UF03 - Llenguatges SQL: DCL i extensió procedimental

Procediments, funcions i Cursors a MySQL

MySQL disposa d'un llenguatge de programació procedimental que permet augmentar el rendiment de les aplicacions.

Lliuraments

Per dur un control de la feina realitzada a classe cal que creu un nou repositori al vostre compte de github.com.

El repositori s'anomenarà ***exercicis_mp02_uf03***, i cal que afegiu a l'usuari ***joanpardogine*** com a col·laborador d'aquest.

Dins d'aquest repositori cal que aneu pujant totes les activitats que anem realitzant. Tant aquelles que es faran a classe durant l'explicació, com aquelles que es demanarà lliurar per poder ser avaluades.

Procediments emmagatzemats.

Què és un procediment emmagatzemat?

Un procediment emmagatzemat és un conjunt d'instruccions (comandaments SQL) a les que se'ls dona un nom, que s'emmagatzema en el servidor. Permeten encapsular tasques repetitives.

Un procediment emmagatzemat pot fer referència a objectes (taules, vistes, etc.) que no existeixen a l'hora de crear-lo. Els objectes han d'existir quan s'executi el procediment emmagatzemat.

Avantatges

- Comparteixen la lògica de l'aplicació amb les altres aplicacions, amb la qual cosa l'accés i les modificacions de les dades es fan en un sol lloc.
- Permeten fer totes les operacions que els usuaris necessiten evitant que tinguin accés directe a les taules.
- Redueixen el tràfic de xarxa; en comptes d'enviar moltes instruccions, els usuaris fan operacions enviant una única instrucció, la qual cosa disminueix el nombre de sol·licituds entre el client i el servidor.

Desavantatges

- Les instruccions que podem utilitzar dins d'un procediment emmagatzemat no estan preparades per implementar lògiques de negocis molt complexes.
- Són difícils de depurar.

Coneixent els avantatges i desavantatges dels procediments emmagatzemats hem d'identificar els casos on ens poden facilitar la implementació de les nostres aplicacions.

`CREATE PROCEDURE nom_procediment()` **Crear procediment**

Els **procediments emmagatzemats** es creen a la base de dades seleccionada. En primer lloc s'han d'escriure i provar les instruccions que s'inclouen en el procediment emmagatzemat, després, si s'obté el resultat esperat, es crea el procediment.

Els procediments emmagatzemats poden fer referència a taules, vistes i altres procediments emmagatzemats. Un procediment emmagatzemat pot incloure qualsevol quantitat i tipus d'instruccions.

Per crear un **procediment emmagatzemat** emprem la instrucció `CREATE PROCEDURE`. Que té la següent sintaxi:

```
CREATE PROCEDURE [nom_procediment]()
  BEGIN
    [Sentències]
  END //
```

Per exemple

Amb les següents instruccions creem un procediment emmagatzemat anomenat `01_llistaTreballadors` que retorna una llista amb les dades de cada treballador.

```
mysql> CREATE PROCEDURE 01_llistaTreballadors()
    -> BEGIN
    ->     SELECT *
    ->     FROM   TREBALLADORS;
    -> END //
```

`CALL nom_procediment()` **Execució d'un procediment**

Per ***cridar*** després al procediment emmagatzemat hem d'utilitzar la clàusula **`CALL`** i seguidament el ***nom del procediment*** emmagatzemat.

La instrucció **`CALL`** invoca un procediment emmagatzemat definit prèviament amb **`CREATE PROCEDURE`**.

La llista de paràmetres inclosa entre parèntesis sempre ha d'estar present. Si no hi ha paràmetres, s'ha d'utilitzar una llista sense paràmetres, és a dir buida **`()`**. Per exemple, **`CALL p()`** i **`CALL p`** són equivalents.

Els noms de paràmetres no són ***case-insensitive***, és a dir, que no es diferencien entre majúscules i minúscules.

`CALL` pot retornar valors a l'usuari que la invoca mitjançant paràmetres que es declaren com a paràmetres **`OUT`** o **`INOUT`**.

La comanda **`CALL`** té la següent sintaxi:

```
CALL NOM_PROCEDIMENT([paràmetres[,...]])
CALL NOM_PROCEDIMENT[()]
```

Per exemple

```
mysql> CALL 01_llistaTreballadors;
```

`BEGIN ... END` **Clàusules d'inici i final**

la sintaxi s'utilitza per escriure sentències compostes, que poden aparèixer dins de programes emmagatzemats (procediments i funcions emmagatzemades, desencadenants i esdeveniments).

`DELIMITER` **Clàusula `DELIMITER`**

Com un procediment emmagatzemat pot tenir moltes ordres SQL entre les paraules claus **`BEGIN ... END`** hem d'informar d'alguna manera a MySQL que no executi aquestes ordres. Per a això utilitzem el comandament **`DELIMITER`** (més info a 20.1 Defining Stored Programs) canviant el caràcter `;`

com a fi d'instrucció. Després hem de codificar el procediment emmagatzemat canviant el delimitador amb la següent sintaxi:

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE 01_llistaTreballadors()
    -> BEGIN
    ->     SELECT *
    ->     FROM   TREBALLADORS;
    -> END //
    -> DELIMITER ;
```

Utilitzem el delimitador `//` com podria ser qualsevol altre, per exemple `$`:

```
mysql> DELIMITER $
mysql> CREATE PROCEDURE 01_llistaTreballadors()
    -> BEGIN
    ->    SELECT *
    ->    FROM   TREBALLADORS;
    ->  END $
mysql> DELIMITER ;
```

Heu d'evitar l'ús del caràcter barra invertida (`\`) perquè aquest és el caràcter d'escapament de MySQL.

