DAW/DAM. UD 7. USUARIOS Y EXTENSIONES. ACTIVIDADES NO EVALUABLES PARTE 2 (SOLUCIONADO)

DAW/DAM. Bases de datos (BD)

UD 7. USUARIOS Y EXTENSIONES

Parte 2. Extensiones en MySQL

Prácticas no evaluables (solucionado)

Abelardo Martínez y **Pau Miñana**
Basado y modificado de Sergio Badal (www.sergiobadal.com) y Raquel Torres.
Curso 2023-2024

Aspectos a tener en cuenta

Importante

Estas actividades son opcionales y no evaluables pero es recomendable hacerlas para un mejor aprendizaje de la asignatura.

> **Si buscas las soluciones por Internet o preguntas al oráculo de ChatGPT, te estarás engañando a ti mismo**. Ten en cuenta que **ChatGPT no es infalible ni todopoderoso**.
>
> Es una gran herramienta para agilizar el trabajo una vez se domina una materia, pero usarlo como atajo en el momento de adquirir habilidades y conocimientos básicos perjudica gravemente tu aprendizaje. Si lo utilizas para obtener soluciones o asesoramiento respecto a las tuyas, revisa cuidadosamente las soluciones propuestas igualmente. Intenta resolver las actividades utilizando los recursos que hemos visto y la documentación extendida que encontrarás en el “Aula Virtual”.

Recomendaciones

Importante

- **No uses NUNCA tildes, ni eñes, ni espacios, ni caracteres no alfanuméricos** (salvo el guión bajo) **en los metadatos** (nombres de elementos de una base de datos).
- Sé coherente con el uso de mayúsculas/minúsculas.
- Usa **variables locales** siempre que puedas.
- Usa los delimitadores más habituales.

1. Procedimientos y funciones

Actividad no evaluable

Crea un ÚNICO *script* para cada uno de los ejercicios que te proponemos. **Recomendaciones**:

- Usa las mayúsculas/minúsculas y tabulaciones para hacerlo lo más legible posible.
- Incluye comentarios con -- cuando creas que son necesarios.
- Incluye siempre un **DROP XXX IF EXISTS** antes de la definición de un elemento, ya que estás en un ámbito académico.
- Utiliza los delimitadores y los tipos de variables más adecuados para cada contexto (variables locales dentro y variables de usuario fuera) si no te indica nada el enunciado.
- En este documento, usaremos los prefijos "p", "f" para los procedimientos y las funciones, respectivamente.

Base de datos

Vamos a utilizar la base de datos siguiente para los ejercicios de procedimientos y funciones:

```
DROP DATABASE IF EXISTS bd_productos;
CREATE DATABASE bd_productos;
USE bd_productos;

CREATE TABLE productos (
prodID  INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
nombre  VARCHAR(20) NOT NULL,
estado  VARCHAR(20) NOT NULL DEFAULT 'disponible',
coste   DECIMAL(10,2) NOT NULL DEFAULT 0.0,
precio  DECIMAL(10,2) NOT NULL DEFAULT 0.0,
CONSTRAINT pro_est_ck CHECK (estado IN ('disponible', 'agotado', 'en oferta',
);
```

```
CREATE TABLE gaming (
gameID          INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
nombre          VARCHAR(50) NOT NULL,
estado          VARCHAR(20) NOT NULL DEFAULT 'disponible',
coste           DECIMAL(10,2) NOT NULL DEFAULT 0.0,
precio          DECIMAL(10,2) NOT NULL DEFAULT 0.0,
precio_copia    DECIMAL(10,2),
CONSTRAINT gam_est_ck CHECK (estado IN ('disponible', 'agotado', 'en oferta',
);

INSERT INTO productos (nombre, estado, coste, precio) VALUES
('Manzanas','disponible', 4, 8),
('Peras', 'disponible', 1, 1.5),
('Melocotones', 'agotado', 5, 8),
('Kiwis', 'agotado', 2, 6),
('Moras','disponible', 12, 23),
('Fresas','disponible', 11, 20),
('Moras','agotado', 10, 20);

INSERT INTO gaming (nombre, estado, coste, precio) VALUES
('Play Station 5','disponible', 300,  423),
('Nintendo Switch Amoled', 'disponible', 200, 523),
('Wii', 'agotado', 225, 325),
('Xbox One', 'disponible', 100, 170),
('Xakarta','disponible', 112, 153),
('Retro GP430','agotado', 100, 299);
```

1.1. Función. Ranking por precio

Crea una función (**fRanking_por_precio**) que cuente el número de productos que hay más caros que el proporcionado. Crea una vista (**vi_ranking**) que use esa función para devolver este resultado de manera ordenada por ese mismo ranking. Por ejemplo:

```
mysql> SELECT * FROM vi_ranking;
+--------+-------------+--------+---------------------+
| prodID | nombre      | precio | posicion_por_precio |
+--------+-------------+--------+---------------------+
|      5 | Moras       |  23.00 |                   1 |
|      6 | Fresas      |  20.00 |                   2 |
|      7 | Moras       |  20.00 |                   2 |
|      1 | Manzanas    |   8.00 |                   4 |
|      3 | Melocotones |   8.00 |                   4 |
|      4 | Kiwis       |   6.00 |                   6 |
|      2 | Peras       |   1.50 |                   7 |
+--------+-------------+--------+---------------------+
7 rows in set (0,01 sec)
```

