **CALL**.

MySQL gestiona els errors produïts durant l'execució de disparadors de la següent manera:

- Si falla un disparador **BEFORE**, no s'executa l'operació en el corresponent registre.
- Un disparador **AFTER** s'executa només si el disparador **BEFORE** (d'existir) i l'operació es van executar amb èxit.
- Una errada durant l'execució d'un disparador **BEFORE** o **AFTER** produeix la fallida de tota la sentència que va provocar la invocació del disparador.
  - En taules transaccionals, la fallida d'un disparador (i evidentment de tota la sentència) hauria de causar la cancel·lació (rollback) de tots els canvis realitzats per la sentència. En taules no transaccionals, qualsevol canvi realitzat abans de l'error no es veu afectat.