CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UNIDAD 7

USUARIOS Y EXTENSIONES (MYSQL)

BASES DE DATOS 22/23
CFGS DAW

BOLETÍN EJERCICIOS 7.2. EXTENSIONES EN MYSQL

Revisado y ampliado por:
Abelardo Martínez y Pau Miñana
Autor:
Sergio Badal

ÍNDICE DE CONTENIDO

UD7.2. BOLETÍN EXTENSIONES EN MYSQL

1. EJERCICIOS SOBRE PROCEDIMIENTOS Y FUNCIONES

Crea un ÚNICO script para cada uno de los ejercicios que te proponemos usando las **buenas prácticas** que has ido aprendiendo durante el curso, como son el uso estas:

1. Comienza siempre por el comando **USE nombre_de_la_base_de_datos;**
2. Usa las mayúsculas/minúsculas y las tabulaciones para hacerlo lo más legible posible.
3. Incluye comentarios con -- cuando creas que son necesarios.
4. Incluye siempre un **DROP XXX IF EXISTS** antes de la definición de un elemento ya que estás un en un ámbito académico.
5. Utiliza los delimitadores y los tipos de variables más adecuados para cada contexto si no te indica nada el enunciado.
6. En este documento, usaremos los prefijos p_, f_ para los procedimientos y las funciones respectivamente.
7. Ejecuta el siguiente script para poder crear los PROGRAMAS que te solicitamos:

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
DROP DATABASE IF EXISTS bd_productos;
CREATE DATABASE bd_productos;
USE bd_productos;
CREATE TABLE productos (
   id INT 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 );
CREATE TABLE gaming (
   id INT 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) );
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 EJERCICIO 1.1: FUNC. RANKING POR PRECIO

Usando el tipo de variables y de delimitadores que prefieras, crea una FUNCIÓN (**f_ranking_por_precio**) que cuente el número de productos que hay más caros que el proporcionado y crea una vista llamada **vi_ranking** que use esa función para devolver este resultado de manera ordenada por ese mismo ranking:

```
mysql> SELECT * FROM vi_ranking;
+----+-------------+--------+---------------------+
| id | nombre      | precio | posicion_por_precio |
+----+-------------+--------+---------------------+
|  5 | Moras       |     23 |                   1 |
|  6 | Fresas      |     20 |                   2 |
|  7 | Moras       |     20 |                   2 |
|  1 | Manzanas    |      8 |                   4 |
|  3 | Melocotones |      8 |                   4 |
|  4 | Kiwis       |      6 |                   6 |
|  2 | Peras       |    1.5 |                   7 |
+----+-------------+--------+---------------------+
7 rows in set (0.00 sec)
```

1.2 EJERCICIO 1.2: PROC. RANKING POR PRECIO

Sin usar ni la función ni la vista del ejercicio anterior y recurriendo al tipo de variables y de delimitadores que prefieras, crea un PROCEDIMIENTO (**p_ranking_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 p_ranking_por_precio('Peras');
+---------------------+
| posicion_por_precio |
+---------------------+
|                   7 |
+---------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

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

1.3 EJERCICIO 1.3: PROC. NOTA

Realiza un PROCEDIMIENTO llamado **p_nota** que reciba una nota numérica real (DECIMAL(10,2)) entre 0 y 10 y devuelva si es un suspenso, suficiente, bien, etc.

Para el cuerpo de programa usa variables de usuario y, para la llamada, prueba con valores y variables de usuario. Utiliza un CASE WHEN y ## como delimitador.

```
mysql> CALL p_nota(3);
+--------------+
| NOTA         |
+--------------+
| Insuficiente |
+--------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> SET @nota=9;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p_nota(@nota);
+---------------+
| NOTA          |
+---------------+
| Sobresaliente |
+---------------+
1 row in set (0.00 sec)
```