Solución

```
-- Función
DROP FUNCTION IF EXISTS fRanking_por_precio;
DELIMITER ||
```

```
CREATE FUNCTION fRanking_por_precio(
    piProdID INTEGER)
RETURNS INTEGER
NOT DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE iNumProductosMasCaros INTEGER;

    SELECT COUNT(*) INTO iNumProductosMasCaros
    FROM productos
    WHERE precio > (SELECT precio
                    FROM productos
                    WHERE prodID = piProdID);
    RETURN iNumProductosMasCaros+1;
END||
DELIMITER ;

-- Vista
DROP VIEW IF EXISTS vi_ranking;
CREATE VIEW vi_ranking AS
SELECT prodID, nombre, precio, fRanking_por_precio(prodID) AS posicion_por_pre
FROM productos
ORDER BY 4;
```

1.2. Procedimiento. Ranking por precio

Sin usar ni la función ni la vista del ejercicio anterior, crea un procedimiento (**pRanking_por_precio**) que reciba como parámetro el nombre de un producto e imprima el ranking en la lista de más vendidos. Si no existe ningún producto con ese nombre o existe más de uno, el procedimiento debe mostrar sendos mensajes de error como vemos en esta traza:

```
mysql> CALL pRanking_por_precio('Peras');
+---------------------+
| posicion_por_precio |
+---------------------+
|                   7 |
+---------------------+
1 row in set (0,00 sec)

Query OK, 0 rows affected (0,00 sec)

mysql> CALL pRanking_por_precio('Plátanos');
ERROR 1644 (45000): No se encuentra ningún producto con ese nombre
mysql> CALL pRanking_por_precio('Moras');
ERROR 1644 (45000): Hay más de un producto con ese nombre
```

Solución

```
DROP PROCEDURE IF EXISTS pRanking_por_precio;
```

```
DELIMITER ||
CREATE PROCEDURE pRanking_por_precio(
    IN pstNombreprod VARCHAR(20))
BEGIN
    DECLARE iNumProductosMasCaros INTEGER;
    DECLARE iCantidad INTEGER;

    SELECT COUNT(*) INTO iCantidad
    FROM productos
    WHERE nombre = pstNombreprod;
    CASE
        WHEN iCantidad=1 THEN
            SELECT COUNT(*) INTO iNumProductosMasCaros
            FROM productos
            WHERE precio> (SELECT precio
                           FROM productos
                           WHERE nombre = pstNombreprod);
            SELECT iNumProductosMasCaros+1 AS posicion_por_precio;
        WHEN iCantidad = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se encuentra ningún
        WHEN iCantidad > 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Hay más de un producto
    END CASE;
END||
DELIMITER ;
```

1.3. Procedimiento. Black Friday

Crea un procedimiento llamado **pBlackFriday** que reciba como parámetro la palabra “ON” o la palabra “OFF” y actúe de esta manera:

ON. Para todas las consolas marcadas como “disponible” (tabla gaming):

- Hace una copia del campo precio al campo precio_copia
- Aumenta en un 15% el precio y las marca como estado = “en oferta”
- Imprime un listado de las consolas afectadas “en oferta”

OFF. Para todas las consolas marcadas como “en oferta” (tabla gaming):

- Hace una copia del campo precio_copia al campo precio
- Las marca como estado = “disponible”
- Imprime un listado de las todas consolas “disponibles”

Una vez lo tengas, ejecuta el procedimiento con ON y con OFF para ver los resultados. Por ejemplo:

```
mysql> CALL pBlackFriday('ON');
+--------+-----------------------+-----------+--------+--------+------------
| gameID | nombre                | estado    | coste  | precio | precio_copia
+--------+-----------------------+-----------+--------+--------+------------
|      1 | Play Station 5        | en oferta | 300.00 | 486.45 |       423.00
|      2 | Nintendo Switch Amoled | en oferta | 200.00 | 601.45 |       523.00
|      4 | Xbox One              | en oferta | 100.00 | 195.50 |       170.00
|      5 | Xakarta               | en oferta | 112.00 | 175.95 |       153.00
+--------+-----------------------+-----------+--------+--------+------------
4 rows in set (0,01 sec)

Query OK, 0 rows affected (0,01 sec)

mysql> CALL pBlackFriday('OFF');
+--------+-----------------------+-----------+--------+--------+------------
```

```
| gameID | nombre                 | estado     | coste  | precio | precio_copi
+--------+------------------------+------------+--------+--------+-----------
|      1 | Play Station 5         | disponible | 300.00 | 423.00 |         NUL
|      2 | Nintendo Switch Amoled | disponible | 200.00 | 523.00 |         NUL
|      4 | Xbox One               | disponible | 100.00 | 170.00 |         NUL
|      5 | Xakarta                | disponible | 112.00 | 153.00 |         NUL
+--------+------------------------+------------+--------+--------+-----------
4 rows in set (0,00 sec)

Query OK, 0 rows affected (0,00 sec)
```

Solución

```
DROP PROCEDURE IF EXISTS pBlackFriday;
DELIMITER ||
CREATE PROCEDURE pBlackFriday(
    IN pstAccion VARCHAR(3))
BEGIN
    CASE pstAccion
        WHEN 'ON' THEN
            UPDATE gaming
            SET precio_copia=precio, precio=precio+(precio*0.15), estado='en o
            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 ;
```

1.4. Función. Otros Rankings

Crea dos funciones prácticamente idénticas a la función del ejercicio 1.1. llamadas **fRanking_por_coste** y **fRanking_por_beneficio** que devuelvan, respectivamente, la posición de ese producto en un listado ordenado por coste y por beneficio, siendo el beneficio la diferencia entre precio y coste.

