

```

1  /* BD.Evaluable3.1.DQL.DAM - JOSE MANUEL MORENO BOLIVAR - 52659570N */
2
3  /*
4  SCRIPT 1B: ACADEMIAS Y CURSOS POR COSTES ASOCIADOS (2 PUNTOS)
5
6  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.
7
8  Se pide:
9  • Crea la función fNumcostesAcadCurso
10 • Crea la función fTotalCostesAcadCurso
11 */
12
13 DROP FUNCTION IF EXISTS fNumcostesAcadCurso;
14 DELIMITER $$
15 CREATE FUNCTION fNumcostesAcadCurso(codcurso VARCHAR(10), codacad VARCHAR(10))
16 RETURNS INT
17 DETERMINISTIC
18 BEGIN
19     DECLARE nCostes INT;
20     SET nCostes = (SELECT COUNT(*)
21                     FROM coste c
22                     WHERE codcurso = c.codcurso
23                     AND codacad = c.codacad);
24     RETURN nCostes;
25 END$$
26 DELIMITER ;
27
28 DROP FUNCTION IF EXISTS fTotalCostesAcadCurso;
29 DELIMITER $$
30 CREATE FUNCTION fTotalCostesAcadCurso(codcurso VARCHAR(10), codacad VARCHAR(10))
31 RETURNS FLOAT
32 DETERMINISTIC
33 BEGIN
34     DECLARE costesTotales INT;
35     SET costesTotales = (SELECT SUM(c.importe)
36                           FROM coste c
37                           WHERE codcurso = c.codcurso
38                           AND codacad = c.codacad
39                           GROUP BY c.codcurso, c.codacad);
40     RETURN costesTotales;
41 END$$
42 DELIMITER ;
43
44 /* Prueba de los SELECTS */
45 SELECT distinct codcurso, codacad, fNumcostesAcadCurso(codcurso, codacad) AS
  Coste_Total
46 FROM coste
47 ORDER BY Coste_Total DESC LIMIT 2;
48
49 SELECT distinct codcurso, codacad, fTotalCostesAcadCurso(codcurso, codacad) AS
  Coste_Total
50 FROM coste
51 ORDER BY Coste_Total DESC LIMIT 2;
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67

```

```

68  /*
69  SCRIPT 2B: LISTAR ALUMNOS POR LETRA Y TIPO (2 PUNTOS)
70
71  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.
72
73
74  Se pide:
75  • Crea el procedimiento pListarAlumnos_porLetrayTipo
76  • Muestra un mensaje de error si no se recibe una 'P' o una 'S' como 2º parámetro.
77  • Para el nombre completo usa la función CONCAT.
78  */
79
80  DROP PROCEDURE IF EXISTS pListarAlumnos_porLetrayTipo;
81  DELIMITER $$
82  CREATE PROCEDURE pListarAlumnos_porLetrayTipo(
83      IN letra CHAR,
84      IN tipo CHAR,
85      OUT resultados INT)
86  BEGIN
87      CASE
88          WHEN tipo = 'P' THEN
89              SELECT COUNT(c.codcurso) INTO resultados
90              FROM curso c, alumno a, padawan p
91              WHERE a.nombre LIKE CONCAT(letra, '%')
92              AND a.idalumno = p.idalumno
93              AND c.codcurso = a.codcurso
94              AND c.codacad = a.codacad;
95
96              SELECT c.nombre, c.descripcion, concat(a.nombre, ' ', a.apellidos) AS
    alumnos
97              FROM curso c, alumno a, padawan p
98              WHERE a.nombre LIKE CONCAT(letra, '%')
99              AND a.idalumno = p.idalumno
100             AND c.codcurso = a.codcurso
101             AND c.codacad = a.codacad
102             ORDER BY c.nombre;
103
104          WHEN tipo = 'S' THEN
105              SELECT COUNT(c.codcurso) INTO resultados
106              FROM curso c, alumno a, senior s
107              WHERE a.nombre LIKE CONCAT(letra, '%')
108              AND a.idalumno = s.idalumno
109              AND c.codcurso = a.codcurso
110              AND c.codacad = a.codacad;
111
112              SELECT c.nombre, c.descripcion, CONCAT(a.nombre, ' ', a.apellidos) AS
    alumnos
113              FROM curso c, alumno a, senior s
114              WHERE a.nombre LIKE concat(letra, '%')
115              AND a.idalumno = s.idalumno
116              AND c.codcurso = a.codcurso
117              AND c.codacad = a.codacad
118              ORDER BY c.nombre;
119
120          ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tipo erróneo';
121      END CASE;
122  END$$
123  DELIMITER ;
124
125  /* Ejemplo para el CALL y @resultados */
126  CALL pListarAlumnos_porLetrayTipo('A','S',@resultados);
127  SELECT @resultados AS 'Numero de resultados';
128
129
130
131
132
133
134