1.4 EJERCICIO 1.4: PROC. BLACK FRIDAY

Crea un PROCEDIMIENTO llamado **p_blackFriday** 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.

```
mysql> CALL p_blackFriday('ON');
+----+------------------------+-----------+-------+--------+--------------+
| id | nombre                 | estado    | coste | precio | precio_copia |
+----+------------------------+-----------+-------+--------+--------------+
|  1 | Play Station 5         | en oferta |   300 | 486.45 |          423 |
|  2 | Nintendo Switch Amoled | en oferta |   200 | 601.45 |          523 |
|  4 | Xbox One               | en oferta |   100 |  195.5 |          170 |
|  5 | Xakarta                | en oferta |   112 | 175.95 |          153 |
+----+------------------------+-----------+-------+--------+--------------+
4 rows in set (0.01 sec)

Query OK, 0 rows affected (0.06 sec)

mysql> CALL p_blackFriday('OFF');
+----+------------------------+------------+-------+--------+--------------+
| id | nombre                 | estado     | coste | precio | precio_copia |
+----+------------------------+------------+-------+--------+--------------+
|  1 | Play Station 5         | disponible |   300 |    423 |         NULL |
|  2 | Nintendo Switch Amoled | disponible |   200 |    523 |         NULL |
|  4 | Xbox One               | disponible |   100 |    170 |         NULL |
|  5 | Xakarta                | disponible |   112 |    153 |         NULL |
+----+------------------------+------------+-------+--------+--------------+
4 rows in set (0.01 sec)

Query OK, 0 rows affected (0.04 sec)
```

1.5 EJERCICIO 1.5: FUNC. OTROS RANKINGS

Crea dos funciones prácticamente idénticas a la función del ejercicio 1.1. llamadas **f_ranking_por_coste** y **f_ranking_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;
+----+-------------+------------------+------------------+------------------+
| id | nombre      | precio           | coste            | beneficio        |
+----+-------------+------------------+------------------+------------------+
|  5 | Moras       | 23 (ranking: 1)  | 23 (ranking: 1)  | 11 (ranking: 1)  |
|  7 | Moras       | 20 (ranking: 2)  | 20 (ranking: 3)  | 10 (ranking: 2)  |
|  6 | Fresas      | 20 (ranking: 2)  | 20 (ranking: 2)  | 9 (ranking: 3)   |
|  1 | Manzanas    | 8 (ranking: 4)   | 8 (ranking: 5)   | 4 (ranking: 4)   |
|  4 | Kiwis       | 6 (ranking: 6)   | 6 (ranking: 6)   | 4 (ranking: 4)   |
|  3 | Melocotones | 8 (ranking: 4)   | 8 (ranking: 4)   | 3 (ranking: 6)   |
|  2 | Peras       | 1.5 (ranking: 7) | 1.5 (ranking: 7) | 0.5 (ranking: 7) |
+----+-------------+------------------+------------------+------------------+
7 rows in set, 3 warnings (0.00 sec)
```

1.6 EJERCICIO 1.6: PROC. INFORME

Usando las funciones **f_ranking_por_precio, f_ranking_por_coste y f_ranking_por_beneficio** crea un PROCEDIMIENTO llamado **p_informe** que imprima el nombre y el id de los primeros productos por precio (mayor), coste (menor) y beneficio (mayor).

Intenta conseguir algo similar a esto:

```
ysql> CALL p_informe();
+----+---------------------------+--------+
| id | Producto de precio máximo | precio |
+----+---------------------------+--------+
|  5 | Moras                     |     23 |
+----+---------------------------+--------+
1 row in set (0.00 sec)

+----+--------------------------+-------+
| id | Producto de coste mínimo | coste |
+----+--------------------------+-------+
|  2 | Peras                    |     1 |
+----+--------------------------+-------+
1 row in set (0.00 sec)

+----+-------------------------------+-----------+
| id | Producto de beneficio máximo | beneficio |
+----+-------------------------------+-----------+
|  5 | Moras                         |        11 |
+----+-------------------------------+-----------+
1 row in set (0.01 sec)