```
mysql> USE empresa;
mysql> DELIMITER //
mysql> DROP PROCEDURE IF EXISTS 01_llistaTreballadors//
mysql> CREATE PROCEDURE 01_llistaTreballadors()
    -> BEGIN
    ->     SELECT *
    ->     FROM   TREBALLADORS;
    -> END //
mysql> DELIMITER ;
```

On:

- Els procediments s'emmagatzemen en una base de dades concreta, per tant primer hem de fer un **USE**.
- La primera ordre és **DELIMITER //** que canvia el delimitador estàndard que és el punt i coma (**;**) a un altre (en aquest cas la doble barra (**//**).
- Posteriorment s'elimina el procediment per si existeix una versió anterior.
- La instrucció **CREATE PROCEDURE** serveix per crear el procediment nou. Cal especificar el nom del procediment seguit de dos parèntesis.
- El cos del procediment és la secció entre **BEGIN ... END**. Aquí escriurem les sentències de SQL per executar el que volem que faci el procediment.
- I finalment tornem a canviar el delimitador al punt i coma **DELIMITER;**, que és l'estàndard.

Sortida

```
mysql> USE empresa;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> DELIMITER //
mysql> DROP PROCEDURE IF EXISTS 01_llistaTreballadors//
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE PROCEDURE 01_llistaTreballadors()
    -> BEGIN
    ->     SELECT *
    ->     FROM TREBALLADORS;
    -> END//
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
mysql>
```

Finalment, per cridar al procediment fem servir l'ordre **CALL**.

```
mysql> CALL 01_llistaTreballadors;
+-----------+-----------+------------+-----------+----------+---------------+----------+--------------+--------------+
| CODI_TREB | NOM_TREB  | COGNOM_TREB | OFICI_TREB | CAP_TREB | DATA_ALTA_TREB | SOU_TREB | COMISSIO_TREB | CODI_DEP_TREB |
+-----------+-----------+------------+-----------+----------+---------------+----------+--------------+--------------+
|      7369 | Isabel    | Sánchez    | Empleat   |     7902 | 2010-12-17    |   625.05 |         NULL |            20 |
|      7499 | Lluis     | Arroyo     | Venedor   |     7698 | 2010-02-20    |  1250.11 |       234.39 |            30 |
|      7521 | Miquel    | Sala       | Venedor   |     7698 | 2011-02-22    |   976.64 |       390.66 |            30 |
|      7566 | Jacinto   | Jiménez    | Director  |     7839 | 2011-04-02    |  2324.41 |         NULL |            20 |
|      7654 | Albert    | Martín     | Venedor   |     7698 | 2011-09-29    |   976.64 |      1093.84 |            30 |
|      7698 | Pere      | Negre      | Director  |     7839 | 2011-05-01    |  2226.75 |         NULL |            30 |
|      7782 | Antoni    | Cerezo     | Director  |     7839 | 2011-06-09    |  1914.22 |         NULL |            10 |
|      7788 | Andreu    | Gil        | Analista  |     7566 | 2011-11-09    |  2343.95 |         NULL |            20 |
|      7839 | Alex      | Rey        | President |     NULL | 2011-11-17    |  3906.58 |         NULL |            10 |
|      7844 | Joan Lluis | Tovar     | Venedor   |     7698 | 2011-09-08    |  1171.97 |         0.00 |            30 |
|      7876 | Ferran    | Alonso     | Empleat   |     7788 | 2011-09-23    |   859.45 |         NULL |            20 |
|      7900 | Ramon     | Jimeno     | Empleat   |     7698 | 2011-12-03    |   742.25 |         NULL |            30 |
|      7902 | Ignasi    | Fernández  | Analista  |     7566 | 2011-12-03    |  2343.95 |         NULL |            20 |
|      7934 | Jordi     | Muñoz      | Empleat   |     7782 | 2012-01-23    |  1015.71 |         NULL |            10 |
+-----------+-----------+------------+-----------+----------+---------------+----------+--------------+--------------+
```

Exemples

Per veure un exemple amb més detall seguim els següents passos:

1. Creació base de dades ***LLIBRERIA***. (fitxer CreaDBLlibreria.sql).

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| LLIBRERIA          |
+--------------------+
```

2. Només s'ha creat un única taula ***LLIBRES***

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_LLIBRERIA |
+---------------------+
| LLIBRES             |
+---------------------+

mysql> DESC LLIBRES;
+----------+-------------+------+-----+---------+----------------
+
| Field    | Type        | Null | Key | Default | Extra
|
+----------+-------------+------+-----+---------+----------------
+
```

```
| CODI      | int(11)      | NO   | PRI | NULL    | auto_increment
|
| TITOL     | varchar(40)  | YES  |     | NULL    |
|
| AUTOR     | varchar(30)  | YES  |     | NULL    |
|
| EDITORIAL | varchar(20)  | YES  |     | NULL    |
|
| PREU      | decimal(5,2) | YES  |     | NULL    |
|
| ESTOC     | int(11)      | YES  |     | NULL    |
|
+-----------+--------------+------+-----+---------+----------------
+
```

3. I volem crear un procediment que ens retorni una llista (**TITOL**, **EDITORIAL** i **ESTOC**) dels llibres que estan prop d'esgotar-se. És a dir, que només en tenim en estoc 5 o menys llibres.

4. Primer creem la consulta SQL que ens retornarà el que ens demanen.

```
mysql>  SELECT TITOL, EDITORIAL, ESTOC
    ->  FROM LLIBRES
    ->  WHERE ESTOC <= 5;
+---------------+-----------+-------+
| TITOL         | EDITORIAL | ESTOC |
+---------------+-----------+-------+
| Aprendre PHP  | Segle XXI |     3 |
| Martin Fierro | Planeta   |     3 |
+---------------+-----------+-------+
```

5. Ara procedirem a crear el procediment amb:

a. El nom del procediment serà *sp01_Llibre_Aprop_Estoc()* i

b. La sentència SQL que hem obtingut abans:

```
SELECT TITOL, EDITORIAL, ESTOC
FROM   LLIBRES
WHERE  ESTOC <= 5;
```

6. I finalment, procedim a crear el procediment.

```
/* Canviem a la base de dades LLIBRERIA per
*  assegurar-nos que és la base de dades seleccionada. */
USE LLIBRERIA;
```

```
-- Modifiquem el delimitador de sentències a //
DELIMITER //

/* Procedim a esborrar el procediment que volem
*  crear per assegurar-nos que el creem des de zero. */
DROP PROCEDURE IF EXISTS sp01_Llibre_Aprop_Estoc//

/* Procedim a crear el nou procediment amb la
*  clàusula CREATE PROCEDURE seguida del nom del procediment
*  i la definició de paràmetres si cal. En aquest cas no cal. */
CREATE PROCEDURE sp01_Llibre_Aprop_Estoc()

-- La clàusula BEGIN indica l'inici del procediment.
  BEGIN

-- A partir d'aquí desenvolupem el procediment en si.

