```sql
/* BD.Evaluable3.1.DQL.DAM - JOSE MANUEL MORENO BOLIVAR - 52659570N */

/*
SCRIPT 1B: ACADEMIAS Y CURSOS POR COSTES ASOCIADOS (2 PUNTOS)

Crea una función para obtener, dado el código de academia y código de curso, el número de costes asociados y una segunda función para obtener, también dado un código de academia y código de curso, los euros totales que ha costado. Usando las dos funciones, lista los 2 cursos que más costes asociados tienen y otro listado para los 2 cursos que más dinero han costado.

Se pide:
• Crea la función fNumcostesAcadCurso
• Crea la función fTotalCostesAcadCurso
*/

DROP FUNCTION IF EXISTS fNumcostesAcadCurso;
DELIMITER $$
CREATE FUNCTION fNumcostesAcadCurso(codcurso VARCHAR(10), codacad VARCHAR(10))
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE nCostes INT;
    SET nCostes = (SELECT COUNT(*)
                   FROM coste c
                   WHERE codcurso = c.codcurso
                   AND codacad = c.codacad);
RETURN nCostes;
END$$
DELIMITER ;

DROP FUNCTION IF EXISTS fTotalCostesAcadCurso;
DELIMITER $$
CREATE FUNCTION fTotalCostesAcadCurso(codcurso VARCHAR(10), codacad VARCHAR(10))
RETURNS FLOAT
DETERMINISTIC
BEGIN
    DECLARE costesTotales INT;
    SET costesTotales = (SELECT SUM(c.importe)
                         FROM coste c
                         WHERE codcurso = c.codcurso
                         AND codacad = c.codacad
                         GROUP BY c.codcurso, c.codacad);
    RETURN costesTotales;
END$$
DELIMITER ;

/* Prueba de los SELECTS */
SELECT distinct codcurso, codacad, fNumcostesAcadCurso(codcurso, codacad) AS Coste_Total
FROM coste
ORDER BY Coste_Total DESC LIMIT 2;

SELECT distinct codcurso, codacad, fTotalCostesAcadCurso(codcurso, codacad) AS Coste_Total
FROM coste
ORDER BY Coste_Total DESC LIMIT 2;
```

```
/*
SCRIPT 2B: LISTAR ALUMNOS POR LETRA Y TIPO (2 PUNTOS)

Crea un procedimiento almacenado para obtener el nombre completo y la descripción del curso en que se encuentran matriculados los alumnos cuyo nombre comienza por la letra pasada como parámetro (de entrada) y son del tipo pasado como parámetro (de entrada) (P para Padawan y S para Senior) y devuelve el número de resultados en un tercer parámetro (de salida).
Ordena los resultados por el nombre completo.

Se pide:
• Crea el procedimiento pListarAlumnos_porLetrayTipo
• Muestra un mensaje de error si no se recibe una 'P' o una 'S' como 2° parámetro.
• Para el nombre completo usa la función CONCAT.
