**Actividades UD7 parte 2 - Extensiones MySQL**

**1. Función. Ranking por precio.**

DROP FUNCTION IF EXISTS fRanking\_por\_precio;

DELIMITER $$

CREATE FUNCTION fRanking\_por\_precio(piProdID INTEGER)

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE numProductosMasCaros INTEGER;

SELECT COUNT(\*) INTO numProductosMasCaros

FROM productos

WHERE precio > (SELECT precio

FROM productos

WHERE prodID = piProdID);

RETURN numProductosMasCaros+1;

END$$

DELIMITER ;

Para crear la vista:

CREATE VIEW vi\_ranking AS

SELECT prodID, nombre, precio, fRanking\_por\_precio(prodID) AS posición\_por\_precio

FROM productos

ORDER BY 4;

Para ver la tabla:

SELECT \* FROM vi\_ranking;

A screen shot of a computer

Description automatically generated

**2. Procedimiento. ranking por precio.**

DROP PROCEDURE IF EXISTS pRanking\_por\_precio;

DELIMITER $$

CREATE PROCEDURE pRanking\_por\_precio (

IN nombreProd VARCHAR(20))

BEGIN

DECLARE iNumProductosMasCaros INTEGER;

DECLARE iCantidad INTEGER;

SELECT COUNT(\*) INTO iCantidad

FROM productos

WHERE nombre = nombreProd;

CASE

WHEN iCantidad = 1 THEN

SELECT COUNT(\*) INTO iNumProductosMasCaros

FROM productos

WHERE precio > (SELECT precio

FROM productos

WHERE nombre = nombreProd);

SELECT iNumProductosMasCaros+1 AS posicion\_por\_precio;

WHEN iCantidad = 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'No se encuentra ningun producto con ese nombre';

WHEN iCantidad > 1 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Hay más de un producto con ese nombre';

END CASE;

END$$

DELIMITER ;

Para llamar al procedimiento:

CALL pRanking\_por\_precio(‘Fresas’);

A screen shot of a computer program

Description automatically generated

**3. Procedimiento. Black Friday**

DROP PROCEDURE IF EXISTS pBlackFriday;

DELIMITER $$

CREATE PROCEDURE pBlackFriday(

IN pAccion VARCHAR(3))

BEGIN

CASE pAccion

WHEN 'ON' THEN

UPDATE gaming

SET precio\_copia = precio, precio = precio+(0.15\*precio), estado = 'en oferta'

WHERE estado = 'disponible';

SELECT \*

FROM gaming

WHERE estado = 'en oferta';

WHEN 'OFF' THEN

UPDATE gaming

SET precio = precio\_copia, precio\_copia = NULL, estado = 'disponible'

WHERE estado = 'en oferta';

SELECT \*

FROM gaming

WHERE estado = 'disponible';

END CASE;

END$$

DELIMITER ;

Para llamar al procedimiento:

CALL pBlackFriday(‘ON’);

A black screen with white text

Description automatically generated

CALL pBlackFriday(‘OFF’);

A screen shot of a computer

Description automatically generated

**4. Función. Otros rankings.**

Ranking por beneficio:

DROP FUNCTION IF EXISTS fRanking\_por\_beneficio;

DELIMITER $$

CREATE FUNCTION fRanking\_por\_beneficio(inProdID INTEGER)

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE posicionBeneficio INTEGER;

SELECT COUNT(\*) INTO posicionBeneficio

FROM productos P1

WHERE (P1.precio-P1.coste) > (SELECT (P2.precio - P2.coste)

FROM productos P2

WHERE P2.prodID = inProdID);

RETURN posicionBeneficio+1;

END$$

DELIMITER ;

Ranking por coste:

DROP FUNCTION IF EXISTS fRanking\_por\_coste;

DELIMITER $$

CREATE FUNCTION fRanking\_por\_coste(inProdID INTEGER)

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE productosCostosos INTEGER;

SELECT COUNT(\*) INTO productosCostosos

FROM productos P1

WHERE P1.coste > (SELECT P2.coste

FROM productos P2

WHERE P2.prodID = inProdID);

RETURN productosCostosos+1;

END$$

DELIMITER ;

**5. Procedimiento. Informe**

Crear primero la función “coste\_menor”

DROP FUNCTION IF EXISTS fRanking\_coste\_menor;

DELIMITER $$

CREATE FUNCTION fRanking\_coste\_menor (

pProdId INTEGER)

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE posicion INTEGER;

SELECT COUNT(\*) INTO posicion

FROM productos

WHERE precio <= (SELECT precio

FROM productos

WHERE prodID = pProdID);

RETURN posicion;

END$$

DELIMITER ;

DROP PROCEDURE IF EXISTS pInforme;

DELIMITER $$

CREATE PROCEDURE pInforme ()

BEGIN

SELECT nombre, prodID AS ‘Producto de precio maximo’, precio

FROM productos

WHERE fRanking\_por\_precio(prodID) = 1;

