**Procedimiento**

**DELIMITER $$**

**DROP** **PROCEDURE** **IF** **EXISTS** pago

**CREATE** **PROCEDURE** pago( **IN** formaPago **varchar**(20), **out** máximo **int** )

**BEGIN**

**SELECT** **MAX**(p.total) **into** maximo

**FROM** pago p

**WHERE** p.forma\_pago = formaPago;

**END**; $$

**DELIMITER ;**

**call** pago("PayPal", @maximo);

Función

**DELIMITER $$**

**DROP** **FUNCTION** **IF** **EXISTS** producto\_menor\_valor\_fabricante

**CREATE** **FUNCTION** producto\_menor\_valor\_fabricante(fabricante **VARCHAR**(20))

**RETURNS** **INT** **UNSIGNED** **DETERMINISTIC** # reads sql / no sql

**BEGIN**

**DECLARE** valorMin **INT** **UNSIGNED**;

**SELECT** **MIN**(p.precio) **INTO** valorMin

**FROM** producto p

**INNER** **JOIN** fabricante f

**ON** p.id\_fabricante = f.id

**WHERE** f.nombre = fabricante;

**RETURN** valorMin;

**END** ;

**DELIMITER ;**

**SELECT** producto\_menor\_valor\_fabricante('Asus');

Trigger

**DELIMITER $$**

**DROP** **TRIGGER** **IF** **EXISTS** trigger\_check\_nota\_before\_update$$

**CREATE** **TRIGGER** trigger\_check\_nota\_before\_update

**BEFORE** **UPDATE**

**ON** alumnos **FOR** **EACH** **ROW**

**BEGIN**

**IF** **NEW**.nota < 0 **THEN**

**set** **NEW**.nota = 0;

**ELSEIF** **NEW**.nota > 10 **THEN**

**set** **NEW**.nota = 10;

**END** **IF**;

**END**

**delimiter ;**

**UPDATE** alumnos **SET** nota = -4 **WHERE** id = 3;

Transacción

**DELIMITER $$**

**DROP** **PROCEDURE** **IF** **EXISTS** comprar\_entrada $$

**CREATE** **PROCEDURE** comprar\_entrada(

**IN** nif **VARCHAR**(9),

**IN** id\_cuenta **INT** **UNSIGNED**,

**IN** id\_butaca **INT** **UNSIGNED**,

**OUT** error **INT** **UNSIGNED**)

**BEGIN**

**DECLARE** **EXIT** HANDLER **FOR** 1264

**BEGIN**

**SET** error = 1;

**SELECT** error, 'Out of range value' **as** 'ERROR TYPE';

**ROLLBACK**;

**END**;

**DECLARE** **EXIT** HANDLER **FOR** 1062

**BEGIN**

**SET** error = 1;

**SELECT** error, 'Duplicate entry for PRIMARY KEY' **as** 'ERROR TYPE';

**ROLLBACK**;

**END**;

**SET** error = 0;

**START** **TRANSACTION**;

**UPDATE** cuentas **SET** saldo = saldo - 5 **WHERE** cuentas.id\_cuenta = id\_cuenta;

**INSERT** **INTO** entradas **VALUES** (id\_butaca, nif);

**SELECT** error;

**COMMIT**;

**END** $$

**DELIMITER ;**

**CALL** comprar\_entrada('18447807J', 5, 8, @error);

-- ----------------------------------------------

**while** valor\_inicial>=uno DO

**INSERT** **INTO** ejercicio **values** (valor\_inicial);

**set** valor\_inicial=valor\_inicial-uno;

**end** **while**;

-- ----------------------------------------------

**repeat**

**INSERT** **INTO** ejercicio **values** (valor\_inicial);

**set** valor\_inicial=valor\_inicial-uno;

UNTIL valor\_inicial<uno

**end** **repeat** ;

-- ----------------------------------------------

bucle: **loop**

**INSERT** **INTO** ejercicio **values** (valor\_inicial);

**set** valor\_inicial=valor\_inicial-uno;

**if** valor\_inicial<uno **then**

**leave** bucle;

**end** **if**;

**end** **loop** ;

-- ----------------------------------------------

**DROP** **PROCEDURE** **IF** **EXISTS** actualizar\_columna\_edad;

**CREATE** **PROCEDURE** actualizar\_columna\_edad()

**BEGIN**

**DECLARE** done **INT** **DEFAULT** 0;

**DECLARE** id\_a **INT**;

**DECLARE** edad **INT**;

**DECLARE** fnac **DATETIME**;

**DECLARE** cur1 **CURSOR** **FOR** **SELECT** id, fecha\_nacimiento **FROM** alumno;

**DECLARE** **CONTINUE** HANDLER **FOR** **NOT** **FOUND** **SET** done = 1;

**OPEN** cur1;

bucle: **LOOP**

**FETCH** cur1

**INTO** id\_a,

fnac;