    SELECT TITOL, EDITORIAL, ESTOC
    FROM   LLIBRES
    WHERE  ESTOC <= 5;

-- La clàusula END indica el final del procediment.
  END //

-- Modifiquem el delimitador de sentències a l'estàndard que és ;
DELIMITER ;
```

Fitxer: sp01_Llibre_Aprop_Estoc.sql

Execució

```
mysql>  CALL sp01_Llibre_Aprop_Estoc;
+---------------+-----------+-------+
| TITOL         | EDITORIAL | ESTOC |
+---------------+-----------+-------+
| Aprendre PHP  | Segle XXI |     3 |
| Martin Fierro | Planeta   |     3 |
+---------------+-----------+-------+
```

Exercici 1

Crea un procediment anomenat **sp02_exercici_llistaActors** que torni el **nom** i **any de naixement** de tots els **actors** que són **homes** de la base de dades **videoclub**.

Solució: sp02_exercici_llistaActors.sql

Variables als procediments emmagatzemats

Les variables del sistema i les variables definides per l'usuari es poden utilitzar en programes emmagatzemats, de la mateixa manera que es poden utilitzar fora del context del programa emmagatzemat. A més, els programes emmagatzemats poden utilitzar DECLARE per definir variables locals, i les rutines emmagatzemades (procediments i funcions) es poden declarar per tenir paràmetres que comuniquen valors entre la rutina i l'usuari que el crida.

DECLARE *nom_var* — Declaració de variables locals

La comanda **DECLARE** té la següent sintaxi:

```
DECLARE nom_var [, nom_var] …
    tipus_de_dada, [DEFAULT valor]
```

On *tipus_de_dada* pot ser:

```
tipus_dada:
INT[(mida)] [UNSIGNED]
| INTEGER[(mida)] [UNSIGNED]
| FLOAT[(mida,decimals)] [UNSIGNED]
| DECIMAL[(mida[,decimals])] [UNSIGNED]
| DATE
| TIME
| DATETIME
| YEAR
| CHAR[(mida)]
| VARCHAR(mida)
| BINARY[(mida)]
```

Per més informació visita el capítol ***Chapter 11 Data Types***.

Els procediments emmagatzemats poden **rebre** i **retornar** informació; per això es fan servir paràmetres, d'**entrada** i **sortida**, respectivament.

Paràmetres d'entrada

Vegem els primers. Els **paràmetres d'entrada** possibiliten passar informació a un procediment.

Perquè un procediment emmagatzemat admeti paràmetres d'entrada s'han de declarar variables com a paràmetres en crear-lo. La sintaxi és:

```
CREATE PROCEDURE nom_procediment(
        IN  nom_variable_entrada tipus_de_dada)
  BEGIN
    sentències
  END //
```

Els paràmetres d'entrada es defineixen després del **nom del procediment**, i previ al nom del paràmetre se li antecedeix la paraula clau `IN`. Els paràmetres són locals al procediment, és a dir, existeixen només dins de procediment. Si es volen declarar diversos paràmetres per procediment, cal separar-los per comes.

Quan el procediment és executat, se n'han d'explicitar valors per a cada un dels paràmetres i en l'ordre que van ser definits.

Els paràmetres d'entrada poden ser dades de qualsevol `tipus_de_dada`. Per més informació visita el capítol ***Chapter 11 Data Types***.

Exemples

Per veure un exemple amb més detall seguim els següents pasos:

Per crear el procediment emmagatzemat anomenat *sp03_Llibres_Autor* que rep un paràmetre d'entrada de tipus ***varchar(30)***, seguim els següents pasos:

1. A qualsevol editor de text, escrivim les següents línies de codi.

```
/* Canviem a la base de dades LLIBRERIA per
*  assegurar-nos que és la base de dades seleccionada.
*/
USE LLIBRERIA;

-- Modifiquem el delimitador de sentències a //
DELIMITER //

/* Procedim a esborrar el procediment que volem
*  crear per assegurar-nos que el creem des de zero. */
DROP PROCEDURE IF EXISTS sp03_Llibres_Autor//

/* Procedim a crear el nou procediment amb la
*  clàusula CREATE PROCEDURE seguida del nom
*  del procediment i la definició de paràmetres
*  si cal. En aquest cas creem un paràmetre
*  d'entrada (IN) que anomenem
*  p_autor i del tipus varchar(30). */
CREATE PROCEDURE sp03_Llibres_Autor(
      IN  p_autor varchar(30))

-- La clàusula BEGIN indica l'inici del procediment.
  BEGIN

-- A partir d'aquí desenvolupem el procediment en si.
    SELECT TITOL, EDITORIAL, PREU
    FROM   LLIBRES
    WHERE  AUTOR = p_autor;
-- La clàusula END indica el final del procediment.
  END //

-- Modifiquem el delimitador de sentències
```

```
-- a l'estàndard que és ;
DELIMITER ;
```

Fitxer: sp03_Llibres_Autor.sql

2. Per crear el procediment a la base de dades, copiem totes les línies que hem escrit a l'editor de text, i les enganxem a la consola.

```
mysql> DELIMITER //
mysql> USE LLIBRERIA//
mysql> DROP PROCEDURE IF EXISTS sp03_Llibres_Autor//
mysql> CREATE PROCEDURE sp03_Llibres_Autor(
    ->      IN  p_autor varchar(30))
    ->      BEGIN
    ->          SELECT TITOL, EDITORIAL, PREU
    ->          FROM   LLIBRES
    ->          WHERE  AUTOR = p_autor;
    ->      END //
mysql> DELIMITER ;
```

3. Per crear executar el procediment que acabem de crear a la base de dades fem una crida la procediment, des de la consola de ***MYSQL***.

