

# **UF3 - Llenguatges SQL: DCL i extensió procedimental**

NF3: Extensió procedimental

# La nostra base de dades: dbuoc

```
CREATE DATABASE IF NOT EXISTS DBUOC;

CREATE TABLE IF NOT EXISTS DBUOC.CLIENTS
(cod_cli TINYINT PRIMARY KEY,
 name_cli VARCHAR(30) NOT NULL,
 nif CHAR(9) NOT NULL UNIQUE,
 address VARCHAR(50),
 city VARCHAR(30),
 phone CHAR(9)
) ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS DBUOC.DEPARTMENTS
(name_dpt VARCHAR(30),
 city_dpt VARCHAR(30),
 phone CHAR(9),
 PRIMARY KEY(name_dpt,city_dpt)
) ENGINE=InnoDB;
```

# La nostra base de dades: dbuoc

```
CREATE TABLE IF NOT EXISTS DBUOC.PROJECTS
(cod_proj TINYINT PRIMARY KEY,
 name_proj VARCHAR(30) NOT NULL UNIQUE,
 price DECIMAL(9,2),
 iniDate DATE,
 prevFiDate DATE,
 fiDate DATE,
 cod_client TINYINT,
 FOREIGN KEY (cod_client) REFERENCES DBUOC.CLIENTS (cod_cli)
      ON DELETE SET NULL ON UPDATE CASCADE,
 CHECK (iniDate < prevFiDate AND iniDate < fiDate)
) ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS DBUOC.EMPLOYEES
(cod_empl TINYINT PRIMARY KEY,
 name_empl VARCHAR(30) NOT NULL,
 surname_empl VARCHAR(30) NOT NULL,
 salary DECIMAL(9,2) CHECK (salary > 12500.00),
 name_dpt VARCHAR(30),
 city_dpt VARCHAR(30),
 num_proj TINYINT,
 FOREIGN KEY (name_dpt,city_dpt) REFERENCES DBUOC.DEPARTMENTS(name_dpt,city_dpt)
      ON DELETE SET NULL ON UPDATE CASCADE,
 FOREIGN KEY (num_proj) REFERENCES DBUOC.PROJECTS(cod_proj)
      ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB;
```

# Extensió procedimental

1. Procediments (procedures)
2. Funcions (functions)
3. Disparadors (triggers)

Tots són conjunt de comandes SQL que s'emmagatzemen sota un nom al servidor i s'executen com una unitat.

Les funcions i procediments es criden.

Els disparadors s'executen quan succeeix un esdeveniment al servidor.

# Procediments

## Sintaxi:

Definició de procediment:

```
DELIMITER //  
CREATE PROCEDURE nomProcediment ([IN | OUT paràmetre tipus, ...])  
BEGIN  
    sentències;  
END;  
//  
DELIMITER ;
```

Si el procediment només té una sentència no cal posar BEGIN i END.

```
CREATE PROCEDURE nomProcediment (paràmetres)  
    sentència;
```

Els paràmetres són opcionals. Si són d'entrada posem IN i si són de sortida posem OUT.

# Procediments

## Sintaxi:

Crida al procediment amb paràmetres:

```
CALL nomProcediment (variables o literals)
```

-> Si les variables són globals, cal posar el símbol @ davant del nom de la variable.

Crida al procediment sense paràmetres:

```
CALL nomProcediment ()
```

Esborrar un procediment:

```
DROP PROCEDURE nomProcediment;
```

# Procediments

## Exemples:

Procediment sense paràmetres que ens mostra per pantalla la quantitat de projectes en curs:

```
DELIMITER //
```

```
CREATE PROCEDURE projectesEncurs ()
```

```
BEGIN
```

```
    SELECT COUNT(*) AS projEncurs FROM PROJECTS WHERE fiDate IS NULL;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
CALL projectesEncurs ();
```