SELECT nombre, prodID AS ‘Producto de coste mínimo’, coste

FROM productos

WHERE fRanking\_coste\_menor(prodID) = 1;

SELECT nombre, prodID AS ‘Producto de beneficio maximo’, precio-coste AS beneficio

FROM productos

WHERE fRanking\_por\_beneficio(prodID) = 1;

END$$

DELIMITER ;

**1.6. Función. Día de la semana**

DROP FUNCTION IF EXISTS fDiaSemana;

DELIMITER $$

CREATE FUNCTION fDiaSemana(

numDia INTEGER)

RETURNS VARCHAR(15)

DETERMINISTIC

NO SQL

BEGIN

DECLARE diaLetra VARCHAR(15);

CASE

WHEN numDia = 1 THEN SET diaLetra = 'Lunes';

WHEN numDia = 2 THEN SET diaLetra = 'Martes';

WHEN numDia = 3 THEN SET diaLetra = 'Miercoles';

WHEN numDia = 4 THEN SET diaLetra = 'Jueves';

WHEN numDia = 5 THEN SET diaLetra = 'Viernes';

WHEN numDia = 6 THEN SET diaLetra = 'Sabado';

WHEN numDia = 7 THEN SET diaLetra = 'Domingo';

ELSE SET diaLetra = 'Dia incorrecto';

END CASE;

RETURN diaLetra;

END$$

DELIMITER ;

Para llamar a la función:

SELECT fDiaSemana(2); (o cualquier otro número)

**1.7. Función. Quitar tildes**

DROP FUNCTION IF EXISTS fCadena\_sin\_tildes;

DELIMITER $$

CREATE FUNCTION fCadena\_sin\_tildes(

texto VARCHAR(10000))

RETURNS VARCHAR(10000)

DETERMINISTIC

NO SQL

BEGIN

DECLARE textoSinTildes VARCHAR(10000);

SET textoSinTildes = texto;

SET textoSinTildes = REPLACE(textoSinTildes, 'Á', 'A');

SET textoSinTildes = REPLACE(textoSinTildes, 'á', 'a');

SET textoSinTildes = REPLACE(textoSinTildes, 'É', 'E');

SET textoSinTildes = REPLACE(textoSinTildes, 'é', 'e');

SET textoSinTildes = REPLACE(textoSinTildes, 'Í', 'I');

SET textoSinTildes = REPLACE(textoSinTildes, 'í', 'i');

SET textoSinTildes = REPLACE(textoSinTildes, 'Ó', 'O');

SET textoSinTildes = REPLACE(textoSinTildes, 'ó', 'o');

SET textoSinTildes = REPLACE(textoSinTildes, 'Ú', 'U');

SET textoSinTildes = REPLACE(textoSinTildes, 'ú', 'u');

RETURN textoSinTildes;

END$$

DELIMITER ;

Para llamar a la función:

SELECT fCadena\_sin\_tildes(‘María’) 🡪 muy importante poner el texto entre ‘’, si no no funciona!!

**2. BD Centro educativo**

**2.1. Triggers. Antes de insertar nota y antes de actualizar nota.**

Antes de insertar nota:

DROP TRIGGER IF EXISTS tCheck\_nota\_before\_insert;

DELIMITER ||

CREATE TRIGGER tCheck\_nota\_before\_insert

BEFORE INSERT ON alumnado

FOR EACH ROW

BEGIN

IF NEW.nota < 0

THEN

SET NEW.nota = 0;

END IF;

IF NEW.nota > 10

THEN

SET NEW.nota = 10;

END IF;

END||

DELIMITER ;

Antes de actualizar nota:

DROP TRIGGER IF EXISTS tCheck\_nota\_before\_update;

DELIMITER $$

CREATE TRIGGER tCheck\_nota\_before\_update

BEFORE UPDATE ON alumnado

FOR EACH ROW

BEGIN

IF NEW.nota < 0

THEN

SET NEW.nota = 0;

END IF;

IF NEW.nota > 10

THEN

SET NEW.nota = 10;

END IF;

END$$

DELIMITER ;

**2.2. Función. Eliminar espacios y pasar a minúscula.**

DROP FUNCTION IF EXISTS fFormato\_email;

DELIMITER $$

CREATE FUNCTION fFormato\_email(

textoEntrada VARCHAR(10000))

RETURNS VARCHAR(10000)

DETERMINISTIC

NO SQL

BEGIN

DECLARE textoSalida VARCHAR(10000);

SET textoSalida = textoEntrada;

SET textoSalida = REPLACE(textoSalida, ' ', '');

SET textoSalida = LOWER(textoSalida);

RETURN textoSalida;

END$$

DELIMITER ;

Alternativa:

SET textoSalida = LOWER(REPLACE(textoEntrada, ' ', '');

RETURN textoSalida;

**2.3. Procedimiento. Crear email**

DROP PROCEDURE IF EXISTS pCrear\_email;