Una vez tengas las tres funciones, crea una vista llamada **vi_ranking_completo** que muestre, para cada producto, su precio con su ranking, su coste con su ranking y su beneficio con su ranking, usando la función CONCAT y ordenado por beneficio descendente para que quede así:

```
mysql> SELECT * FROM vi_ranking_completo;
+--------+-------------+--------------------+--------------------+-----------
| prodID | nombre      | precio             | coste              | beneficio
+--------+-------------+--------------------+--------------------+-----------
|      5 | Moras       | 23.00 (ranking: 1) | 23.00 (ranking: 1) | 11.00 (rank
|      7 | Moras       | 20.00 (ranking: 2) | 20.00 (ranking: 3) | 10.00 (rank
|      6 | Fresas      | 20.00 (ranking: 2) | 20.00 (ranking: 2) | 9.00 (ranki
|      1 | Manzanas    | 8.00 (ranking: 4)  | 8.00 (ranking: 5)  | 4.00 (ranki
|      4 | Kiwis       | 6.00 (ranking: 6)  | 6.00 (ranking: 6)  | 4.00 (ranki
|      3 | Melocotones | 8.00 (ranking: 4)  | 8.00 (ranking: 4)  | 3.00 (ranki
|      2 | Peras       | 1.50 (ranking: 7)  | 1.50 (ranking: 7)  | 0.50 (ranki
+--------+-------------+--------------------+--------------------+-----------
7 rows in set (0,01 sec)
```

Solución

```
-- Funciones
DROP FUNCTION IF EXISTS fRanking_por_beneficio;
DELIMITER ||
CREATE FUNCTION fRanking_por_beneficio(
    piProdID INTEGER)
RETURNS INTEGER
NOT DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE iNumProductosMasRentables INTEGER;
    SELECT COUNT(*) INTO iNumProductosMasRentables
    FROM productos P1
    WHERE (P1.precio-P1.coste) > (SELECT (P2.precio-P2.coste)
                                  FROM productos P2
                                  WHERE P2.prodID = piProdID);
    RETURN iNumProductosMasRentables+1;
END||
DELIMITER ;

DROP FUNCTION IF EXISTS fRanking_por_coste;
DELIMITER ||
CREATE FUNCTION fRanking_por_coste(
    piProdID INTEGER)
RETURNS INTEGER
NOT DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE iNumProductosMasCostosos INTEGER;
    SELECT COUNT(*) INTO iNumProductosMasCostosos
    FROM productos P1
    WHERE P1.coste > (SELECT P2.coste
                      FROM productos P2
                      WHERE P2.prodID = piProdID);
    RETURN iNumProductosMasCostosos+1;
END||
```

```
DELIMITER ;

-- Vista
DROP VIEW IF EXISTS vi_ranking_completo;
CREATE VIEW vi_ranking_completo AS
SELECT prodID, nombre,
CONCAT(precio, ' (ranking: ', fRanking_por_precio(prodID), ')') AS precio,
CONCAT(precio, ' (ranking: ', fRanking_por_coste(prodID), ')') AS coste,
CONCAT(precio-coste, ' (ranking: ', fRanking_por_beneficio(prodID), ')') AS be
FROM productos
ORDER BY precio-coste DESC;
```

1.5. Procedimiento. Informe

Usando las funciones **fRanking_por_precio**, **fRanking_por_coste** y **fRanking_por_beneficio** crea un procedimiento llamado **pInforme** que imprima el nombre y el prodID de los primeros productos por precio (mayor), coste (menor) y beneficio (mayor). Intenta conseguir algo similar a esto:

```
mysql> CALL pInforme();
+--------+----------------------------+--------+
| prodID | Producto de precio máximo  | precio |
+--------+----------------------------+--------+
|      5 | Moras                      |  23.00 |
+--------+----------------------------+--------+
1 row in set (0,00 sec)

+--------+---------------------------+-------+
| prodID | Producto de coste mínimo  | coste |
+--------+---------------------------+-------+
|      2 | Peras                     |  1.00 |
+--------+---------------------------+-------+
1 row in set (0,00 sec)

+--------+--------------------------------+-----------+
| prodID | Producto de beneficio máximo  | beneficio |
+--------+--------------------------------+-----------+
|      5 | Moras                          |     11.00 |
+--------+--------------------------------+-----------+
1 row in set (0,01 sec)

Query OK, 0 rows affected (0,01 sec)
```