```
mysql>  CALL sp03_Llibres_Autor('Lewis Carroll');
+-----------------------------------+-----------+-------+
| TITOL                             | EDITORIAL | PREU  |
+-----------------------------------+-----------+-------+
| Alícia al país de les meravelles  | EMECE     | 20.00 |
| Alícia al país de les meravelles  | Plaça     | 35.00 |
+-----------------------------------+-----------+-------+
```

Activitat 2: Treballadors amb fills (del ClickEdu)

Problema 1

```
/*  Una empresa emmagatzema les dades dels seus
**  treballadors en una taula anomenada "TREBALLADORS"
```

```
**  a la base de dades anomenada LLIBRERIA.
**  Consta de 8 activitats.
**  Cal que lliuris un fitxer anomenat:
**  sp04_problema_001_CognomNom.sql
**  amb totes les comandes que es demanen a cada apartat. */
```

Fitxer: sp04_problema_001.sql

Variables en els procediments.

Per declarar una variable dins d'un procediment emmagatzemat utilitzarem la següent estructura:

```
DECLARE nom_variable tipusdedada(mida) DEFAULT valor;
```

Per exemple

```
DECLARE suma float DEFAULT 0;
```

On:

- S'especifica el nom de la variable després de la paraula clau `DECLARE`.
- S'ha d'especificar el tipus de variable i la seva mida (són els mateixos tipus que utilitza MySQL).
- El valor per defecte quan es declara una variable és `NULL`. Es pot canviar emprant la paraula clau `DEFAULT`.
- Per guardar el resultat d'una consulta en una variable utilitzem la paraula clau `INTO` dins la `SELECT`.
- L'àmbit de les variables declarades dins d'un procediment es perd quan s'acaba el procediment. Per definir una variable global s'ha de col·locar el símbol `@` davant de la variable.

Per exemple: Per crear un procediment que mostri per pantalla la suma de sous actual i la suma de sous després d'aplicar un increment del 3%. La suma de sous futura ha de quedar com una variable global.

En qualsevol editor de text, escrivim les següents línies de codi.

```
/* Canviem a la base de dades empresa per
*  assegurar-nos que és la base de dades seleccionada. */
```

```
USE empresa;

-- Modifiquem el delimitador de sentències a //
DELIMITER //

/* Procedim a esborrar el procediment que volem
*  crear per assegurar-nos que el creem des de zero. */
DROP PROCEDURE IF EXISTS sp05_SumaSous//

/* Procedim a crear el nou procediment amb la
*  clàusula CREATE PROCEDURE seguida del nom del procediment
*  i la definició de paràmetres si cal. En aquest cas no cal. */
CREATE PROCEDURE sp05_SumaSous()

-- La clàusula BEGIN indica l'inici del procediment.
   BEGIN

-- A partir d'aquí desenvolupem el procediment en si.

-- Declaració de variables
     DECLARE suma float;
     DECLARE sumafutura float DEFAULT 0;

-- Inicialització de variables
     SET suma = 0;

-- Primera sentència SQL amb la que obtenim la suma
-- dels sous de tots els treballadors.
     SELECT SUM(SOU_TREB)   INTO suma
     FROM TREBALLADORS;

-- Segona sentència SQL amb la que creem una cadena amb
-- el text "Suma de sous actuals: " i la suma dels sous
-- de tots els treballadors, que hem obtingut abans. I
-- a més a més li assignem l'alias Actual.
     SELECT  CONCAT("Suma de sous actuals: ",
                    suma)
             AS Actual;

-- Tercera sentència SQL amb la que obtenim el valor que
-- ens demanen, la suma dels sous de tots els treballadors
-- amb un augment del 3%. I el resultat l'assignem a la
-- variable @sumafutura.
     SET @sumafutura = suma * 1.03;

-- Quarta sentència SQL amb la que creem una cadena amb
-- el text "Suma de sous amb l'augment: " i la suma dels sous
```

```
-- de tots els treballadors amb l'augment del 3%, que hem
-- obtingut abans. I a més a més li assignem l'alias Futur.
     SELECT  CONCAT("Suma de sous amb l'augment: ",
                    @sumafutura)
             AS Futur;
-- La clàusula END indica el final del procediment.
  END //

-- Modifiquem el delimitador de sentències a l'estàndard que és ;
DELIMITER ;
```

Fitxer: sp05_SumaSous.sql

```
mysql> USE empresa;
mysql> DELIMITER //
mysql> DROP PROCEDURE IF EXISTS sp05_SumaSous//
mysql> CREATE PROCEDURE sp05_SumaSous()
    ->   BEGIN
    ->     DECLARE suma float;
    ->     DECLARE sumafutura float DEFAULT 0;
    ->     SET suma = 0;
    ->     SELECT SUM(SOU_TREB)  INTO suma
    ->      FROM TREBALLADORS;
    ->     SELECT   CONCAT( "Suma de sous actuals: ", suma)
    ->                AS Actual;
    ->     SET @sumafutura = suma * 1.03;
    ->   SELECT CONCAT("Suma de sous després de l'augment: ",
    ->                  @sumafutura)
    ->               AS Futur;
    ->   END//
mysql> DELIMITER ;
```

```
mysql> CALL sp05_SumaSous;
+------------------------------+
```

```
| Actual                                |
+---------------------------------------+
| Suma de sous actuals: 22677.7 |
+---------------------------------------+

+-----------------------------------------------------+
| Futur                                               |
+-----------------------------------------------------+
| Suma de sous després de l'augment: 23358.010078125  |
+-----------------------------------------------------+


mysql> SELECT @sumafutura;

+------------------+
| @sumafutura      |
+------------------+
| 23358.010078125  |
+------------------+

mysql> SELECT suma;

ERROR 1054 (42S22): Unknown column 'suma' in

'field list'


mysql> SELECT @suma;

+-------+
| @suma |
+-------+
|  null |
+-------+
```

Exercici 2

Crea un procediment **sp06_comptaPelisPerActors** que compti totes les pel·lícules (de la base de dades **videoclub**) que ha fet un actor, el nom **(exacte)** del qual li passarem com a paràmetre.

Solució: sp06_comptaPelisPerActors.sql