DELIMITER $$

CREATE PROCEDURE pCrear\_email(

IN nombre VARCHAR(20),

IN apellido1 VARCHAR(20),

IN apellido2 VARCHAR(20),

IN dominio VARCHAR(30),

OUT email VARCHAR(150))

BEGIN

SET email = CONCAT(fFormato\_email(nombre), LEFT(fFormato\_email(apellido1), 3), LEFT(fFormato\_email(apellido2),3), '@', fFormato\_email(dominio));

END$$

DELIMITER ;

**2.4. Trigger. Antes de insertar email.**

DROP TRIGGER IF EXISTS tCrear\_email\_before\_insert;

DELIMITER $$

CREATE TRIGGER tCrear\_email\_before\_insert

BEFORE INSERT ON alumnado

FOR EACH ROW

BEGIN

IF (NEW.email IS NULL)

THEN

CALL pCrear\_email(NEW.nombre, NEW.apellido1, NEW.apellido2, 'alumnado.com', @email); 🡪 para llamar a un procedimiento dentro de un trigger.

SET NEW.email = @email;

END IF;

END$$

DELIMITER ;

**2.5. Trigger. Después de actualizer email.**

Para crear la tabla log\_cambios\_email:

CREATE TABLE log\_cambios\_email(

id INTEGER NOT NULL AUTO\_INCREMENT PRIMARY KEY,

ida INTEGER,

fecha\_hora TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

old\_email VARCHAR(150),

new\_email VARCHAR(150),

CONSTRAINT log\_ida\_fk FOREIGN KEY (ida) REFERENCES alumnado(ida)

);

Para crear el trigger:

DROP TRIGGER IF EXISTS tGuardar\_email\_after\_update;

DELIMITER $$

CREATE TRIGGER tGuardar\_email\_after\_update

AFTER UPDATE ON alumnado

FOR EACH ROW

BEGIN

IF (NEW.email <> OLD.email)

THEN

INSERT INTO log\_cambios\_email (ida, old\_email, new\_email) VALUES

(OLD.ida, OLD.email, NEW.email);

END IF;

END$$

DELIMITER ;

**2.6. Trigger. Después de borrar alumnado.**

Para crear la tabla log\_alumno\_eliminado:

CREATE TABLE log\_alumnado\_eliminado(

id INTEGER NOT NULL AUTO\_INCREMENT PRIMARY KEY,

ida INTEGER,

fecha\_hora TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

nombre VARCHAR(50),

apellido1 VARCHAR(50),

email VARCHAR(150)

);

Para crear el trigger:

DROP TRIGGER IF EXISTS tGuardar\_alumnado\_after\_delete;

DELIMITER $$

CREATE TRIGGER tGuardar\_alumnado\_after\_delete

AFTER DELETE ON alumnado

FOR EACH ROW

BEGIN

INSERT INTO log\_alumnado\_eliminad (ida, nombre, apellido1, apellido2, email) VALUES

(OLD.ida, OLD.nombre, OLD.apellido1, OLD.apellido2, OLD.email);

END$$

DELIMITER ;

**2.7. Trigger. Participaciones N:M**

Trigger de antes del borrado:

DROP TRIGGER IF EXISTS tBefore\_delete\_matricula;

DELIMITER $$

CREATE TRIGGER tBefore\_delete\_matricula

BEFORE DELETE ON matricula

FOR EACH ROW

BEGIN

IF (

((SELECT COUNT(\*)

FROM matricula

WHERE ida = OLD.ida) <= 1)

OR

((SELECT COUNT(\*)

FROM matricula

WHERE idm = OLD.idm) <= 1)

)

THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'No se puede borrar el alumno';

END IF;

END$$

DELIMITER ;

Trigger de antes de la actualización:

DROP TRIGGER IF EXISTS tBefore\_update\_matricula;

DELIMITER $$

CREATE TRIGGER tBefore\_update\_matricula

BEFORE UPDATE ON matricula

FOR EACH ROW

BEGIN

IF (NEW.ida <> OLD.ida)

THEN

IF ((SELECT COUNT(\*)

FROM matricula

WHERE ida = OLD.ida) = 1)

THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'No se puede actualizar esta tabla 1';

END IF;

END IF;

IF (NEW.idm <> OLD.idm)

THEN

If ((SELECT COUNT(\*)

FROM matricula

WHERE idm = OLD.idm) = 1)

THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'No se puede actualizar esta tabla 2';

END IF;

END IF;

END$$

DELIMITER ;

Solución propuesta por las soluciones: Prohibir las actualizaciones:

DROP TRIGGER IF EXISTS tBefore\_update\_matricula;

DELIMITER $$

CREATE TRIGGER tBefore\_update\_matricula

BEFORE UPDATE ON matricula

FOR EACH ROW

BEGIN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE\_TEXT = ‘Operacion no permitida’;

END$$

DELIMITER ;