Solución

```
-- Función
DROP FUNCTION IF EXISTS fRanking_por_coste_menor;
DELIMITER ||
CREATE FUNCTION fRanking_por_coste_menor(
    piProdID INTEGER)
RETURNS INTEGER
NOT DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE iNumProductosMenorCoste INTEGER;

    SELECT COUNT(*) INTO iNumProductosMenorCoste
    FROM productos P1
    WHERE P1.coste < (SELECT P2.precio
                      FROM productos P2
                      WHERE P2.prodID = piProdID);
    RETURN iNumProductosMenorCoste;
END||
DELIMITER ;

-- Procedimiento
DROP PROCEDURE IF EXISTS pInforme;
DELIMITER ##
CREATE PROCEDURE pInforme()
BEGIN
    SELECT prodID, nombre AS 'Producto de precio máximo', precio
    FROM productos
    WHERE fRanking_por_precio(prodID) = 1;
    SELECT prodID, nombre AS 'Producto de coste mínimo', coste
```

```
    FROM productos
    WHERE fRanking_por_coste_menor(prodID) = 1;
    SELECT prodID, nombre AS 'Producto de beneficio máximo', precio-coste AS b
    FROM productos
    WHERE fRanking_por_beneficio(prodID) = 1;
END##
DELIMITER ;
```

1.6. Función. Día de la semana

Escribe una función **fDiaSemana** que reciba como parámetro de entrada un valor numérico que represente un día de la semana y que devuelva una cadena de caracteres con el nombre del día de la semana correspondiente. Por ejemplo, para el valor de entrada 1 debería devolver la cadena lunes:

```
mysql> SELECT fDiaSemana(1) AS 'Día';
+-------+
| Día   |
+-------+
| Lunes |
+-------+
1 row in set (0,00 sec)
```

Solución

```
DROP FUNCTION IF EXISTS fDiaSemana;
DELIMITER $$
CREATE FUNCTION fDiaSemana (
    piDia INTEGER)
RETURNS VARCHAR(15)
DETERMINISTIC
NO SQL
BEGIN
    DECLARE stDiaTexto VARCHAR(15);
```

```
    IF ((piDia < 0) OR (piDia > 7))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El día de la semana debe e
    END IF;
    CASE
        WHEN piDia = 1 THEN SET stDiaTexto = 'Lunes';
        WHEN piDia = 2 THEN SET stDiaTexto = 'Martes';
        WHEN piDia = 3 THEN SET stDiaTexto = 'Miércoles';
        WHEN piDia = 4 THEN SET stDiaTexto = 'Jueves';
        WHEN piDia = 5 THEN SET stDiaTexto = 'Viernes';
        WHEN piDia = 6 THEN SET stDiaTexto = 'Sábado';
        WHEN piDia = 7 THEN SET stDiaTexto = 'Domingo';
        ELSE SET stDiaTexto = '¡Día incorrecto!';
    END CASE;
    RETURN stDiaTexto;
END$$
DELIMITER ;
```

1.7. Función. Quitar tildes

Escribe una función **fCadena_sin_tildes** que reciba una cadena de entrada y devuelva la misma cadena pero sin tildes. La función tendrá que reemplazar todas las vocales que tengan acento por la misma vocal pero sin acento. Por ejemplo, si la función recibe como parámetro de entrada la cadena María la función debe devolver la cadena Maria:

```
mysql> SELECT fCadena_sin_tildes('María') AS 'Texto sin tildes';
+------------------+
| Texto sin tildes |
+------------------+
| Maria            |
+------------------+
1 row in set (0,00 sec)
```

Solución

```
DROP FUNCTION IF EXISTS fCadena_sin_tildes;
DELIMITER ||
CREATE FUNCTION fCadena_sin_tildes (
    pstCadena VARCHAR(10000))

RETURNS VARCHAR(10000)
DETERMINISTIC
NO SQL
```

```
BEGIN
    DECLARE stCadenaFinal VARCHAR(10000);

    SET stCadenaFinal = pstCadena;
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'Á', 'A');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'á', 'a');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'É', 'E');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'é', 'e');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'Í', 'I');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'í', 'i');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'Ó', 'O');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'ó', 'o');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'Ú', 'U');
    SET stCadenaFinal = REPLACE(stCadenaFinal, 'ú', 'u');

    RETURN stCadenaFinal;
END ||
DELIMITER ;
```

2. BD Centro educativo

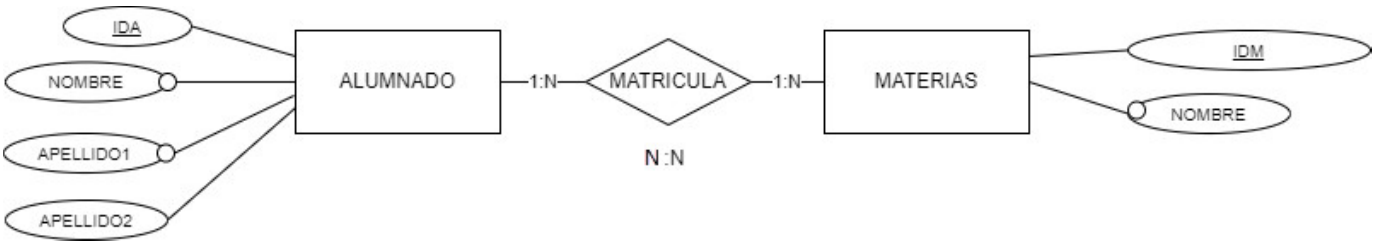
Actividad no evaluable

Crea un ÚNICO *script* para cada uno de los ejercicios que te proponemos. **Recomendaciones**:

- Usa las mayúsculas/minúsculas y tabulaciones para hacerlo lo más legible posible.
- Incluye comentarios con -- cuando creas que son necesarios.
- Incluye siempre un **DROP TRIGGER IF EXISTS** antes de la definición de un elemento, ya que estás en un ámbito académico. En entornos reales es más común usar, en su lugar, CREATE TRIGGER IF EXISTS.
- Utiliza los delimitadores y los tipos de variables más adecuados para cada contexto (variables locales dentro y variables de usuario fuera) si no te indica nada el enunciado.
- En este documento, usaremos el prefijo "t" para los triggers.