uery OK, 0 rows affected (0.01 sec)
```

1.7 EJERCICIO 1.7: OTRAS FUNCIONES

a. Escribe una función que reciba un número entero de entrada y devuelva TRUE si el número es par o FALSE en caso contrario.

b. Escribe una función 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.

c. Escribe una función que reciba tres números reales como parámetros de entrada y devuelva el mayor de los tres.

d. Escribe una función 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.

2. EJERCICIOS SOBRE TRIGGERS

Crea un ÚNICO script para cada uno de los ejercicios que te proponemos usando las **buenas prácticas** que has ido aprendiendo durante el curso. Estas son algunas de ellas:

1. Comienza siempre por el comando **USE nombre_de_la_base_de_datos;**
2. Usa las mayúsculas/minúsculas y las tabulaciones para hacerlo lo más legible posible.
3. Incluye comentarios con -- cuando creas que son necesarios.
4. 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 (recuerda que esta opción no es válida en PROCEDURE y FUNCTION).**
5. Utiliza los delimitadores y los tipos de variables más adecuados para cada contexto, si no te indican lo contrario (DELIMITER ||, variables locales dentro y variables de usuario fuera).
6. En este documento, usaremos el prefijo “t_” para los triggers, pero es totalmente opcional.

Ejecuta el siguiente script para poder crear los PROGRAMAS (TRIGGERS) que te solicitamos:

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
DROP DATABASE IF EXISTS bd_alumnado;
CREATE DATABASE bd_alumnado;
USE bd_alumnado;
CREATE TABLE alumnado (
   ida INT 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 INT PRIMARY KEY,
   nombre VARCHAR(50) NOT NULL);
CREATE TABLE matricula (
   ida INT,
   idm INT,
  PRIMARY KEY (ida, idm),
  FOREIGN KEY (ida) REFERENCES alumnado(ida),
  FOREIGN KEY (idm) REFERENCES materias(idm)
);
```

2.1 EJERCICIO 2.1: Tabla alumnado

2.1.1 Triggers antes de insertar y antes de actualizar

- Crea un nuevo trigger: **t_check_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 : **t_check_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.

Ejecuta el SCRIPT inicial de la BD de alumnado y verifica que los triggers se están ejecutando correctamente con estas sentencias:

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
USE bd_alumnado;
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;
```

```
mysql> DELETE FROM alumnado;
Query OK, 2 rows affected (0,01 sec)

mysql> INSERT INTO alumnado (ida, nombre, apellido1, nota) VALUES (123, 'Sergio', 'Badal', -2);
Query OK, 1 row affected (0,00 sec)

mysql> INSERT INTO alumnado  (ida, nombre, apellido1, nota) VALUES (124, 'Paula', 'Murillo', 22);
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)
```

2.1.2 Función para eliminar espacios y pasar a minúsculas

- Escribe una función llamada **f_formato_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 f_formato_email('cADEna de pruEBA');**

```
mysql> SELECT f_formato_email('cADEna de pruEBA');
+-------------------------------------+
| f_formato_email('cADEna de pruEBA') |
+-------------------------------------+
| cadenadeprueba                      |
+-------------------------------------+
1 row in set (0.00 sec)
```

2.1.3 Procedimiento almacenado para crear email

- Escribe un procedimiento llamado **p_crear_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 p_crear_email('Juan', 'De Osma', 'Gracia', 'micorreo.com', @email);**

```
mysql> CALL p_crear_email('Juan', 'De Osma', 'Gracia', 'micorreo.com', @email);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @email AS EMAIL;
+-------------------------+
| EMAIL                   |
+-------------------------+
| juandeogra@micorreo.com |
+-------------------------+
1 row in set (0.00 sec)
```

2.1.4 Trigger antes de insertar

- Crea un nuevo trigger: **t_crear_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 la nueva email se deberá hacer uso del procedimiento **p_crear_email**.

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

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
USE bd_alumnado;
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;
```

```
mysql> DELETE FROM alumnado;
Query OK, 2 rows affected (0.00 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 |
| 112 | Sara   | Polendas  | Zarra     | sarapolzar@micorreo.com |   10 |
+-----+--------+-----------+-----------+-------------------------+------+
2 rows in set (0.00 sec)
```

Fíjate que los triggers se quedan almacenados en la base de datos, ejecutándose todos y cada uno de ellos cuando es necesario.

En este caso, hemos disparado los triggers que controlan las notas y los que controlan el correo electrónico.

2.1.5 Trigger después de actualizar

- 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: **t_guardar_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 una tabla llamada log_cambios_email.

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

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
USE bd_alumnado;
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;
```

(mira la salida en la página siguiente)

Fíjate que los triggers se quedan almacenados en la base de datos, ejecutándose todos y cada uno de ellos cuando es necesario.

En este caso, hemos disparado los triggers que controlan las notas y los que controlan el correo electrónico y los que registran los cambios de dicho correo.