```
/*      Exercici 2
** Crea el procediment sp06_comptaPelisPerActors
** que compti totes les pel·lícules (de la base
** de dades videoclub) que ha fet un actor, el
** nom (exacte) del qual li passarem com a paràmetre. */
/* mysql> desc ACTORS;
+--------------+----------------------+------+-----+---------+-------+
| Field        | Type                 | Null | Key | Default | Extra |
| id_actor     | smallint(5) unsigned | NO   | PRI | NULL    |       |
| nom_actor    | varchar(30)          | YES  | MUL | NULL    |       |
| nacio_actor  | varchar(20)          | YES  |     | NULL    |       |
| anynaix_actor | smallint(5) unsigned | YES  |     | NULL    |       |
| sexe_actor   | varchar(6)           | YES  |     | NULL    |       |
** mysql> desc ACTORS_PELLICULES;
| Field     | Type                 | Null | Key | Default | Extra
|
+-----------+----------------------+------+-----+---------+------
-+
| id_peli   | smallint(5) unsigned | NO   | PRI | 0       |
|
| id_actor  | smallint(5) unsigned | NO   | PRI | 0       |
|
| papel     | varchar(40)          | YES  |     | NULL    |
|
| principal | tinyint(1)           | YES  |     | NULL    |
|

** Simulacre: si ens passen com a paràmetre el nom_actor="Nicolas
Cage";
** Llavors busquem el seu id_actor a la taula ACTORS
**      SELECT id_actor
**      FROM   ACTORS
**      WHERE  nom_actor="Nicolas Cage";
**      +----------+
**      | id_actor |
**      +----------+
**      |        1 |
**      +----------+
** amb aquest id_actor = 1, llavors podem comptar
** quantes pel·lícules ha fet aquest actor.
**      SELECT COUNT(id_actor)
**      FROM   ACTORS_PELLICULES
**      WHERE  id_actor=1; //id_actor=codi_Actor;
**      +-----------------+
**      | COUNT(id_actor) |
**      +-----------------+
**      |               2 |
**      +-----------------+ */
```

```
-- Ara creem el procediment.

/* Canviem a la base de dades videoclub per
** assegurar-nos que és la base de dades seleccionada. */
USE videoclub;

-- Modifiquem el delimitador de sentències a //
DELIMITER //

/* Procedim a esborrar el procediment que volem
** crear per assegurar-nos que el creem des de zero. */
DROP PROCEDURE IF EXISTS sp06_comptaPelisPerActors//

/* Procedim a crear el nou procediment amb la
** clàusula CREATE PROCEDURE seguida del nom del procediment
** i la definició de paràmetres si cal. En aquest cas
** creem un paràmetre d'entrada (IN) que anomenem
** p_nomActor que és el tipus varchar(30). */
CREATE PROCEDURE sp06_comptaPelisPerActors(
       IN pi_nomActor varchar(30))
-- La clàusula BEGIN indica l'inici del procediment.
  BEGIN
-- A partir d'aquí desenvolupem el procediment en si.
-- Declaració de variables.
     DECLARE codi_Actor smallint;
     DECLARE qtat_Pelis smallint;

/* Primera sentència SQL amb la que obtenim el nom de l'actor
** fent servir el parametre pi_nomActorque ens han passat.
** I el guardem a la variable codi_Actor que hem declarat. */
     SELECT id_actor INTO codi_Actor
     FROM ACTORS
     WHERE nom_actor = pi_nomActor;
/* Segona sentència SQL amb la que comptem la quantitat
** de registres que hi ha a la taula ACTORS_PELLICULES
** que tenen l'id_actor. Fent servir en el filtre (WHERE)
** el parametre codi_Actor que hem guardat. D'aquesta
** manera comptarem només les pel·lícules del "nostre" actor. */

     SELECT COUNT(id_peli) INTO qtat_Pelis
     FROM ACTORS_PELLICULES
     WHERE id_actor=codi_Actor;

     SELECT CONCAT( "L'actor " , pi_nomActor , " ha fet ",
qtat_Pelis, " pelicules!") AS Pel·lícules;


-- La clàusula END indica el final del procediment.
  END //

-- Modifiquem el delimitador de sentències a l'estàndard que és ;
```

```
DELIMITER ;
```

Fitxer: sp06_comptaPelisPerActors.sql

```
mysql> CALL sp06_comptaPelisPerActors("Nicolas
Cage");
+-----------------------------------------+
| Pel·lícules                             |
+-----------------------------------------+
| L'actor Nicolas Cage ha fet 2 pelicules! |
+-----------------------------------------+
```

Paràmetres en els procediments.

Els procediments poden tenir múltiples paràmetres separats per coma (,) que hauran de tenir la següent estructura:

```
MODE nom_parametre tipus_paràmetre(mida)
```

On el **MODE** pot ser:

- **IN**: el mode per defecte. Significa que és un valor que es rep, però que no modificarà el valor de la variable global.
- **OUT**: Significa que no es rep cap valor, però es modificarà el valor de la variable global.
- **INOUT**: Significa que rep un valor i que es modificarà el valor de la variable global.

Per exemple: Crea un procediment que rebi el codi d'un departament i torni la suma dels sous dels seus empleats.

