

Correction du DS 1

Julien REICHERT

Partie 1

Pour les trois premiers exercices, on admet que tous les éléments des listes sont hachables. Pour le quatrième, l'énoncé précise que ce sont des entiers donc tout est en ordre.

Exercice 1.1

```
def plus_frequent(l):
    d = effectifs(l) # Définie plus loin
    rep = l[0] # Si l est vide, cela explosera, c'est prévu
    for x in d:
        if d[x] > d[rep]:
            rep = x
    return rep
```

Exercice 1.2

```
def memes_elements(l1, l2):
    d1, d2 = dict(), dict()
    for x1 in l1:
        d1[x1] = 42 # Des ensembles sont plus adaptés, mais hors programme
    for x2 in l2:
        d2[x2] = 42
    return d1 == d2
```

Exercice 1.3

```
def permutation(l1, l2):
    return effectifs(l1) == effectifs(l2)
```

Exercice 1.4

```
def montagne(l):
    d = effectifs(l)
    phase = 0
    indice_min = min(d.keys())
    indice_max = max(d.keys())
    if indice_max - indice_min != len(d.keys()) - 1:
        return False # il y a un trou, cela contredit forcément la condition
    for i in range(indice_min + 1, indice_max):
        if d[i] < d[i-1]:
            phase = 1
        elif phase == 1 and d[i] > d[i-1]:
            return False
    return True
```

```
def effectifs(l): # Utile pour les exercices 1.1, 1.3 et 1.4
    d = dict()
    for x in l:
        if x in d:
            d[x] += 1
        else:
            d[x] = 1
    return d
```

Partie 2

Exercice 2.1

- **Equipes** : chaque attribut est une clé, mais si déjà on crée un identifiant, ce sera la clé primaire.
- **Matches** : **Id_match** est une clé (clé primaire), **Date** ne sera pas une clé car il y a des jours avec plusieurs matchs (non pénalisé car tout le monde ne peut pas savoir), même en ajoutant **Phase** (qui n'est clairement pas une clé!), et le couple (**Equipe1**, **Equipe2**) non plus (Portugal-Grèce en 2004, par exemple). Cependant, il est vrai que le couple (**Date**, **Equipe1**) (idem avec **Equipe2**), ainsi que le triplet (**Phase**, **Equipe1**, **Equipe2**) (pas de match retour) est une clé. Les attributs **Equipe1** et **Equipe2**, au vu du type mentionné, sont des clés étrangères vers la table **Equipes**.
- **Parieurs** : cf. **Equipes**.
- **Paris** : seul le couple (**Parieur**, **Id_match**) a du sens. Il est formé de deux clés étrangères (tables intuitives).
- **Resultats** : l'attribut **Id_match** est une clé primaire, qui est par ailleurs une clé étrangère vers la table **Matches**.

Exercice 2.2

Comme on le voit sur l'énoncé des dernières questions, cette information se déduit du contenu des autres tables. Ceci étant, il n'est pas interdit qu'il y ait une redondance dans une base de données, donc la table **Paris** aurait pu avoir un attribut **Score_pari** valant **NULL** jusqu'à ce que le résultat du match soit connu, occasionnant une mise à jour en même temps que l'insertion du résultat dans la table **Resultats**.

Exercice 2.3

Les tables à deux attributs (identifiant et valeur non numérique) peuvent très bien disparaître au profit du remplacement dans toutes les tables où l'identifiant est mentionné en tant que clé étrangère de la valeur. Ceci permet une meilleure lecture par l'humain des données de ces tables, mais cette lecture n'est en général pas faite directement par l'utilisateur, et le surcoût en espace engendré suggère de maintenir la structure telle qu'elle a été présentée.

En outre, le résultat d'un match pourrait donner lieu à d'autres attributs dans la table des matchs, quitte à mettre à jour ces champs, initialement à **NULL** au moment de la création de l'enregistrement. C'est ici un choix personnel. Et comme annoncé, **NULL** est hors-programme.