Vamos a utilizar la base de datos siguiente para los ejercicios de triggers:

Diagrama E-R



Modelo físico

```
DROP DATABASE IF EXISTS bd_alumnado;
CREATE DATABASE bd_alumnado;
USE bd_alumnado;

CREATE TABLE alumnado (
```

```
ida         INTEGER PRIMARY KEY,
nombre      VARCHAR(50) NOT NULL,
apellido1   VARCHAR(50) NOT NULL,
apellido2   VARCHAR(50),
email       VARCHAR(150),
nota        DECIMAL(10,2)
);

CREATE TABLE materias (
idm     INTEGER PRIMARY KEY,
nombre  VARCHAR(50) NOT NULL
);

CREATE TABLE matricula (
ida INTEGER,
idm INTEGER,
PRIMARY KEY (ida, idm),
FOREIGN KEY (ida) REFERENCES alumnado(ida),
FOREIGN KEY (idm) REFERENCES materias(idm)
);
```

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

Crea un nuevo trigger **tCheck_nota_before_insert**.

- Se ejecuta sobre la tabla alumnado antes de una operación de inserción.
- Si el nuevo valor de la nota que se quiere insertar es negativo, se guarda como 0.
- Si el nuevo valor de la nota que se quiere insertar es mayor que 10, se guarda como 10.

Crea un nuevo trigger **tCheck_nota_before_update**.

- Se ejecuta sobre la tabla alumnado antes de una operación de actualización.
- Si el nuevo valor de la nota que se quiere actualizar es negativo, se guarda como 0.
- Si el nuevo valor de la nota que se quiere actualizar es mayor que 10, se guarda como 10.

Verifica que los triggers se están ejecutando correctamente con estas sentencias:

> **DELETE FROM** alumnado;
>
> **INSERT INTO** alumnado (ida, nombre, apellido1, nota) **VALUES** (123, 'Sergio', 'Badal', -2);
>
> **INSERT INTO** alumnado (ida, nombre, apellido1, nota) **VALUES** (124, 'Paula', 'Murillo', 22);
>
> **SELECT** * **FROM** alumnado;

Y la salida será:

```
mysql> DELETE FROM alumnado;
Query OK, 0 rows affected (0,00 sec)

mysql> INSERT INTO alumnado (ida, nombre, apellido1, nota) VALUES (123, 'Sergi
Query OK, 1 row affected (0,00 sec)

mysql> INSERT INTO alumnado  (ida, nombre, apellido1, nota) VALUES (124, 'Paul
```

```
Query OK, 1 row affected (0,00 sec)

mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+-------+-------+
| ida | nombre | apellido1 | apellido2 | email | nota  |
+-----+--------+-----------+-----------+-------+-------+
| 123 | Sergio | Badal     | NULL      | NULL  |  0.00 |
| 124 | Paula  | Murillo   | NULL      | NULL  | 10.00 |
+-----+--------+-----------+-----------+-------+-------+
2 rows in set (0,00 sec)
```

Solución

```
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 ;


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 minúsculas

Escribe una función llamada **fFormato_email** que dada una cadena de texto devuelva la misma cadena sin espacios y en minúsculas. Las tildes son muy complejas de gestionar: Ignóralas. Prueba que funciona con esta llamada:

```
SELECT fFormato_email('cADEna de pruEBA');
```

Y la salida será:

```
mysql> SELECT fFormato_email('cADEna de pruEBA');
+-----------------------------------+
| fFormato_email('cADEna de pruEBA') |
+-----------------------------------+
| cadenadeprueba                    |
+-----------------------------------+
1 row in set (0,01 sec)
```

Solución

```
DROP FUNCTION IF EXISTS fFormato_email;
DELIMITER ||
CREATE FUNCTION fFormato_email(
    pstCadena VARCHAR(10000))

RETURNS VARCHAR(10000)
```

```
DETERMINISTIC
NO SQL
BEGIN
    DECLARE stCadenaFinal VARCHAR(10000);

    SET stCadenaFinal = LOWER(REPLACE(pstCadena, ' ', ''));
    RETURN stCadenaFinal;
END ||
DELIMITER ;
```

2.3. Procedimiento. Crear email

Escribe un procedimiento llamado **pCrear_email** que dados los parámetros nombre, apellido1, apellido2 y dominio, cree una dirección de correo electrónico y la devuelva como salida. Usa la función anterior. El correo electrónico estará formado por:

- Nombre completo (SIN ESPACIOS).
- Los tres primeros caracteres del parámetro apellido1 (SIN ESPACIOS).
- Los tres primeros caracteres del parámetro apellido2 (SIN ESPACIOS).
- El carácter @ y el dominio (“micorreo.com”) pasado como parámetro.
- Todo el email debe ser convertido a minúsculas antes de devolverlo.

Prueba que funciona con esta llamada:

```
CALL pCrear_email('Pepe', 'De la Osa', 'García', 'micorreo.com', @email);
```

Y la salida será:

```
mysql> CALL pCrear_email('Pepe', 'De la Osa', 'García', 'micorreo.com', @emai
Query OK, 0 rows affected (0,00 sec)

mysql> SELECT @email AS EMAIL;
+--------------------------+
| EMAIL                    |
+--------------------------+
| pepedelgar@micorreo.com |
+--------------------------+
1 row in set (0,00 sec)
```

Solución

```
DROP PROCEDURE IF EXISTS pCrear_email;
DELIMITER ||
CREATE PROCEDURE pCrear_email(
    IN pstNombre VARCHAR(50) ,
    IN pstApellido1 VARCHAR(50) ,
    IN pstApellido2 VARCHAR(50) ,
    IN pstDominio VARCHAR(50) ,
    OUT pstEmail VARCHAR(150)
)
BEGIN
    SET pstEmail = CONCAT(fFormato_email(pstNombre), LEFT(fFormato_email(pstAp
END||
DELIMITER ;
```

2.4. Trigger. Antes de insertar email

Crea un nuevo trigger **tCrear_email_before_insert**.

- Se ejecuta sobre la tabla "alumnado" antes de una operación de inserción.
- Si el nuevo valor del email que se quiere insertar es NULL, entonces se le creará automáticamente una dirección de email y se insertará en la tabla.
- Si el nuevo valor del email no es NULL se guardará en la tabla el valor del email.
- Nota: Para crear el nuevo email se deberá hacer uso del procedimiento **pCrear_email**.

Verifica que los triggers se están ejecutando correctamente con estas sentencias:

```
DELETE FROM alumnado;
INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota, email)
VALUES (111, 'Pedro', 'Lara' , 'Bielsa', -2, 'plara@micorreo.com');
INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota)
VALUES (112, 'Sara', 'Polendas', 'Zarra', 22);
SELECT * FROM alumnado;
```

Y la salida será:

```
mysql> DELETE FROM alumnado;
Query OK, 4 rows affected (0,01 sec)

mysql> INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota, email)
    -> VALUES (111, 'Pedro', 'Lara' , 'Bielsa', -2, 'plara@micorreo.com');
Query OK, 1 row affected (0,00 sec)

mysql> INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota)
    -> VALUES (112, 'Sara', 'Polendas', 'Zarra', 22);
Query OK, 1 row affected (0,00 sec)

mysql> SELECT * FROM alumnado;
```

```
+-----+--------+-----------+-----------+--------------------+-------+
| ida | nombre | apellido1 | apellido2 | email              | nota  |
+-----+--------+-----------+-----------+--------------------+-------+
| 111 | Pedro  | Lara      | Bielsa    | plara@micorreo.com |  0.00 |
| 112 | Sara   | Polendas  | Zarra     | NULL               | 10.00 |
+-----+--------+-----------+-----------+--------------------+-------+
2 rows in set (0,00 sec)
```

Solución

```
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, 'micorreo.
        SET NEW.email = @email;
    END IF;
END||
DELIMITER ;
```

2.5. Trigger. Después de actualizar email

Crea una nueva tabla llamada **log_cambios_email** que contenga los siguientes campos:

- id: clave primaria (entero autonumérico)
- ida: id del alumno (entero), con clave ajena a alumnado
- fecha_hora: marca de tiempo con el instante del cambio (fecha y hora)
  - fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP
- old_email: valor anterior del email (cadena de caracteres)
- new_email: nuevo valor con el que se ha actualizado

Crea un nuevo trigger **tGuardar_email_after_update**.

- Se ejecuta sobre la tabla alumnado después de una operación de actualización.
- Cada vez que un alumno/a modifique su dirección de email se deberá insertar un nuevo registro en la tabla log_cambios_email.

Verifica que los triggers se están ejecutando correctamente con estas sentencias:

```
DELETE FROM log_cambios_email;
DELETE FROM alumnado;
INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota, email)
VALUES (111, 'Pedro', 'Lara' , 'Bielsa', -2, 'plara@micorreo.com');
INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota)
VALUES (112, 'Sara', 'Polendas', 'Zarra', 22);

SELECT * FROM alumnado;
UPDATE alumnado SET email='pedro@micorreo.com' WHERE ida=111;
UPDATE alumnado SET nota=99 WHERE ida=112;

SELECT * FROM alumnado;
SELECT * FROM log_cambios_email;
```

Y la salida será:

```
mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+--------------------+-------+
| ida | nombre | apellido1 | apellido2 | email              | nota  |
+-----+--------+-----------+-----------+--------------------+-------+
| 111 | Pedro  | Lara      | Bielsa    | plara@micorreo.com |  0.00 |
| 112 | Sara   | Polendas  | Zarra     | NULL               | 10.00 |
+-----+--------+-----------+-----------+--------------------+-------+
2 rows in set (0,01 sec)

mysql>
mysql> UPDATE alumnado SET email='pedro@micorreo.com' WHERE ida=111;
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE alumnado SET nota=99 WHERE ida=112;
Query OK, 0 rows affected (0,00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql>
mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+--------------------+-------+
| ida | nombre | apellido1 | apellido2 | email              | nota  |
+-----+--------+-----------+-----------+--------------------+-------+
| 111 | Pedro  | Lara      | Bielsa    | pedro@micorreo.com |  0.00 |
| 112 | Sara   | Polendas  | Zarra     | NULL               | 10.00 |
+-----+--------+-----------+-----------+--------------------+-------+
2 rows in set (0,00 sec)

mysql> SELECT * FROM log_cambios_email;
+----+------+---------------------+--------------------+--------------------+
| id | ida  | fecha_hora          | old_email          | new_email          |
+----+------+---------------------+--------------------+--------------------+
|  1 |  111 | 2024-03-07 12:35:02 | plara@micorreo.com | pedro@micorreo.com |
+----+------+---------------------+--------------------+--------------------+
1 row in set (0,00 sec)
```

Solución

```
-- Nueva 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_pk FOREIGN KEY (ida) REFERENCES alumnado(ida)
);

-- Trigger
DROP TRIGGER IF EXISTS tGuardar_email_after_update;
DELIMITER ||
CREATE TRIGGER tGuardar_email_after_update
BEFORE 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

Crea un nuevo trigger **tGuardar_alumnado_after_delete**.

