Ejercicios de SQL

Tienes una base de datos con las siguientes tablas y campos (a continuación un pseudo código de la creación de las tablas)

```
create table people {
  id integer primary key,
  name string,
  age integer,
  boss_id integer reference people(id),
-- foreign key a esta misma tabla personas (el jefe)
  favorite_author_id integer reference people(id)
-- foreign key a esta misma tabla (autor preferido)
}

create table books {
  id integer primary key,
  title string,
  owner_id integer reference people(id),
-- foreign key a la tabla personas (el dueño del libro)
  author_id integer reference people(id),
-- foreign key a la tabla personas (el autor del libro)
}
```

Ejercicio 1 (obligatorio)

Crea una consulta SQL que devuelva el listado de personas con la cantidad de libros de las que es dueño, este listado debe aparecer ordenado de mayor a menor. Las personas que no tengan ningún libro, no deben aparecer en el resultado. Si más de una persona tiene la misma cantidad de libros, entonces deben quedar ordenadas alfabéticamente entre ellas. La salida debe ser algo como:

```
name    | amount
--------+---------
paco    | 10
perico  | 8
pocholo | 8
pancho  | 5
pupo    | 5
pirolo  | 3
```

Ejercicio 2 (obligatorio)

Crea una consulta SQL que devuelva la edad del dueño de cada libro y la edad del autor. La salida debe ser algo como:

```
title       | owner_age | author_age
------------+-----------+-----------
papa goriot | 15        | 50
don quijote | 18        | 34
la iliada   | 25        | 70
```

Ejercicio 3 (obligatorio)

Crea una consulta SQL que devuelva el listado de personas **que han escrito más libros que la cantidad que poseen**. O sea cada persona puede haber escrito *N* libros y ser dueña de *M* libros. El resultado solo deben salir los nombres de las personas en las *N* > *M*.

```
name
----------
paco
perico
pirolo
```

Ejercicio 4 (opcional)

Se quiere actualizar la columna **`favorite_author_id`** de la tabla **`people`**, y almacenar en ella, el autor preferido de cada persona. Ese nuevo valor debe calcularse a partir de los autores de los libros de cada persona.

Por ejemplo, si una persona, es dueña de 10 libros, de los cuales 5 son del autor X, 3 son del autor Y, y 2 son del autor Z. debe aparecer el id del autor X en la columna **`favorite_author_id`** de dicha persona.

Se quiere realizar una sentencia SQL (UPDATE) para actualizar esa columna para todas las personas.

Ejercicios de PHP

Ejercicio 1 (obligatorio)

Se quiere implementar un método `sort_csv($csv, $column)` en PHP que ordene un *string* que representa un CSV que reciba como parámetro. El método además recibirá el número de la columna por la cual se quiere ordenar. La columna cero es la primera de todas. El método retorna un *string*. Por ejemplo, data la variable:

```
$data = <<<CSV
paco,2,3.6
perico,5,1.7
pirolo,1.3,2.6
pocholo,11,-1.5
CSV;
```

Nota: lo anterior es completamente equivalente a:

```
$data = "paco,2,3.6\nperico,5,1.7\npirolo,1.3,2.6\npocholo,11,-1.5";
```

Ejemplo 1: Si se llama al método `sort_csv($data, 1)` se obtiene como resultado

```
pirolo,1.3,2.6
paco,2,3.6
perico,5,1.7
pocholo,11,-1.5
```

Nota: Fíjese que están subrayados los elementos de la columna #1 por los cuales está ordenado. El resultado anterior es lo que se ve si se imprime en pantalla la salida. Pero el *string* resultante es:
`"pirolo,1.3,2.6\npaco,2,3.6\nperico,5,1.7\npocholo,11,-1.5";`

Ejemplo 2: Y si se llama al método `sort_csv($data, 2)` se obtiene

```
pocholo,11,-1.5
perico,5,1.7
pirolo,1.3,2.6
paco,2,3.6
```

Nota: Fíjese también que en los datos hay puntos (.) y comas (,). Las comas son el separador de datos del CSV y los puntos son parte de los números.

Ejercicio 2 (obligatorio)

Para el mundial de Fútbol se quiere crear una clase para controlar la fase clasificatoria por grupos. Se quiere crear la clase `Group` de forma tal que permita lo siguiente:

- El constructor de la clase recibe un *array* con 4 *strings* que representa los nombres de los equipos del grupo. Debe validar que la entrada sea correcta en caso contrario lanzar excepciones.
- La clase debe tener un método `match($team1, $score1, $team2, $score2)` que permite definir el resultado de un encuentro entre 2 equipos del grupo. Las variables `$team1` y `$team2` son *string* (los nombres) y las variables `$score1` y `$score2` es la cantidad de goles de cada equipo en ese partido. Debe validar que los nombres de los equipos sean correctos, que 2 equipos no jueguen más de una vez, etc.
- Debe implementar además un método `result()` que debe devolver la lista de los 4 equipos ordenados por clasificación hacia la próxima ronda. El método debe devolver en todo momento la tabla de posiciones actual, incluso aunque no se hayan jugado todos los partidos.
- Debe utilizarse la siguiente forma de calcular los puntos y determinar el desempate.
    a. Formas de ganar puntos
        i. Ganar un partido 3 puntos para el ganador
        ii. Perder un partido 0 puntos para el perdedor
        iii. Empatar un partido 1 punto para cada equipo.
    b. Formas de desempatar
        i. Mayor cantidad de puntos
        ii. Mayor diferencia de goles
        iii. Mayor números de goles a favor
    c. Si 2 o más equipos quedan empatados según los criterios anteriores entonces
        i. Mayor número de puntos obtenidos en los partidos entre ellos
        ii. Mayor diferencia de goles en los partidos entre ellos
        iii. Número de goles anotados en los partidos entre ellos

Nota: en el caso c) cuando se dice "los partidos entre ellos" se refiere a los partidos entre todos los empatados (que pueden ser 2, 3 o 4).