```

```

135  /*
136  SCRIPT 3B: PÉRDIDA SEMÁNTICA EN ALUMNOS (T+D) (2 PUNTOS)
137
138  Al traducir del diseño conceptual al modelo relacional, existía una pérdida semántica
139  en las especializaciones que no eran Parcial+Solapada (PS), quedándose como una
140  restricción de integridad que dijimos que "ya resolveríamos más adelante".
141  Pues bien, mediante triggers podemos traducir cualquier especialización (TS, TD o PD)
142  y superar así esa pérdida semántica, aunque no de manera sencilla. Solo las disjuntas
143  tienen una solución con triggers más o menos sencilla.
144  Te proponemos crear los triggers necesarios para asegurar la especialización DISJUNTA
145  de los ALUMNOS en PADAWAN y SENIOR durante las inserciones, ignorando las
146  actualizaciones y los borrados por su complejidad. (Los datos del script DDL pueden
147  no cumplir estas restricciones, pero el trigger servirá a partir de ahora)
148
149  Se pide:
150  • Crea el trigger necesario para prevenir inserciones incorrectas en PADAWAN
151  • Crea el trigger necesario para prevenir inserciones incorrectas en SENIOR
152  */
153
154  DROP TRIGGER IF EXISTS insercionDisjuntaPadawan;
155  DELIMITER $$
156  CREATE TRIGGER insercionDisjuntaPadawan
157  BEFORE INSERT ON padawan
158  FOR EACH ROW
159  BEGIN
160      IF (
161          (SELECT COUNT(s.idalumno)
162           FROM senior s
163           WHERE NEW.idalumno = s.idalumno) > 0)
164      THEN
165          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Padawan
166          que tenga una especialización Senior';
167      ELSEIF (
168          NEW.idalumno NOT IN (SELECT idalumno FROM alumno))
169      THEN
170          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Padawan
171          que no sea alumno';
172      END IF;
173  END$$
174  DELIMITER ;
175
176  /* Prueba de insercion en la tabla PADAWAN */
177  INSERT INTO padawan VALUES('ALU006',current_date()); /** Alumno senior **/
178  INSERT INTO padawan VALUES('ALU016',current_date()); /** No Alumno **/
179
180  DROP TRIGGER IF EXISTS insercionDisjuntaSenior;
181  DELIMITER $$
182  CREATE TRIGGER insercionDisjuntaSenior
183  BEFORE INSERT ON senior
184  FOR EACH ROW
185  BEGIN
186      IF (
187          (SELECT COUNT(p.idalumno)FROM padawan p
188           WHERE NEW.idalumno = p.idalumno) > 0)
189      THEN
190          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Senior
191          que tenga una especialización Padawan';
192      ELSEIF (
193          NEW.idalumno NOT IN (SELECT idalumno FROM alumno))
194      THEN
195          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede insertar un Senior
196          que no sea alumno';
197      END IF;
198  END$$
199  DELIMITER ;
200
201  /* Prueba de insercion en la tabla SENIOR */
202  INSERT INTO senior VALUES('ALU009','sable láser',current_date()); /** Alumno padawan
203  **/
204  INSERT INTO senior VALUES('ALU016','sable láser',current_date()); /** No Alumno **/