*/

DROP PROCEDURE IF EXISTS pListarAlumnos_porLetrayTipo;
DELIMITER $$
CREATE PROCEDURE pListarAlumnos_porLetrayTipo(
    IN letra CHAR,
    IN tipo CHAR,
    OUT resultados INT)
BEGIN
    CASE
        WHEN tipo = 'P' THEN
            SELECT COUNT(c.codcurso) INTO resultados
            FROM curso c, alumno a, padawan p
            WHERE a.nombre LIKE CONCAT(letra, '%')
            AND a.idalumno = p.idalumno
            AND c.codcurso = a.codcurso
            AND c.codacad = a.codacad;

            SELECT c.nombre, c.descripcion, concat(a.nombre, ' ', a.apellidos) AS alumnos
            FROM curso c, alumno a, padawan p
            WHERE a.nombre LIKE CONCAT(letra, '%')
            AND a.idalumno = p.idalumno
            AND c.codcurso = a.codcurso
            AND c.codacad = a.codacad
            ORDER BY c.nombre;

        WHEN tipo = 'S' THEN
            SELECT COUNT(c.codcurso) INTO resultados
            FROM curso c, alumno a, senior s
            WHERE a.nombre LIKE CONCAT(letra, '%')
            AND a.idalumno = s.idalumno
            AND c.codcurso = a.codcurso
            AND c.codacad = a.codacad;

            SELECT c.nombre, c.descripcion, CONCAT(a.nombre, ' ', a.apellidos) AS alumnos
            FROM curso c, alumno a, senior s
            WHERE a.nombre LIKE concat(letra, '%')
            AND a.idalumno = s.idalumno
            AND c.codcurso = a.codcurso
            AND c.codacad = a.codacad
            ORDER BY c.nombre;

        ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tipo erróneo';
    END CASE;
END$$
DELIMITER ;

/* Ejemplo para el CALL y @resultados */
CALL pListarAlumnos_porLetrayTipo('A','S',@resultados);
SELECT @resultados AS 'Numero de resultados';
```

```
/*
SCRIPT 3B: PÉRDIDA SEMÁNTICA EN ALUMNOS (T+D) (2 PUNTOS)

Al traducir del diseño conceptual al modelo relacional, existía una pérdida semántica en las especializaciones que no eran Parcial+Solapada (PS), quedándose como una restricción de integridad que dijimos que "ya resolveríamos más adelante".
Pues bien, mediante triggers podemos traducir cualquier especialización (TS, TD o PD) y superar así esa pérdida semántica, aunque no de manera sencilla. Solo las disjuntas tienen una solución con triggers más o menos sencilla.
Te proponemos crear los triggers necesarios para asegurar la especialización DISJUNTA de los ALUMNOS en PADAWAN y SENIOR durante las inserciones, ignorando las actualizaciones y los borrados por su complejidad. (Los datos del script DDL pueden no cumplir estas restricciones, pero el trigger servirá a partir de ahora)

Se pide:
• Crea el trigger necesario para prevenir inserciones incorrectas en PADAWAN
• Crea el trigger necesario para prevenir inserciones incorrectas en SENIOR
*/

DROP TRIGGER IF EXISTS insercionDisjuntaPadawan;
DELIMITER $$
CREATE TRIGGER insercionDisjuntaPadawan
BEFORE INSERT ON padawan
FOR EACH ROW
BEGIN
    IF (
        (SELECT COUNT(s.idalumno)
         FROM senior s
         WHERE NEW.idalumno = s.idalumno) > 0)
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Padawan que tenga una especialización Senior';
    ELSEIF (
        NEW.idalumno NOT IN (SELECT idalumno FROM alumno))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Padawan que no sea alumno';
    END IF;
END$$
DELIMITER ;

/* Prueba de insercion en la tabla PADAWAN */
INSERT INTO padawan VALUES('ALU006',current_date()); /** Alumno senior **/
INSERT INTO padawan VALUES('ALU016',current_date()); /** No Alumno **/

DROP TRIGGER IF EXISTS insercionDisjuntaSenior;
DELIMITER $$
CREATE TRIGGER insercionDisjuntaSenior
BEFORE INSERT ON senior
FOR EACH ROW
BEGIN
    IF (
        (SELECT COUNT(p.idalumno)FROM padawan p
         WHERE NEW.idalumno = p.idalumno) > 0)
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Senior que tenga una especialización Padawan';
    ELSEIF (
        NEW.idalumno NOT IN (SELECT idalumno FROM alumno))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Senior que no sea alumno';
    END IF;
END$$
DELIMITER ;

/* Prueba de insercion en la tabla SENIOR */
INSERT INTO senior VALUES('ALU009','sable láser',current_date()); /** Alumno padawan **/
INSERT INTO senior VALUES('ALU016','sable láser',current_date()); /** No Alumno **/
```

```
/*
SCRIPT 4B: TRIGGERS PARA LAS PARTICIPACIONES 1:N (2 PUNTOS)

Al traducir del diseño conceptual al modelo relacional, existía una pérdida semántica en las participaciones 1:N, quedándose como una restricción de integridad que dijimos que "ya resolveríamos más adelante".
Pues bien, mediante triggers podemos traducir cualquier participación 1:N y superar así esa pérdida semántica de manera sencilla.