Implementació sense BEGIN END ja que només és una sentència

```
CREATE PROCEDURE projectesEncurs ()
```

```
    SELECT COUNT(*) AS projEncurs FROM PROJECTS WHERE fiDate IS NULL;
```

# Procediments

## Exemples:

Implementació de procediment amb paràmetre de sortida que ens diu la quantitat de projectes en curs:

```
DELIMITER //
```

```
CREATE PROCEDURE projectesAcabats (OUT projAc INTEGER)
```

```
BEGIN
```

```
    SELECT COUNT(*) FROM PROJECTS WHERE fiDate IS NOT NULL INTO projAc;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
CALL projectesAcabats (@a);
```

```
SELECT @a;
```



# Procediments

## Exemples:

Implementació de procediment amb paràmetre d'entrada i sortida que ens diu quants empleats treballen en un projecte passat com a paràmetre.

```
DELIMITER //
```

```
CREATE PROCEDURE numEmpleatsPerprojecte (IN p INTEGER, OUT ne INTEGER)
```

```
BEGIN
```

```
    SELECT COUNT(*) FROM empleats WHERE num_proj=p INTO ne;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
CALL numEmpleatsPerprojecte (1, @ne);
```

```
SELECT @ne;
```

# Funcions

## Sintaxi:

Definició de procediment:

```
DELIMITER //
```

```
CREATE FUNCTION nomFuncio ([param tipus,...]) RETURNS tipus
```

```
BEGIN
```

```
    sentències;
```

```
    RETURN expressio;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

Els paràmetres són opcionals i només poden ser d'entrada. No cal indicar IN.

# Funcions

## Sintaxi:

Crida al procediment amb paràmetres:

```
SELECT nomFuncio (variables o literals)
```

-> Si les variables són globals, cal posar el símbol @ davant del nom de la variable.

Crida al procediment sense paràmetres:

```
SELECT nomFuncio ()
```

Esborrar un procediment:

```
DROP FUNCTION nomFuncio;
```

# Funcions

## Exemples:

Implementació de funció que ens saluda:

```
CREATE FUNCTION hello (s CHAR(30)) RETURNS CHAR(50)  
    RETURN CONCAT ('Hello ', s, '!');
```

```
SELECT hello('world');
```

```
DELIMITER //  
CREATE FUNCTION hello (s CHAR(30)) RETURNS CHAR(50)  
BEGIN  
    RETURN CONCAT ('Hello ', s, '!');  
END;  
//  
DELIMITER ;
```

# Exemples:

1. procediment que obtingui el preu màxim, mínim i mitjana dels projectes amb dos decimals.

```
CREATE PROCEDURE calculProjectes ()  
    SELECT round(min(price),2), round(max(price),2),  
           |round(avg(price),2)  
    FROM PROJECTS;  
  
CALL calculProjectes();  
  
DROP PROCEDURE calculProjectes;
```

# Exemples:

1. procediment que obtingui el preu màxim, mínim i mitjana dels projectes amb zero decimals i els deixi ens uns paràmetres de sortida.

```
• CREATE PROCEDURE calculProjectes (OUT pmin FLOAT,  
                                     OUT pmax FLOAT,  
                                     OUT pavg FLOAT)  
    SELECT ROUND(MIN(price),0), ROUND(MAX(price),0),  
           ROUND(AVG(price),0) FROM PROJECTS  
    INTO pmin, pmax, pavg;  
  
• CALL calculProjectes(@a, @b, @c);  
  
• SELECT @a, @b, @c;  
  
• DROP PROCEDURE calculProjectes;
```

# Exemples:

2. funció que retorni la suma dels sous dels empleats que treballen en un projecte que passem per paràmetre (nom)

```
DELIMITER //
CREATE FUNCTION calculSous (nom VARCHAR(30)) RETURNS DECIMAL(11,2)
BEGIN
    DECLARE sum DECIMAL(11,2);
    SELECT sum(salary) FROM EMPLOYEES JOIN PROJECTS
        ON (num_proj=cod_proj) WHERE name_proj=nom INTO sum;
    RETURN sum;
END;
//
DELIMITER ;

SELECT calculSous('GESCOM');

DROP FUNCTION calculSous;
```

# Exemples:

3. procediment que esborri el contingut de les taules de la base de dades dbuoc.