```
/* Modifiquem el delimitador de sentències a // */
DELIMITER //

* Canviem a la base de dades empresa per
** assegurar-nos que és la base de dades seleccionada. */
USE empresa//

/* Procedim a esborrar el procediment que volem
```

```
** crear per assegurar-nos que el creem des de zero. */
DROP PROCEDURE IF EXISTS sp07_sumaSousDepartament//

/* Procedim a crear el nou procediment amb la
** clàusula CREATE PROCEDURE seguida del nom del procediment
** i la definició de paràmetres si cal. En aquest cas
** creem un paràmetre d'entrada(IN) que anomenem
** pi_num_dept que és el tipus smallint i
** creem un paràmetre de sortida (OUT) que anomenem
** po_suma. */
CREATE PROCEDURE sp07_sumaSousDepartament(
      IN  pi_num_dept smallint,
      OUT po_suma float)

-- La clàusula BEGIN indica l'inici del procediment.
  BEGIN

-- A partir d'aquí desenvolupem el procediment en si.
-- Declaració de variables. Si cal.
       SELECT SUM(SOU_TREB)
         INTO po_suma
       FROM   TREBALLADORS
       WHERE  CODI_DEP_TREB=pi_num_dept;

-- La clàusula END indica el final del procediment.
  END //

-- Tornem el delimitador de sentències a l'estàndard que és ;
DELIMITER ;
```

Fitxer: sp07_sumaSousDepartament.sql

```
mysql> set @sumadep10=0;
+-------------+
| @sumadep10  |
+-------------+
|           0 |
+-------------+

mysql> CALL sp07_sumaSousDepartament(10, @sumadep10);

mysql> SELECT @sumadep10;
+-----------------+
| @sumadep10      |
+-----------------+
| 6836.509765625  |
```

```
+----------------+
```

Exercici 3

Crea la funció **sp08_llistaActorsSexe(*cadena*)** que torni tots els actors o actrius de la base de dades videoclub depenent d'un paràmetre d'entrada, que serà una cadena amb dos possibles valors (home,dona).

Solució: sp08_comptaPelisPerActors.sql

Estructures de control en els procediments.

MySQL permet utilitzar diferents estructures de control per millorar els resultats dels procediments:

IF **Estructura condicional.**

L'estructura condicional **IF** implementa una construcció condicional bàsica. I permet executar un bloc d'instruccions si es compleix una condició. La seva sintaxi és:

```
IF condició THEN
   sentències;
END IF;
```

La sentència **IF** pot anar acompanyada d'un bloc de sentències que s'executarien si no es compleix la condició. La seva sintaxi és:

```
IF condició THEN
   sentències si es compleix la condició;
ELSE
   sentències si no es compleix la condició;
```

```
END IF;
```

També podem tenir sentències `ELSEIF` niades (***nested***) per un conjunt de condicions.

```
IF condició1 THEN
   sentències SI es compleix la condició1;
   ELSEIF condició2 THEN
       sentències si NO es compleix la condició1
       I SI que es compleix la condició2
     ELSEIF condició3 THEN
        sentències si NO es compleix la condició1,
        I si NO es compleix la condició2,
        I SI que es compleix la condició3
ELSE
   sentències si NO es compleix la condició1,
        I si NO es compleix la condició2
        I si NO es compleix la condició3
END IF;
```

Per exemple: Procediment que rep el codi d'un empleat i torna el seu nivell de sou. Tenint en compte el següent criteri:

- Un sou inferior a **1.000 €** té un **nivell Baix**.
- Un sou entre **1.001 €** i **2.000 €** té un **nivell Mitjà**.
- Un sou superior a **2.001 €** té un **nivell Alt**.

```
DELIMITER //
USE empresa//
DROP PROCEDURE IF EXISTS sp09_nivellSou//
```

```
CREATE PROCEDURE sp09_nivellSou(
  IN  pi_coditreballador integer,
  OUT po_nivell varchar(12))
  BEGIN
    DECLARE salari float DEFAULT 0;
      SELECT SOU_TREB
        INTO salari
      FROM   TREBALLADORS
      WHERE  CODI_TREB=pi_coditreballador;
   IF  salari IS NULL THEN
      SET po_nivell = "No existeix!";
   ELSE
      IF  salari < 1000 THEN
            SET po_nivell =  "Baix";
         ELSEIF  salari < 2000 THEN
            SET po_nivell =  "Mitjà";
      ELSE
            SET po_nivell =  "Alt";
      END IF;
   END IF;
END//


DELIMITER ;
```

```
mysql> CALL sp09_nivellSou(7499, @nivelltreb);
```

```
mysql> SELECT @nivelltreb;
+-------------+
| @nivelltreb |
+-------------+
| Alt         |
+-------------+
mysql> CALL sp09_nivellSou(111, @nivelltreb);

mysql> SELECT @nivelltreb;
+---------------+
| @nivelltreb   |
+---------------+
| No existeix!  |
+---------------+
```

Per exemple: Procediment (`sp10_NomINivellSou`) que rep el **codi d'un empleat** i torna el **nom del treballador** i seu **nivell de sou**. Tenint en compte el següent criteri:

- Un sou inferior a **1.000 €** té un **nivell Baix**.
- Un sou entre **1.001 €** i **2.000 €** té un **nivell Mitjà**.
- Un sou superior a **2.001 €** té un **nivell Alt**.

CASE — Estructura condicional

L'estructura condicional **CASE** permet executar diferents blocs de codi depenent dels possibles valors que pugui prendre una variable (semblant a **switch** en JAVA).

La seva sintaxi és:

```
CASE  variable
   WHEN valor1 THEN sentències quan variable=valor1;
   [WHEN valor2 THEN sentències quan variable=valor2;]
```

```
    ...
    [ELSE sentències quan variable no és cap dels
valor1, valor2,...;]
END CASE;
```

Per exemple: Desenvolupa un procediment que rebi el codi d'un empleat i torni l'**increment** del seu sou. Tenint en compte el següent criteri:

- Els **President** pugen un **10%** el seu sou.
- Els **Analista** un **15%**.
- Els **Director** un **5%**.
- Els **Venedor** un **3%**.
- La **resta** de treballadors un **4%**

```
DELIMITER //

DROP PROCEDURE IF EXISTS 07_incrementEmpleatSegonsFuncio//
CREATE PROCEDURE 07_incrementEmpleatSegonsFuncio(
    IN  coditreballador integer,
    OUT increment float)

  BEGIN
    DECLARE salari float DEFAULT 0;
    DECLARE ofici varchar(20);
      SELECT SOU_TREB, OFICI_TREB
        INTO salari, ofici
      FROM   TREBALLADORS
      WHERE  CODI_TREB=coditreballador;
      CASE ofici
         WHEN "President" THEN
              SET increment =  salari * 0.1;
         WHEN "Analista" THEN
              SET increment =  salari * 0.15;
         WHEN "Director" THEN
              SET increment =  salari * 0.05;
         WHEN "Venedor" THEN
              SET increment =  salari * 0.03;
         ELSE
```