Por ejemplo, para la siguiente corrida (del grupo H del mundial de Fútbol de 2018):

```
$groupA = new Group('Colombia', 'Japón', 'Senegal', 'Polonia');
$groupA.match('Senegal', 0, 'Colombia', 1);
$groupA.match('Japón', 0, 'Polonia', 1);
$groupA.match('Senegal', 2, 'Japón', 2);
$groupA.match('Polonia', 0, 'Colombia', 3);
$groupA.match('Polonia', 1, 'Senegal', 2);
```

```
$groupA.match('Colombia', 1, 'Japón', 3);
$result = $groupA.result();
```

Los resultados son:

Equipo	Ganados	Empates	Perdidos	A Favor	En Contra	Diferencia	Puntos
Colombia	2	0	1	5	3	+2	6
Japón	1	1	1	5	4	+1	4
Senegal	1	1	1	4	4	0	4
Polonia	1	0	2	2	5	-3	3

El valor de la variable `$result` debería ser:

```
['Colombia', 'Japón', 'Senegal', 'Polonia']
```

Nota: fíjese que Japón y Senegal terminan con la misma cantidad de puntos pero clasifica Japón por tener una mayor diferencia de goles.

Ejercicios de Programación

Ejercicio 1 (obligatorio)

Se tiene un array de números enteros que puede contener elementos repetidos. Y se quiere obtener un nuevo array son los elementos del primer array, pero sin repeticiones. El array resultante debe tener **el mismo orden** que el anterior. En caso de haber elementos repetidos, en el array resultante **sólo debe aparecer la última ocurrencia** de ese elemento en el array original.

Ejemplo

Entrada: [1, 2, 10, 3, 4, 9, 5, 6, 8, 7, 8, 9, 10]
Salida: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Ejercicio 2 (obligatorio)

Dadas las siguientes definiciones.

- Un *string* es periódico si está compuesto por la repetición de un *substring*. Por ejemplo

    - `"blablablabla"` es periódico porque se repite `"bla"` 4 veces.
    - `"blablebli"` no es periódico.
    - `"blablabl"` tampoco es periódico.
- Un número entero es periódico, si su representación en binario es periódica. Por ejemplo:
    - El número 365 (`"101101101"`) es periódico porque el `"101"` se repite 3 veces.
    - El número 682 (`"1010101010"`) es periódico porque el `"10"` se repite 5 veces

Se quiere implementar una función (en el lenguaje de tu preferencia) que reciba como parámetro un número entero. Y devuelva el menor número periódico mayor que el que es recibido como parámetro.

```
def next_periodic(n)
  # esta es la función que tienes que implementar
end

# Ejemplo
input = 300
output = next_periodic(input)
# el output debería ser 365 porque es el primer número, mayor que 300
# que es períodico
```

Ejercicio 3 (obligatorio)

Implemente una función (en el lenguaje de su preferencia) que reciba como parámetro 8 números enteros: x1, y1, x2, y2, x3 y3, x4, y4.

- (x1, y1) representa un vértice de un rectángulo con los lados paralelos a los ejes de coordenadas
- (x2, y2) representa el vértice opuesto del rectángulo mencionado.
- (x3, y3) representa un vértice de un segundo rectángulo con los lados paralelos a los ejes de coordenadas.
- (x4, y4) representa el vértice opuesto del segundo rectángulo.

Se quiere que la función devuelva **el área de intersección de ambos rectángulos**. Si los rectángulos no se intersectan el área es cero.

Ejemplos:

Entrada (x1, y1, x2, y2, x3, y3, x4, y4)	Salida (Area)
0, 0, 20, 20, 10, 10, 30, 30	100
0, 20, 20, 0, 10, 30, 30, 10	100

0, 0, 30, 30, 10, 10, 20, 20	100
0, 20, 30, 0, 10, 10, 30, 20	200
0, 0, 10, 10, 20, 20, 30, 30	0

Ejercicio 4 (opcional)

Implemente una función (en el lenguaje de su preferencia) que reciba como parámetro un número entero N y devuelva **la cantidad de números super primos que hay entre cero y N**. Un número super primo se define de la siguiente forma:

- Si a un número primo, se le quita la cifra menos significativa y se mueve a la posición más significativa y el número resultante sigue siendo primo.
- Si repites el paso anterior, tantas veces como cifras tenga el número.
- Entonces el número dado es super primo (y todos los números intermedios también)

Ejemplo:

- El número 1193 es primo
- El número 3119 es primo → el 3 que estaba antes al final, ahora está al principio
- El número 9311 es primo → el 9 que antes estaba al final, ahora está al principio
- El número 1931 es primo → el 1 que antes estaba al final, ahora está al principio

Y como los 4 números son primos, entonces **el 1193 es un número super primo.**

Ejemplo de Entrada/Salida:

Entrada: 10 000
Salida: 33

Eso significa que hay 33 números super primos entre cero y 10 000.