**SET** edad = 0;

**IF** done = 1 **THEN**

**LEAVE** bucle;

**END** **IF**;

**START** **TRANSACTION**;

**UPDATE** alumno

**SET** edad = calcular\_edad(fnac)

**WHERE** id = id\_a;

**COMMIT**;

**END** **LOOP** bucle;

**CLOSE** cur1;

**END** $$

**DELIMITER ;**

**CALL** actualizar\_columna\_edad();

-- ----- --------------------------------------

**DELIMITER $$**

**DROP** **PROCEDURE** **IF** **EXISTS** crear\_lista\_emails\_alumnos $$

**CREATE** **PROCEDURE** crear\_lista\_emails\_alumnos(**OUT** lista **VARCHAR**(300))

**BEGIN**

**DECLARE** done **INT** **DEFAULT** 0;

**DECLARE** mail **VARCHAR**(30);

**DECLARE** cur1 **CURSOR** **FOR**

**SELECT** email

**FROM** alumno;

**DECLARE** **CONTINUE** HANDLER **FOR** **NOT** **FOUND** **SET** done = 1;

**OPEN** cur1;

bucle: **LOOP**

**FETCH** cur1

**INTO** mail;

**IF** done = 1 **THEN**

**LEAVE** bucle;

**END** **IF**;

**START** **TRANSACTION**;

**SET** lista = **CONCAT\_WS**(';',lista, mail);

**COMMIT**;

**END** **LOOP** bucle;

**CLOSE** cur1;

**END** $$

**DELIMITER ;**

**CALL** crear\_lista\_emails\_alumnos(@lista);

**SELECT** @lista;

# 1)Se desea tener un procedimiento almacenado tal que dados un nombre y apellidos del empleado, compruebe si este empleado es jefe de un almacén.

DELIMITER $$

DROP FUNCTION IF EXISTS es\_jefe $$

CREATE FUNCTION es\_jefe(nombre VARCHAR(45), apellidos VARCHAR(45))

RETURNS BOOLEAN DETERMINISTIC

BEGIN

DECLARE id TINYINT UNSIGNED;

SET id = (SELECT e.id\_empleado

FROM empleado e

WHERE e.nombre = nombre

AND e. apellidos = apellidos);

IF id IN (SELECT a.id\_empleado\_jefe

FROM almacen a) THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END $$

DELIMITER ;

SELECT es\_jefe('Ringo', 'Rooksby');

SELECT es\_jefe('Ada', 'Byron');

# 2)En caso afirmativo deberemos recuperar todos los empleados de ese almacén.

# 3)De cada empleado de ese almacén calcularemos el tiempo medio de devolución de los alquileres de ese empleado.

# 4)Al final del procedimiento devolveremos la suma de todos los tiempos medios de devolución de alquileres(apartado 3)

#

DELIMITER $$

DROP PROCEDURE IF EXISTS tiempo\_medio\_devolucion\_por\_jefe $$

CREATE PROCEDURE tiempo\_medio\_devolucion\_por\_jefe(IN nombre VARCHAR(45), IN apellidos VARCHAR(45), OUT sum\_avg\_time INTEGER)

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE id\_jefe TINYINT UNSIGNED;

DECLARE id\_almacen TINYINT UNSIGNED;

DECLARE id\_e TINYINT UNSIGNED;

DECLARE average\_per\_emp INTEGER;

DECLARE num\_empleados INTEGER;

DECLARE empleados\_almacen CURSOR FOR SELECT id\_empleado

FROM empleado e

WHERE e.id\_almacen = id\_almacen;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

SET sum\_avg\_time = 0;

SET id\_jefe = (SELECT e.id\_empleado

FROM empleado e

WHERE e.nombre = nombre

AND e.apellidos = apellidos);

SET id\_almacen = (SELECT a.id\_almacen

FROM almacen a

WHERE a.id\_empleado\_jefe = id\_jefe

LIMIT 1);

SELECT COUNT(e.id\_empleado) INTO num\_empleados

FROM empleado e

WHERE e.id\_almacen = id\_almacen;

OPEN empleados\_almacen;

bucle: LOOP

FETCH empleados\_almacen

INTO id\_e;

SET average\_per\_emp = 0;

IF done = 1 THEN

LEAVE bucle;

END IF;

SELECT AVG(DATEDIFF(a.fecha\_devolucion, a.fecha\_alquiler))

INTO average\_per\_emp

FROM alquiler a

WHERE a.id\_empleado = id\_e

AND a.fecha\_devolucion > a.fecha\_alquiler;

SELECT average\_per\_emp;

IF(average\_per\_emp is not null) THEN

SET sum\_avg\_time = sum\_avg\_time + average\_per\_emp;

SELECT sum\_avg\_time;

ELSE

SET num\_empleados = num\_empleados - 1;

END IF;

END LOOP bucle;

SET sum\_avg\_time = sum\_avg\_time / num\_empleados;

CLOSE empleados\_almacen;

END $$

DELIMITER ;

CALL tiempo\_medio\_devolucion\_por\_jefe('Jon', 'Stephens', @suma);

SELECT @suma;