```
                SET increment =  salari * 0.04;
        END CASE;
    END//

DELIMITER ;
```

```
mysql> CALL 05_incrementEmpleatSegonsFuncio(7839, @increment);

mysql> SELECT @increment;
+--------------------+
| @increment         |
+--------------------+
| 390.65802001953125 |
+--------------------+
```

WHILE — Estructura repetitiva

L'estructura repetitiva **WHILE** repeteix un bloc de codi mentre es compleixi la condició (***condició de permanència***). La condició s'avalua al principi de cada iteració, per tant, potser no s'arriba a executar cap vegada.

La seva sintaxi és:

```
WHILE condició
    DO
        sentències
END WHILE;
```

REPEAT **Estructura repetitiva.**

L'estructura repetitiva **REPEAT** repeteix un bloc de codi fins que es compleixi una condició (***condició de sortida***). La condició s'avalua al final de cada iteració, per tant, com a mínim s'executarà una vegada.

La seva sintaxi és:

```
REPEAT
    sentències
    UNTIL  condició
END REPEAT;
```

SHOW PROCEDURE STATUS **Llistar procediments existents**

Aquesta comanda és una extensió MySQL. Retorna les característiques d'un procediment emmagatzemat, com la base de dades, el nom, el tipus, el creador, les dates de creació i modificació i la informació del joc de caràcters. Una declaració similar, **SHOW FUNCTION STATUS**, mostra informació sobre les funcions emmagatzemades (Per més informació vegeu la secció 13.7.5.21, **SHOW FUNCTION STATUS Statement**).

```
SHOW PROCEDURE STATUS
   [LIKE 'patró' | WHERE condició];
```

Per llistar els procediments que pertanyen a una base de dades determinada podem executar:

```
mysql>  SHOW PROCEDURE STATUS
```

```
        >  WHERE Name LIKE 'sp0%' &&
db="LLIBRERIA";
+-----------+------------------------+-----------+----------------+-...
| Db        | Name                   | Type      | Definer        | ...
+-----------+------------------------+-----------+----------------+-...
| LLIBRERIA | sp01_Llibre_Aprop_Estoc | PROCEDURE | root@localhost | ...
| LLIBRERIA | sp02_Llibres_Autor      | PROCEDURE | root@localhost | ...
| LLIBRERIA | sp03_Llibres_Autor      | PROCEDURE | root@localhost | ...
+-----------+------------------------+-----------+----------------+-...
```

També podem llistar els procediments que pertanyen a una BD en concret amb un format més entenedor fent servir el paràmetre \G.

```
mysql>  SHOW PROCEDURE STATUS
    >  WHERE name LIKE 'sp0%' && db="LLIBRERIA" \G;
*************************** 1. row ***************************
                  Db: LLIBRERIA
                Name: sp01_Llibre_Aprop_Estoc
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2019-11-02 05:30:59
             Created: 2019-11-02 05:30:59
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
*************************** 2. row ***************************
                  Db: LLIBRERIA
                Name: sp02_Llibres_Autor
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2019-11-02 05:30:59
             Created: 2019-11-02 05:30:59
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
*************************** 3. row ***************************
                  Db: LLIBRERIA
                Name: sp03_Llibres_Autor
```

```
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2019-11-02 05:30:59
             Created: 2019-11-02 05:30:59
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
3 rows in set (0.00 sec)
```

I fins i tot, podem cercar un procediment en concret pel nom.

```
mysql>  SHOW PROCEDURE STATUS
    >  WHERE name = 'sp01_Llibre_Aprop_Estoc';
+-----------+-------------------------+-----------+-----------...
| Db        | Name                    | Type      | Definer   ...
+-----------+-------------------------+-----------+-----------...
| LLIBRERIA | sp01_Llibre_Aprop_Estoc | PROCEDURE | root@local...
+-----------+-------------------------+-----------+-----------...
```

SHOW CREATE PROCEDURE Mostrar el contingut d'un procediment

Aquesta comanda és una extensió MySQL. Retorna la cadena exacta que es pot utilitzar per tornar a crear el procediment emmagatzemat demanat. Una declaració similar, **SHOW CREATE FUNCTION**, mostra informació sobre les funcions emmagatzemades (Per més informació 13.7.5.10, SHOW CREATE FUNCTION Statement").

```
SHOW CREATE PROCEDURE [NOM_PROCEDIMENT] [\G];
```

Si el que volem és llistar els contingut d'un procediment fem

```
mysql>  SHOW CREATE PROCEDURE sp01_Llibre_Aprop_Estoc;
+------------------------+---------+-------------------------...
```

```
| Procedure                | sql_mode | Create Procedure          ...
+--------------------------+----------+--------------------------...
| sp01_Llibre_Aprop_Estoc |          | CREATE DEFINER=`root`@`lo...
BEGIN                                                              ...
   SELECT TITOL, EDITORIAL, ESTOC                                  ...
   FROM LLIBRES                                                    ...
   WHERE ESTOC <= 5;                                               ...
  END | utf8                 | utf8_general_ci      | latin1_swe...
+--------------------------+----------+--------------------------...
```

Exercici 4

Activitat 3: Activitats de procediments emmagatzemats

Sobre la base de dades **videoclub**.

1. Dissenya un procediment que rebi un codi de pel·lícula i mostri per pantalla el nom de la pel·lícula i els actors/actrius que hi actuen, juntament amb el paper que interpreten.

Solució: **PardoJoan_Act_03_ProcEmm_MySQL_Apartat_001.sql**

Cursors en MySQL

Els CURSORS permeten gestionar un conjunt de resultats dins d'un procediment emmagatzemat. ***MySQL*** admet CURSORS dins de programes emmagatzemats. Els CURSORS tenen aquestes propietats:

- **Asensitive** (No Sensible): el servidor pot o no fer una còpia de la taula de resultats.
- **Només de lectura**: no modificable
- **No controlable**: només es pot recórrer en una direcció i no poden saltar-se les files.

Per fer-l'ho servir cal seguir els següents passos:

1. Declarar un cursor i associar-lo a una consulta que torna més d'un registre.
2. Obrir un cursor per poder recórrer tots els registres que conté.
3. Declarar un controlador per finalizar el recorregut del cursor.
4. Obtenir el següent element del cursor.
5. Tancar el cursor.

`DECLARE nomCursor CURSOR` **Declarar el cursor.**

Per declarar un cursor fem servir la sintaxi:

```
DECLARE nomCursor CURSOR FOR
    SELECT ...;
```

Les declaracions del CURSORS han d'aparèixer abans de les **declaracions** del manipulador i després de les **declaracions de variables** i condicions.

`OPEN nomCursor` **Obrir el cursor.**

A l'obrir el `CURSOR` aquest situa un **punter** a la primera fila del resultat de la consulta. Per obrir el cursor fem servir la sintaxi:

```
OPEN nomCursor;
```

`FETCH nomCursor` **Extreure els valors del cursor.**

Aquesta instrucció recupera la següent fila per a la instrucció `SELECT` associada al **cursor** especificat (que ha d'estar obert) i avança el **punter** del **cursor**. Si existeix una fila, les columnes obtingudes s'emmagatzemaran a les variables anomenades. El nombre de columnes recuperades per la instrucció `SELECT` ha de coincidir amb el nombre de variables de sortida especificades a la instrucció `FETCH`. Si no hi ha més files disponibles, es produeix una condició `No Data` amb el valor `SQLSTATE "02000"`. Per detectar aquesta condició, podeu configurar un controlador (o per a una condició `NOT FOUND`).(Per més informació 13.6.6, "Cursors"").

Per extreure valors d'un cursor i avançar-lo a la següent posició fem servir la sintaxi:

```
FETCH [[NEXT] FROM] nomCursor
    INTO nomVariable1 [, nomVariable2] ...
```

`CLOSE nomCursor` **Tancar el cursor.**

Per tancar un cursor fem servir la sintaxi:

```
CLOSE nomCursor
```

Controladors

Durant l'execució del programa emmagatzemat es poden presentar **condicions** que requereixin una gestió especial. Per exemple sortir de l'actual bloc de programa o continuar amb l'execució.

Per poder gestionar cadascuna d'aquestes **condicions**, cal haver declarar prèviament i de manera específica un **controladors** per cada condició que volem tractar. I, si aquesta condició es presenta, llavors la comanda especificada al controlador s'executarà. La sentència a executar tant pot ser una comanda senzilla, com una declaració composta. En aquest segon cas caldrà escriure-la entre les clàusules `BEGIN ... END`.

Els **controladors** es poden definir per a:

1) **condicions generals**, o per
2) **advertències o excepcions**, o per
3) **condicions específiques**, i per
4) codis d'**error particulars**.

`DECLARE ... HANDLER` **Declarar un controlador**

Les declaracions dels **controladors** cal que aparèguin després de declaracions de les variable o de les **controladors** .

Els valors de l'acció del controlador (`ACCIÓ_CONTROLADOR`) indica com es seguirà després de l'execució de la sentència del **controlador**. I aquests valors poden ser:

- `CONTINUE`: Continua l'execució del programa actual.
- `EXIT`: L'execució finalitza per a la `BEGIN ... END` declaració composta en què es declara el **controlador**. Això és cert, fins i tot si la condició es produeix en un bloc interior.

I el valor de la CONDICIÓ_ACTIVA indica la **condició específica** o **classe de condicions** que activa el **controlador**.

```
DECLARE ACCIÓ_CONTROLADOR HANDLER
    FOR  [, CONDICIÓ_ACTIVA] ...
    sentències

ACCIÓ_CONTROLADOR:
    CONTINUE
  | EXIT

CONDICIÓ_ACTIVA:
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

Per exemple, com declarar un controlador de final de la llista de resultats fem servir la sintaxi:

```
DECLARE CONTINUE HANDLER
   FOR NOT FOUND
      SET final = 1;
```

On ***final*** serà una variable a la que cridarem per comprovar si el cursor està al **final** de la consulta.

Per exemple: Procediment que calcula l'increment global de sous. Tenint en compte el següent criteri:

- Els **President** pugen un **10%** el seu sou.
- Els **Analista** un **15%**.
- Els **Director** un **5%**.
- Els **Venedor** un **3%**.
- La **resta** de treballadors un **4%**

```
  DELIMITER //
  DROP PROCEDURE IF EXISTS 06 incrementEmpleatSegonsFuncio//
  CREATE PROCEDURE 06 incrementEmpleatSegonsFuncio(
            OUT sumaincrement float)
  BEGIN
     DECLARE salari float DEFAULT 0;
     DECLARE funcio varchar(20);
     DECLARE final int DEFAULT 0; -- Igual 0 = False

/*
mysql> SELECT TRUE, true, FALSE, false;
    ->          1,    1,     0,    0
*/

     DECLARE elcursor CURSOR FOR
        SELECT SOU_TREB, OFICI_TREB
        FROM   TREBALLADORS;

     DECLARE CONTINUE HANDLER FOR NOT FOUND SET final = 1; -- True

     OPEN elcursor;
     SET sumaincrement =  0;
     elbucle:LOOP
       FETCH elcursor INTO salari, funcio;

       IF final = 1 THEN
          LEAVE elbucle;
       END IF;

       CASE funcio
```

```
        WHEN "President" THEN
            SET sumaincrement =  sumaincrement + (salari * 0.1);
        WHEN "Analista" THEN
            SET sumaincrement =  sumaincrement + (salari *  0.15);
        WHEN "Director" THEN
            SET sumaincrement =  sumaincrement + (salari *  0.05);
        WHEN "Venedor" THEN
            SET sumaincrement =  sumaincrement + (salari * 0.03);
        ELSE
            SET sumaincrement =  sumaincrement + (salari * 0.04);
      END CASE;
    END LOOP elbucle;
  CLOSE elcursor;

  END//

DELIMITER ;
```

```
mysql> CALL 06_incrementEmpleatSegonsFuncio(@sumaincrement);

mysql> SELECT @sumaincrement;
+--------------------+
| @sumaincrement     |
+--------------------+
| 1678.0711669921875 |
+--------------------+
```