```
DELIMITER //
```

- `CREATE PROCEDURE esborrarBd ()`  
  `BEGIN`  
    `DELETE FROM CLIENTS;`  
    `DELETE FROM PROJECTS;`  
    `DELETE FROM EMPLOYEES;`  
    `DELETE FROM DEPARTMENTS;`  
  `END;`  
  `//`  
  `DELIMITER ;`
- `CALL esborrarBd();`
- `DROP PROCEDURE esborrarBd;`



# Variables

## Declaració de variables:

```
DECLARE nomVar tipus [DEFAULT val]
```

```
DECLARE v1 INT DEFAULT 5;
```

## Assignar valors a variables:

```
SET nomVar = valor;
```

```
SET v1=5;
```

**Variables globals:** accessibles des de l'entorn i des dels procediments i funcions.

```
@nomVar
```

# Estructures de control condicionals

## Sentència IF

```
IF condició_cerca THEN llista_sentències  
    [ELSEIF condició_cerca THEN llista_sentències] ...  
    [ELSE llista_sentències]  
END IF
```

## Sentència CASE

```
CASE variable  
    WHEN value1 THEN llista_sentències  
    [WHEN value2 THEN llista_sentències] ...  
    [ELSE llista_sentències]  
END CASE
```

```
CASE  
    WHEN condició_cerca THEN llista_sentències  
    [WHEN condició_cerca THEN llista_sentències] ...  
    [ELSE llista_sentències]  
END CASE
```

# Exemple estructura if

## Exemple:

```
DELIMITER //
CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)
  BEGIN
    DECLARE s VARCHAR(20);
    IF n > m THEN SET s = '>';
      ELSEIF n = m THEN SET s = '=';
      ELSE SET s = '<';
    END IF;
    SET s = CONCAT(n, ' ', s, ' ', m);
    RETURN s;
  END //
DELIMITER ;

SELECT SimpleCompare(4,7);
```

# Exemple estructura case

```
DELIMITER //  
CREATE FUNCTION CaseCompare(n INT, m INT)  
  RETURNS VARCHAR(20)  
  BEGIN  
    DECLARE s VARCHAR(20);  
    CASE  
      WHEN n > m THEN SET s = '>';  
      WHEN n = m THEN SET s = '=';  
      ELSE SET s = '<';  
    END CASE;  
    SET s = CONCAT(n, ' ', s, ' ', m);  
    RETURN s;  
  END //  
DELIMITER ;  
  
SELECT SimpleCom,pare(4,7);
```

# Estructures de control iteratives

## Sentència WHILE

```
WHILE condicio_cerca DO  
    lista_sentències  
END WHILE
```

## Sentència REPEAT:

```
REPEAT  
    llista_sentències  
UNTIL condicio_cerca  
END REPEAT
```

# Exemple WHILE

## Iteratives:

### Sentència WHILE

```
DELIMITER //  
CREATE PROCEDURE dowhile (p1 INT)  
BEGIN  
    SET @x = 0;  
    WHILE @x < p1 DO  
        SET @x = @x + 1;  
    END WHILE;  
END;  
//  
DELIMITER ;  
CALL dowhile(100);  
SELECT @X;
```

# Exemple REPEAT

## Iteratives:

### Sentència REPEAT

```
DELIMITER //
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
    SET @x = 0;
    REPEAT SET @x = @x + 1;
    UNTIL @x > p1
    END REPEAT;
END;
//
DELIMITER ;
CALL dorepeat(100);
SELECT @X;
```

# DISPARADORS (TRIGGERS)

Els disparadors o triggers són un conjunt de sentències que s'executen com una unitat. Estan associats a una taula i s'executen quan es produeix un esdeveniment sobre la taula.

Els esdeveniments són les operacions d'INSERT, UPDATE o DELETE sobre la taula.

I es pot executar abans o després de l'esdeveniment o operació d'actualització.