- Se ejecuta sobre la tabla alumnado después de una operación de borrado.
- Cada vez que se elimine un alumno/a de la tabla alumnado se deberá insertar un nuevo registro en una tabla llamada log_alumnado_eliminado.

La tabla **log_alumnado_eliminado** contiene los siguientes campos:

- id: clave primaria (entero autonumérico)
- ida: id del alumno (entero), **SIN CLAVE AJENA (¡la habremos borrado!)**
- fecha_hora: marca de tiempo con el instante del cambio (fecha y hora)
  - fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP
- nombre: nombre del alumno eliminado (cadena de caracteres)
- apellido1: primer apellido del alumno eliminado (cadena de caracteres)
- apellido2: segundo apellido del alumno eliminado (cadena de caracteres)
- email: email del alumno eliminado (cadena de caracteres)

Verifica que los triggers se están ejecutando correctamente con estas sentencias:

```
DELETE FROM log_cambios_email;
DELETE FROM alumnado;
DELETE FROM log_alumnado_eliminado;

INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota, email)
VALUES (111, 'Pedro', 'Lara' , 'Bielsa', -2, 'plara@micorreo.com');
INSERT INTO alumnado (ida, nombre, apellido1, apellido2, nota)
VALUES (112, 'Sara', 'Polendas', 'Zarra', 22);

SELECT * FROM alumnado;
DELETE FROM alumnado WHERE ida=111;
SELECT * FROM alumnado;
SELECT * FROM log_alumnado_eliminado;
```

Y la salida será:

```
mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+--------------------+-------+
| ida | nombre | apellido1 | apellido2 | email              | nota  |
+-----+--------+-----------+-----------+--------------------+-------+
| 111 | Pedro  | Lara      | Bielsa    | plara@micorreo.com |  0.00 |
| 112 | Sara   | Polendas  | Zarra     | NULL               | 10.00 |
+-----+--------+-----------+-----------+--------------------+-------+
2 rows in set (0,00 sec)

mysql> DELETE FROM alumnado WHERE ida=111;
Query OK, 1 row affected (0,01 sec)

mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+-------+-------+
| ida | nombre | apellido1 | apellido2 | email | nota  |
+-----+--------+-----------+-----------+-------+-------+
| 112 | Sara   | Polendas  | Zarra     | NULL  | 10.00 |
+-----+--------+-----------+-----------+-------+-------+
1 row in set (0,00 sec)

mysql> SELECT * FROM log_alumnado_eliminado;
+----+------+---------------------+--------+-----------+-----------+----------
| id | ida  | fecha_hora          | nombre | apellido1 | apellido2 | email
+----+------+---------------------+--------+-----------+-----------+----------
|  3 |  111 | 2024-03-07 13:06:06 | Pedro  | Lara      | Bielsa    | plara@mic
+----+------+---------------------+--------+-----------+-----------+----------
1 row in set (0,00 sec)
```

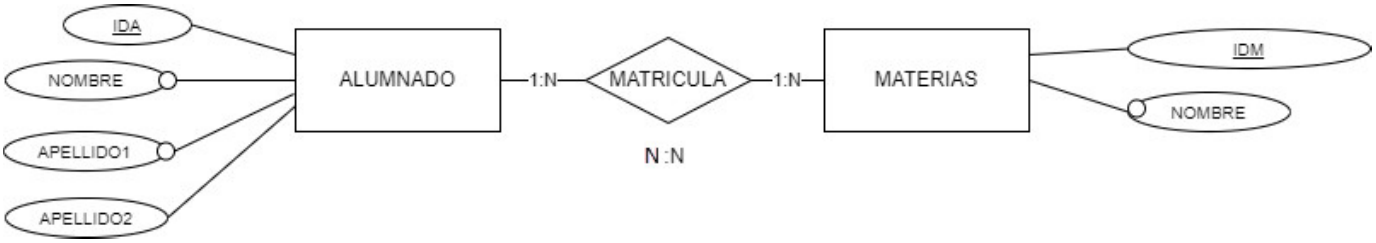
Solución

```
-- Nueva tabla log_alumnado_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),
apellido2   VARCHAR(50),
email       VARCHAR(150)
);

-- 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_eliminado (ida, nombre, apellido1, apellido2, ema
    VALUES (OLD.ida, OLD.nombre, OLD.apellido1, OLD.apellido2, OLD.email);
END||
DELIMITER ;
```

2.7. Trigger. Participaciones N:N

Recordamos el diagrama de base de datos:



Ejecuta el SCRIPT inicial de la BD de alumnado y estas instrucciones:

```
DROP TRIGGER IF EXISTS tBefore_delete_matricula;
DROP TRIGGER IF EXISTS tBefore_update_matricula;
DELETE FROM matricula;
DELETE FROM materias;
DELETE FROM alumnado;

INSERT INTO alumnado (ida, nombre, apellido1, nota, email) VALUES (111, 'Pedro', 'López', -2, 'plopez@lopez.com');
INSERT INTO alumnado (ida, nombre, apellido1, nota) VALUES (112, 'Sara', 'Gómez', 22);
INSERT INTO materias (idm, nombre) VALUES (211, 'Mates');
INSERT INTO materias (idm, nombre) VALUES (212, 'Lengua');
INSERT INTO matricula (ida, idm) VALUES (111, 211);
INSERT INTO matricula (ida, idm) VALUES (112, 212);

SELECT * FROM alumnado;
SELECT * FROM materias;
SELECT * FROM matricula;
```

Crea los triggers que consideres necesarios para implementar las restricciones de integridad necesarias para garantizar las participaciones N:N en la base de datos.

- IMPORTANTE: En N:N **Ignora las inserciones, prohíbe los updates y gestiona los borrados**.
- No se puede controlar que un nuevo alumno/a tenga una materia asociada y gestionar los updates de la tabla de cruces, puesto que es extremadamente complejo.

Verifica que los triggers se están ejecutando correctamente con estas sentencias:

```
UPDATE matricula SET ida=112 WHERE ida=111; -- (debería dar error)
UPDATE matricula SET idm=221 WHERE idm=211; -- (debería dar error)

DELETE FROM matricula WHERE ida=111; -- (debería dar error)
DELETE FROM matricula WHERE idm=211; -- (debería dar error)

INSERT INTO matricula (ida, idm) VALUES (112, 211); -- (debería permitirse)
INSERT INTO matricula (ida, idm) VALUES (111, 212); -- (debería permitirse)

DELETE FROM matricula WHERE ida=112 AND idm=212;-- (debería permitirse)
UPDATE matricula SET idm=212 WHERE ida=112 AND idm=211; -- (debería dar error)
```

Y la salida será:

```
mysql> UPDATE matricula SET ida=112 WHERE ida=111; -- (debería dar error)
ERROR 1644 (45000): Operación no permitida
*****
====> No se pueden modificar las matrículas.
*****

mysql> UPDATE matricula SET idm=221 WHERE idm=211; -- (debería dar error)
ERROR 1644 (45000): Operación no permitida
*****
====> No se pueden modificar las matrículas.
*****

mysql> DELETE FROM matricula WHERE ida=111; -- (debería dar error)
ERROR 1644 (45000): Operación no permitida
*****
====> No puede haber ALUMNADO sin MATERIAS y viceversa.
*****
```

```
mysql> DELETE FROM matricula WHERE idm=211; -- (debería dar error)
ERROR 1644 (45000): Operación no permitida
*****
====> No puede haber ALUMNADO sin MATERIAS y viceversa.
*****

mysql> INSERT INTO matricula (ida, idm) VALUES (112, 211); -- (debería permiti
Query OK, 1 row affected (0,01 sec)

mysql> INSERT INTO matricula (ida, idm) VALUES (111, 212); -- (debería permiti
Query OK, 1 row affected (0,00 sec)

mysql> DELETE FROM matricula WHERE ida=112 AND idm=212;-- (debería permitirse)
Query OK, 1 row affected (0,00 sec)

mysql> UPDATE matricula SET idm=212 WHERE ida=112 AND idm=211; -- (debería dar
ERROR 1644 (45000): Operación no permitida
*****
====> No se pueden modificar las matrículas.
*****
```

Solución

```
-- Triggers
DROP TRIGGER IF EXISTS tBefore_update_matricula;
DELIMITER ||
CREATE TRIGGER tBefore_update_matricula
-- tBefore_update_matricula:
-- Gestionar los updates de la tabla de cruces es extremadamente complejo.
BEFORE UPDATE ON matricula
FOR EACH ROW
```

```
BEGIN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Operación no permitida\n*****\
END||
DELIMITER ;

DROP TRIGGER IF EXISTS tBefore_delete_matricula;
DELIMITER ||
CREATE TRIGGER tBefore_delete_matricula
-- tBefore_delete_matricula:
-- Si se borra una MATRICULA comprobaremos que existe una MATERIA para cada AL
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 = 'Operación no permitida\n**
    END IF;
END||
DELIMITER ;
```

3. Bibliografía

- MySQL 8.0 Reference Manual. Defining Stored Programs. https://dev.mysql.com/doc/refman/8.0/en/stored-programs-defining.html
- Tutorial de Triggers SQL con ejemplos sencillos. https://www.srcodigofuente.es/aprender-sql/triggers-sql?utm_content=cmp-true
- MySQL 8.0 Reference Manual. SIGNAL Statement. https://dev.mysql.com/doc/refman/8.0/en/signal.html
- MySQL SIGNAL Statement. https://www.mysqltutorial.org/mysql-stored-procedure/mysql-signal/
- MySQL - SIGNAL Statement. https://www.tutorialspoint.com/mysql/mysql_signal_statement.htm
- MySQL Tutorial. https://www.w3schools.com/mysql/
- MySQL Tutorial. https://www.mysqltutorial.org/
- Oracle Database Documentation. https://docs.oracle.com/en/database/oracle/oracle-database/index.html