Exercice 2.4

Les deux requêtes sont ici dans l'ordre.

```
SELECT Pays FROM Equipes WHERE Id_equipe = 19;
```

```
SELECT Id_equipe FROM Equipes WHERE Pays = "Qatar";
```

Ne pas oublier les guillemets !

On aurait pu faire le malin en écrivant pour la première `SELECT "Australie" AS Pays.`

Exercice 2.5

Un résultat avec les deux équipes (apprendre à faire des auto-jointures) :

```
SELECT Equipes.Pays AS Finaliste_1, Equipes2.Pays AS Finaliste_2
FROM Equipes JOIN Matches ON Equipes.Id_equipe = Matches.Equipe1
JOIN Equipes AS Equipes2 ON Equipes2.Id_equipe = Matches.Equipe2
WHERE Phase = "Finale"
```

Deux résultats, un par équipe (plus simple) :

```
SELECT Pays FROM Equipes WHERE Id_equipe IN
(SELECT Equipe1 AS Finaliste FROM Matches WHERE Phase = "Finale"
UNION
SELECT Equipe2 AS Finaliste FROM Matches WHERE Phase = "Finale")
```

Il est aussi possible de faire une réunion de deux tables obtenues par une jointure (principe analogue).

Exercice 2.6

On peut faire comme l'exercice précédent de deux façons différentes. La plus simple est sans réunion.

```
SELECT Equipes.Pays AS Equipe_1, Equipes2.Pays AS Equipe_2
FROM Equipes JOIN Matches ON Equipes.Id_equipe = Matches.Equipe1
JOIN Equipes AS Equipes2 ON Equipes2.Id_equipe = Matches.Equipe2
WHERE Date = (SELECT MIN(Date) FROM Matches)
```

Exercice 2.7

Seul le couple formé par le parieur et l'identifiant du match ont de l'intérêt en vue de ce qui suit.

```
SELECT Parieur, Paris.Id_match FROM Paris
JOIN Resultats ON Paris.Id_match = Resultats.Id_match
WHERE (Score1 > Score2 AND Buts1 > Buts2)
OR (Score1 < Score2 AND Buts1 < Buts2)
OR (Score1 = Score2 AND Buts1 = Buts2)
```

Exercice 2.8

Même remarque.

```
SELECT Parieur, Paris.Id_match FROM Paris
JOIN Resultats ON Paris.Id_match = Resultats.Id_match
WHERE Score1 = Buts1 AND Score2 = Buts2
```

Exercice 2.9

```
SELECT Parieur, COUNT(*) AS Points FROM succes
GROUP BY Parieur ORDER BY Points DESC
```

Exercice 2.10

On va éviter la programmation par copier-coller et écrire une première fonction qui retourne les deux informations nécessaires, ce qui facilite les fonctions attendues dans un premier temps.

Le tri sera fait dans une troisième fonction, sans objectif de complexité on se contentera d'un tri sélection non en place.

```

def deux_scores(PARIS, RESULTATS):
    parieurs = [] # Ou un dictionnaire
    points1 = []
    points2 = []
    for (parieur, match, score1, score2) in PARIS:
        for (id_match, buts1, buts2) in RESULTATS: # sans garantie sur l'ordre...
            if match == id_match:
                if (score1, score2) == (buts1, buts2):
                    pt2 = 1
                if score1 == score2 and buts1 == buts2:
                    pt1 = 1
                if score1 > score2 and buts1 > buts2:
                    pt1 = 1
                if score1 < score2 and buts1 < buts2:
                    pt1 = 1
                if pt1 > 0: # pt2 ne peut valoir 1 que si pt1 aussi
                    if parieur in parieurs:
                        indice = parieur.index(parieurs)
                    else:
                        parieurs.append(parieur)
                        points1.append(0)
                        points2.append(0)
                        indice = -1
                    points1[indice] += pt1
                    points2[indice] += pt2
                break # pas nécessaire, pas fan du mot-clé, mais autant le signaler
    return parieurs, points1, points2 # Aucun point = parieur non mentionné.

def score1(PARIS, RESULTATS):
    parieurs, points1, _ = deux_scores(PARIS, RESULTATS)
    return zip(parieurs, points1) # merci zip !

def score2(PARIS, RESULTATS):
    parieurs, _, points2 = deux_scores(PARIS, RESULTATS)
    return zip(parieurs, points2)

def classement(points):
    n = len(points)
    reponse = []
    pointsbis = points[:]
    for _ in range(n):
        maxi = 0
        for i in range(1, n):
            if pointsbis[i][1] > pointsbis[maxi][1]:
                maxi = i
        reponse.append(pointsbis[maxi])
        pointsbis[maxi] = ("", -1)
    return reponse