```
CREATE TRIGGER trigger_name  
    trigger_time trigger_event  
ON tbl_name FOR EACH ROW  
    trigger_body
```

```
trigger_time: { BEFORE | AFTER }
```

```
trigger_event: { INSERT | UPDATE | DELETE }
```



# DISPARADORS (TRIGGERS)

En el cos del trigger podem fer referència als valors antics i/o nous de les columnes.

- **NEW.nom\_col**: fa referència al nou valor que tindrà la columna després d'executar-se l'operació d'actualització que activa el disparador. No es pot utilitzar amb **DELETE**.
- **OLD.nom\_col**: fa referència al valor de la columna anterior a l'execució de l'operació que activa el disparador. No es pot utilitzar amb **INSERT**.
- Les dues es poden utilitzar a l'operació **UPDATE**.

Els disparadors s'utilitzen per:

- Control d'errors abans d'inserir o actualitzar registres.
- Per a fer operacions addicionals a la BD i així mantenir la BD íntegra.

# Examples

```
CREATE DATABASE test;  
USE test;
```

```
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;
```

```
SET @sum = 0;
```

```
INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
```

```
SELECT @sum AS 'Total amount inserted';
```

# Examples

```
DELIMITER //
```

```
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;
```

```
//
```

```
DELIMITER ;
```

```
UPDATE account SET amount=200;
```

```
SELECT * FROM account;
```


# Exemples

1/ Afegir a la taula DBUOC.DEPARTMENTS la columna pressupost. Afegir un disparador que s'executi a l'inserir empleat i actualitzi el pressupost sumant el sou i un disparador que s'executi a l'esborrar empleat i actualitzi el pressupost del departament restant el sou.

2/ Definir nova BD GEOGRAFIAi una taula RUTES on volem guardar distàncies en km i milles. Defineix disparador que a l'inserir ruta (només especificant km), calculi les milles i les inserexi. Taula ruta: nom\_ruta (CP), dist\_km, dist\_mil (1 Km=0.621371 milles i 1 Milla=1.609344 Km)

# Exemple 1

Afegim el camp i l'actualitzem:

- `ALTER TABLE DEPARTMENTS ADD COLUMN budget DECIMAL(11,2);`
- `SELECT * FROM DEPARTMENTS;`
- `UPDATE DEPARTMENTS SET budget=`  
`(SELECT SUM(salary) FROM`  
`EMPLOYEES WHERE DEPARTMENTS.name_dpt=EMPLOYEES.name_dpt`  
`AND DEPARTMENTS.city_dpt=EMPLOYEES.city_dpt);`
- `UPDATE DEPARTMENTS SET budget=0 WHERE`  
`name_dpt='PROG' AND city_dpt='Girona';`

# Example 1

```
CREATE TRIGGER sumBudget AFTER INSERT ON EMPLOYEES
FOR EACH ROW
UPDATE DEPARTMENTS SET budget=budget+
NEW.salary WHERE name_dpt=NEW.name_dpt AND city_dpt=NEW.city_dpt;

SELECT * FROM DEPARTMENTS;

INSERT INTO DBUOC.EMPLOYEES
VALUES (9, 'Sergio', 'Rey', 45000.00, 'PROG', 'Tarragona', 1);


SELECT * FROM DEPARTMENTS;

CREATE TRIGGER restBudget AFTER DELETE ON EMPLOYEES
FOR EACH ROW
UPDATE DEPARTMENTS SET budget=budget-
OLD.salary WHERE name_dpt=OLD.name_dpt AND city_dpt=OLD.city_dpt;

DELETE FROM EMPLOYEES WHERE cod_empl=9;