```

```

196  /*
197  SCRIPT 4B: TRIGGERS PARA LAS PARTICIPACIONES 1:N (2 PUNTOS)
198
199  Al traducir del diseño conceptual al modelo relacional, existía una pérdida semántica
200  en las participaciones 1:N, quedándose como una restricción de integridad que dijimos
201  que "ya resolveríamos más adelante".
202  Pues bien, mediante triggers podemos traducir cualquier participación 1:N y superar
203  así esa pérdida semántica de manera sencilla.
204  Te proponemos crear los triggers necesarios para asegurar la participación 1:N de la
205  relación entre CURSO y ALUMNO, ignorando las inserciones porque no afectan a la 1:N.
206
207  Se pide:
208  • Crea el trigger necesario para prevenir borrados de ALUMNO que rompan la
209  participación 1:N en la relación de CURSO con ALUMNO.
210  • Crea el trigger necesario para prevenir actualizaciones del campo CODCURSO de la
211  tabla ALUMNO que rompan la participación 1:N en la relación de CURSO con ALUMNO.
212  */
213
214  DROP TRIGGER IF EXISTS borradoAlumno;
215  DELIMITER $$
216  CREATE TRIGGER borradoAlumno
217  BEFORE DELETE ON Alumno
218  FOR EACH ROW
219  BEGIN
220      IF ((SELECT COUNT(idalumno)
221          FROM Alumno
222          WHERE OLD.codacad = codacad
223          AND OLD.codcurso = codcurso) <= 1)
224      THEN
225          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede borrar el ultimo
226          alumno que queda en el curso';
227      END IF;
228  END$$
229  DELIMITER ;
230
231  /* Prueba de borrado del ultimo alumno asociado a un curso */
232  DELETE FROM Alumno WHERE idalumno = 'ALU003';
233
234  DROP TRIGGER IF EXISTS actualizacionAlumno;
235  DELIMITER $$
236  CREATE TRIGGER actualizacionAlumno
237  BEFORE UPDATE ON alumno
238  FOR EACH ROW
239  BEGIN
240      IF
241          (((SELECT COUNT(idalumno)
242              FROM Alumno
243              WHERE OLD.codacad = codacad
244              AND OLD.codcurso = codcurso) <= 1)
245          AND NEW.codcurso <> OLD.codcurso
246          )
247      THEN
248          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede actualizar el campo
249          CODCURSO del ultimo alumno que queda en el curso';
250      END IF;
251  END$$
252  DELIMITER ;
253
254  /* Prueba de borrado del ultimo alumno asociado a un curso */
255  UPDATE Alumno SET codcurso = 8050 WHERE idalumno = 'ALU003';
256
257
258
259
260

```

```

261  /*
262  SCRIPT 5B: LÍNEAS DE COSTES CONSECUTIVAS (2 PUNTOS)
263
264  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.
265
266  Se pide:
267  • Crea el trigger tAntesActualizarLineasCoste
268    ◦ Muestra un mensaje de error cuando se intenta cambiar el campo LINEA
269  • Crea el trigger tAntesInsertarLineasCoste
270    ◦ 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.
271  */
272
273  DROP TRIGGER IF EXISTS tAntesActualizarLineasCoste;
274  DELIMITER $$
275  CREATE TRIGGER tAntesActualizarLineasCoste
276  BEFORE UPDATE ON coste
277  FOR EACH ROW
278  BEGIN
279      IF
280          NEW.linea <> OLD.linea
281      THEN
282          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede actualizar el campo
LINEA de la tabla Coste';
283      END IF;
284  END$$
285  DELIMITER ;
286
287  /* Prueba update linea */
288  UPDATE coste SET linea = 8 WHERE codacad = 'CRAIT' AND codcurso = 5050 AND linea = 3;
289
290  DROP TRIGGER IF EXISTS tAntesActualizarLineasCoste;
291  DELIMITER $$
292  CREATE TRIGGER tAntesActualizarLineasCoste
293  BEFORE INSERT ON coste
294  FOR EACH ROW
295  BEGIN
296      IF (NEW.linea <> (
297          SELECT COUNT(*)
298          FROM coste
299          WHERE codcurso = NEW.codcurso
300          AND codacad = NEW.codacad)+1)
301      THEN
302          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El numero de linea debe ser
consecutivo';
303      END IF;
304  END$$
305  DELIMITER ;
306
307  /* Prueba de INPUT erroneo */
INSERT INTO coste VALUES (6, 5050, 'CRAIT', 'Alquiler de sables láser', 5.5);

```