```

Partie 3

Exercice 3.1

Il faudrait une table `Descriptions` avec trois attributs, `piece_unique` de type entier, `avers` de type chaîne de caractères (voire texte, car cela peut être long) et `revers` de type chaîne de caractères (même remarque). Dans ce cas, la table `Pieces` peut avoir un attribut supplémentaire qui serait une clé étrangère vers l'attribut `piece_unique` de la table `Descriptions` pas forcément identique à l'identifiant de la pièce.

Autre possibilité : créer des descriptions sans tenir compte de l'avers ou du revers dans une table avec un identifiant de description et un texte, dans ce cas on peut renvoyer à ces identifiants de description avec deux clés étrangères dans la table `Pieces`.

Dans les deux cas, il peut être intéressant d'utiliser une description identique pour des pièces différentes car émises une autre année, voire mieux dans le deuxième cas en rassemblant des pièces différentes partageant un côté, par exemple les euros d'un pays qui ont le même côté face quelle que soit la valeur (l'Estonie est dans ce cas).

Exercice 3.2

```
SELECT COUNT(*) FROM Pieces
```

Exercice 3.3

```
SELECT COUNT(*) FROM  
(  
  SELECT pays, valeur, unite, annee, valable  
  FROM Pieces GROUP BY pays, valeur, unite, annee, valable  
) AS td
```

ou, en gérant bien le `DISTINCT`,

```
SELECT COUNT(DISTINCT pays, valeur, unite, annee, valable) FROM Pieces
```

Exercice 3.4

Il y a au moins une pièce quand l'enregistrement existe, ce n'est pas nécessaire d'incorporer la condition en question.

```
SELECT COUNT(DISTINCT valise) FROM Rangement
```

Là aussi, une sous-requête avec une table dérivée peut s'imaginer.

Exercice 3.5

```
SELECT valise FROM Rangement GROUP BY valise ORDER BY COUNT(*) DESC LIMIT 1
```

Pour tenir compte des égalités, on peut compléter :

```
SELECT valise FROM Rangement GROUP BY valise HAVING COUNT(*) =  
(SELECT COUNT(*) FROM Rangement GROUP BY valise ORDER BY COUNT(*) DESC LIMIT 1)
```

Exercice 3.6

```
SELECT valeur, nom_unite, valise, compartiment, rangee, colonne, 2022 - annee  
FROM Pieces JOIN Unites ON unite = id_unite  
JOIN Rangement ON id_piece = piece  
WHERE annee = (SELECT MIN(annee) FROM Pieces)
```

Exercice 3.7

Avec des flottants, inutile de craindre une égalité.

```
SELECT symbole FROM Unites JOIN Cours ON id_unite = monnaie
ORDER BY euros DESC LIMIT 1
```

Exercice 3.8

```
SELECT valise, compartiment, rangee, colonne
FROM Rangement JOIN Pieces ON piece = id_piece
JOIN Unites ON id_unite = unite
WHERE nom_unite = "złoty"
```

Exercice 3.9

```
SELECT SUM(valeur * euros) FROM Pieces
JOIN Cours ON unite = monnaie
WHERE valable
```

Exercice 3.10

```
def fortune(PIECES, COURS):
    rep = 0.
    for sextuplet in PIECES:
        if sextuplet[5]:
            valeur = sextuplet[2]
            unite = sextuplet[3]
            for (monnaie, euros) in COURS:
                if monnaie == unite:
                    rep += valeur * euros
    return rep
```