SELECT * FROM DEPARTMENTS;
```

# Example 2

-  CREATE TABLE GEOGRAFIA (  
    dist\_km DECIMAL(15,2) PRIMARY KEY,  
    dist\_mil DECIMAL(15,2)  
);
- CREATE TRIGGER insertDisp BEFORE INSERT ON GEOGRAFIA  
FOR EACH ROW  
    SET NEW.dist\_mil=0.621371\*NEW.dist\_km;
- INSERT INTO GEOGRAFIA (dist\_km)  
VALUES (1.0), (2.0), (3.0), (4.0), (5.0);
- SELECT \* FROM GEOGRAFIA;

# CURSORS

Fins ara hem gestionat el resultat d'una sentència SELECT quan aquest és un literal.

MySQL permet també la gestió del resultat quan aquest està format per una taula amb diverses files i columnes mitjançant els **CURSORS**.

El cursor permet gestionar el resultat d'una consulta de fila en fila. És a dir, el cursor “viaja” seqüencial per tot el resultat de la consulta prenent el valor de cada fila a cada iteració. El procés sempre va relacionat amb una estructura de control iterativa.

Les operacions són:

- declarar i obrir (DECLARE, OPEN)
- assignar al cursor una fila (FETCH)
- tancar (CLOSE)



# CURSORS

Declarem el cursor:

```
DECLARE cursor_name CURSOR FOR select_statement;
```

La sentència SELECT no pot tenir una clausula INTO.

Els cursors s'han de definir després de la declaració de les variables i abans dels gestors d'errors.

Obrim el cursor:

```
OPEN cursor_name;
```

# CURSORS

Assignem valor al cursor:

```
FETCH cursor_name INTO var1 [, var2] ...;
```

Aquesta declaració llegeix la fila següent de la sentència SELECT associada al cursor especificat i que ha d'estar obert, i avança el punter del cursor . Si hi ha una fila, les columnes captades s'emmagatzemen en les variables especificades. El nombre de columnes retornades per la sentència SELECT ha de coincidir amb el nombre de variables de sortida especificades.

Si no hi ha disponibles més files, es produeix una condició d'error amb valor SQLSTATE ' 02000 ' . Cal gestionar aquesta condició de finalització.

# CURSORS

Tanquem el cursor:

```
CLOSE cursor_name;
```

Gestionem la finalització de la lectura:

```
DECLARE done BOOLEAN DEFAULT FALSE;  
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET  
done=TRUE;  
WHILE NOT done DO  
  
END WHILE;
```

# Exemple a DBUOC

Creació procediment per a emmagatzemar en una taula els projectes amb preu superior a 1000000.

- `CREATE TABLE DBUOC.EXPENSIVEPROJ (`  
    `codProj TINYINT,`  
    `price DECIMAL(9,2)`  
    `);`
- `USE DBUOC;`
- `CALL curdemo();`
- `DROP PROCEDURE curdemo;`
- `SELECT * FROM EXPENSIVEPROJ;`
- `SELECT * FROM PROJECTS;`

#	codProj	price
1	2	1200000.00
2	4	2400000.00

# Exemple a DBUOC

```
DELIMITER //
//
CREATE PROCEDURE curdemo()
BEGIN
    DECLARE done BOOLEAN DEFAULT FALSE;
    DECLARE priceProj DECIMAL(9,2);
    DECLARE codProj TINYINT;
    DECLARE cur1 CURSOR FOR SELECT cod_proj, price FROM DBUOC.PROJECTS;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    DELETE FROM DBUOC.EXPENSIVEPROJ;
    OPEN cur1;
    FETCH cur1 INTO codProj, priceProj;
    WHILE NOT done DO
        IF priceProj > 1000000 THEN
            INSERT INTO DBUOC.EXPENSIVEPROJ VALUES (codProj, priceProj);
        END IF;
        FETCH cur1 INTO codProj, priceProj;
    END WHILE;
    CLOSE cur1;
END;
//
DELIMITER ;
```