Te proponemos crear los triggers necesarios para asegurar la participación 1:N de la relación entre CURSO y ALUMNO, ignorando las inserciones porque no afectan a la 1:N.

Se pide:
• Crea el trigger necesario para prevenir borrados de ALUMNO que rompan la participación 1:N en la relación de CURSO con ALUMNO.
• Crea el trigger necesario para prevenir actualizaciones del campo CODCURSO de la tabla ALUMNO que rompan la participación 1:N en la relación de CURSO con ALUMNO.
*/

DROP TRIGGER IF EXISTS borradoAlumno;
DELIMITER $$
CREATE TRIGGER borradoAlumno
BEFORE DELETE ON Alumno
FOR EACH ROW
BEGIN
    IF ((SELECT COUNT(idalumno)
        FROM Alumno
        WHERE OLD.codacad = codacad
        AND OLD.codcurso = codcurso) <= 1)
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede borrar el ultimo alumno que queda en el curso';
    END IF;
END$$
DELIMITER ;

/* Prueba de borrado del ultimo alumno asociado a un curso */
DELETE FROM Alumno WHERE idalumno = 'ALU003';

DROP TRIGGER IF EXISTS actualizacionAlumno;
DELIMITER $$
CREATE TRIGGER actualizacionAlumno
BEFORE UPDATE ON alumno
FOR EACH ROW
BEGIN
    IF
        (((SELECT COUNT(idalumno)
           FROM Alumno
           WHERE OLD.codacad = codacad
           AND OLD.codcurso = codcurso) <= 1)
        AND NEW.codcurso <> OLD.codcurso
        )
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede actualizar el campo CODCURSO del ultimo alumno que queda en el curso';
    END IF;
END$$
DELIMITER ;

/* Prueba de borrado del ultimo alumno asociado a un curso */
UPDATE Alumno SET codcurso = 8050 WHERE idalumno = 'ALU003';
```

```
/*
SCRIPT 5B: LÍNEAS DE COSTES CONSECUTIVAS (2 PUNTOS)

Crea los triggers necesarios para asegurar que los números de línea de un mismo coste
son consecutivos cuando se inserta una nueva línea de coste de un curso. Para hacerlo
más sencillo, ignora los borrados, gestiona las inserciones y prohíbe las
actualizaciones del campo NUMLINEA en esa tabla.

Se pide:
• Crea el trigger tAntesActualizarLineasCoste
    ◦   Muestra un mensaje de error cuando se intenta cambiar el campo LINEA
• Crea el trigger tAntesInsertarLineasCoste
    ◦   El campo NUMLINEA debe ser >0 y siempre consecutivo para un mismo COSTE, de
    manera que, si por ejemplo insertamos la línea 6 del coste del curso '6050' y
    academia 'GLEE', debe existir antes el 5. Si no existe esa línea 5, debe
    cancelarse la operación con un mensaje de error.
*/

DROP TRIGGER IF EXISTS tAntesActualizarLineasCoste;
DELIMITER $$
CREATE TRIGGER tAntesActualizarLineasCoste
BEFORE UPDATE ON coste
FOR EACH ROW
BEGIN
    IF
        NEW.linea <> OLD.linea
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede actualizar el campo
        LINEA de la tabla Coste';
    END IF;
END$$
DELIMITER ;

/* Prueba update linea */
UPDATE coste SET linea = 8 WHERE codacad = 'CRAIT' AND codcurso = 5050 AND linea = 3;

DROP TRIGGER IF EXISTS tAntesActualizarLineasCoste;
DELIMITER $$
CREATE TRIGGER tAntesActualizarLineasCoste
BEFORE INSERT ON coste
FOR EACH ROW
BEGIN
    IF (NEW.linea <> (  SELECT COUNT(*)
                        FROM coste
                        WHERE codcurso = NEW.codcurso
                        AND codacad = NEW.codacad)+1)
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El numero de linea debe ser
        consecutivo';
    END IF;
END$$
DELIMITER ;

/* Prueba de INPUT erroneo */
INSERT INTO coste VALUES(6, 5050, 'CRAIT', 'Alquiler de sables láser', 5.5);
```