```
mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+--------------------------+------+
| ida | nombre | apellido1 | apellido2 | email                    | nota |
+-----+--------+-----------+-----------+--------------------------+------+
| 111 | Pedro  | Lara      | Bielsa    | plara@micorreo.com       |    0 |
| 112 | Sara   | Polendas  | Zarra     | sarapolzar@micorreo.com  |   10 |
+-----+--------+-----------+-----------+--------------------------+------+
2 rows in set (0.00 sec)

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> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+--------------------------+------+
| ida | nombre | apellido1 | apellido2 | email                    | nota |
+-----+--------+-----------+-----------+--------------------------+------+
| 111 | Pedro  | Lara      | Bielsa    | pedro@micorreo.com       |    0 |
| 112 | Sara   | Polendas  | Zarra     | sarapolzar@micorreo.com  |   10 |
+-----+--------+-----------+-----------+--------------------------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM log_cambios_email;
+----+------+---------------------+-------------------+-------------------+
| id | ida  | fecha_hora          | old_email         | new_email         |
+----+------+---------------------+-------------------+-------------------+
|  2 |  111 | 2022-03-14 12:24:19 | plara@micorreo.com | pedro@micorreo.com |
+----+------+---------------------+-------------------+-------------------+
1 row in set (0.00 sec)
```

2.1.6 Trigger después de borrar

- Crea un nuevo trigger: **t_guardar_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:

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
USE bd_alumnado;
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;
```

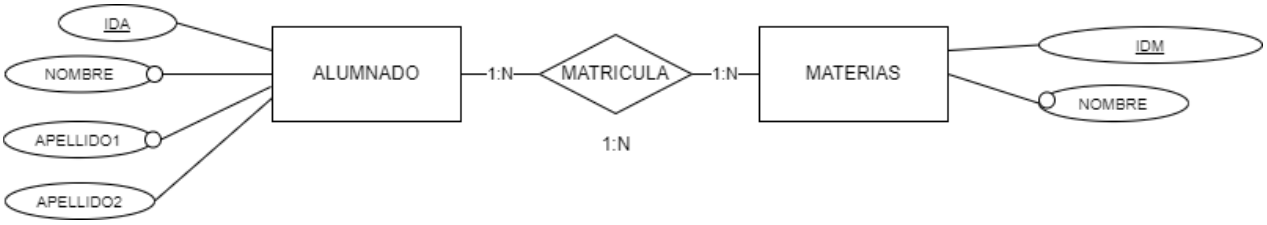
```
mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+-------------------------+------+
| ida | nombre | apellido1 | apellido2 | email                   | nota |
+-----+--------+-----------+-----------+-------------------------+------+
| 111 | Pedro  | Lara      | Bielsa    | plara@micorreo.com      |    0 |
| 112 | Sara   | Polendas  | Zarra     | sarapolzar@micorreo.com |   10 |
+-----+--------+-----------+-----------+-------------------------+------+
2 rows in set (0.00 sec)

mysql> DELETE FROM alumnado WHERE ida=111;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM alumnado;
+-----+--------+-----------+-----------+-------------------------+------+
| ida | nombre | apellido1 | apellido2 | email                   | nota |
+-----+--------+-----------+-----------+-------------------------+------+
| 112 | Sara   | Polendas  | Zarra     | sarapolzar@micorreo.com |   10 |
+-----+--------+-----------+-----------+-------------------------+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM log_alumnado_eliminado;
+----+------+--------+-----------+-----------+--------------------+-------
| id | ida  | nombre | apellido1 | apellido2 | email              | fecha_
+----+------+--------+-----------+-----------+--------------------+-------
|  5 |  111 | Pedro  | Lara      | Bielsa    | plara@micorreo.com | 2022-0
+----+------+--------+-----------+-----------+--------------------+-------
1 row in set (0.00 sec)
```

2.2 EJERCICIO 2.2: Participaciones 1:N

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

```
\! cls; -- LIMPIAR LA CONSOLA (Windows)
USE bd_alumnado;
DROP TRIGGER IF EXISTS t_before_delete_matriculas;
DROP TRIGGER IF EXISTS t_before_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 1:N en la siguiente 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)
```