Correction du DS 1

Julien Reichert

Partie 1

```
# Exercice 1.1

def sommepuissances(n, k):
    rep = 0
    for i in range(1, n+1): # Attention au +1
        rep += i ** k
    return rep

# Exercice 1.2

def majoritaire(l): # Plein d'algorithmes possibles, utilisons un dictionnaire ici.
    dico = {}
    for x in l:
        if x in dico:
            dico[x] += 1
        else:
            dico[x] = 1
    for cle in dico:
        if 2 * dico[cle] > len(l):
            return True
    return False

# Exercice 1.3

def posmax2(ll):
    rep = None # Initialiser avec ll[0][0] a été accepté mais des listes peuvent être vides
    for i in range(len(ll)):
        for j in range(len(ll[i])):
            if rep is None or ll[i][j] > ll[rep[0]][rep[1]]:
                rep = (i, j)
    return rep

# Exercice 1.4

def pgenfp2im(n):
    if n == 0: # Ne pas oublier ce cas !
        return 0
    while n <= 1000000000:
        n *= 2
    return n // 2 # Sinon c'est incorrect !
```

```
# Exercice 1.5

def premiers_entre_eux(x, y):
    if x == 1 or y == 1:
        return True
    if x == 0 or y == 0:
        return False
    while y != 0:
        x, y = y, x % y
    return x == 1
```

Partie 2

Exercice 2.1

- `SERIE` : chaque attribut est une clé, mais si déjà on crée un identifiant, ce sera la clé primaire.
- `COLLECTION` : `id_serie` est une clé étrangère, et `id_collection` est une clé au vu des explications de l'énoncé, mais `titre_collection` pas forcément car il est possible d'utiliser un terme comme « série originale ». Cependant, il faut que le couple `id_serie, titre_collection` soit une clé.
- `AUTEUR` : `id_serie` est une clé étrangère, et quitte à utiliser des artifices on préfère que `nom_auteur` n'ait pas d'homonymie, mais pour autant ce n'est pas une clé car une même personne peut être l'auteur de plusieurs séries. Et même le couple `(id_serie, nom_auteur)` n'est pas une clé car on peut être scénariste et dessinateur d'une même série (par exemple Albert Uderzo après la mort de René Goscinny pour Astérix). Finalement, le triplet des arguments est une clé car on ne voudrait pas de doublon.
- `ALBUM` : `id_collection` est une clé étrangère, et le couple `(id_collection, numero)` est censé être une clé (on peut aussi remplacer le numéro par le titre pour avoir une autre clé).

Exercice 2.2

Les entités concernées sont les séries (ou les collections, dans le doute les deux tables ont été créées) de bande dessinée et les albums (les auteurs n'ont pas fait l'objet d'une table pour simplifier mais ce serait pleinement légitime de les considérer comme des entités).

Quant à la table décrivant une relation, on utilise la phrase « l'auteur crée une bande dessinée », en remplaçant éventuellement le verbe créer par un verbe adapté au rôle de l'auteur.

Exercice 2.3

La relation représentée par la table `AUTEUR` est de type `*-*` car un auteur peut contribuer à plusieurs séries et une série peut avoir plusieurs auteurs.

Exercice 2.4

```
SELECT titre FROM SERIE WHERE id = 19
```

Exercice 2.5

```
SELECT DISTINCT nom_auteur FROM AUTEUR JOIN SERIE ON id_serie = id WHERE titre = 'Tintin'
```

Inutile de préciser le rôle (on ne peut pas en présumer sans voir les données), mais on peut éviter d'avoir plusieurs fois le nom d'Hergé s'il est enregistré pour tous les rôles grâce à `DISTINCT` (ce n'est pas crucial pour avoir tous les points).

Exercice 2.6

```
SELECT MIN(date) FROM ALBUM
```

Exercice 2.7

```
SELECT COUNT(*) FROM ALBUM WHERE possede
```

(Une version élégante s'imagine, avec `SUM(possede)` sans `WHERE`.)

Exercice 2.8

Version simple :

```
SELECT id_serie
FROM ALBUM
JOIN COLLECTION ON ALBUM.id_collection = COLLECTION.id_collection
WHERE possede
GROUP BY id_serie
ORDER BY COUNT(*) DESC LIMIT 1
```

Version tenant compte des égalités :

```
SELECT id_serie FROM ALBUM
JOIN COLLECTION ON ALBUM.id_collection = COLLECTION.id_collection
WHERE possede
GROUP BY id_serie
HAVING COUNT(*) =
(
  SELECT COUNT(*) AS nombre_maximal FROM ALBUM
  JOIN COLLECTION ON ALBUM.id_collection = COLLECTION.id_collection
  WHERE possede
  GROUP BY id_serie
  ORDER BY COUNT(*) DESC LIMIT 1
)
```

Exercice 2.9

```
SELECT SERIE.titre, COUNT(DISTINCT id_collection) AS nombre_collections, COUNT(*) AS nombre_albums
FROM SERIE JOIN COLLECTION ON id = id_serie
JOIN ALBUM ON COLLECTION.id_collection = ALBUM.id_collection
GROUP BY SERIE.titre
```

Partie 3

Exercice 3.1

Si on cherche une clé primaire, il faut permettre à un client de répondre à toutes les questions d'une séance, mais aussi de participer à plusieurs séances, dont on numérotera toujours les questions de `1` à `40`, et bien entendu il faut permettre à plusieurs clients de participer à une séance.

Ceci étant, une réponse ne peut être donnée qu'une fois, ce qui impose pour la table `REPONSES` la clé primaire `(Id_seance, Id_client, Question)`, et de manière analogue pour la table `SOLUTIONS` la clé primaire `(Id_seance, Question)`.

Exercice 3.2

Pour donner les solutions aux questions d'une séance, on peut les affecter à un identifiant de client fictif (par exemple `-1`) et donc se servir de la table `REPONSES` uniquement.

Par ailleurs, le nombre de fautes à une séance peut se déduire en étudiant les tables `REPONSES` et `SOLUTIONS` (ou la version adaptée) pour le client et la séance en question.

Exercice 3.3

Dans l'ordre...

- `SELECT COUNT(*) FROM CLIENTS;`
- `SELECT DISTINCT Date FROM SEANCES`
  ou `SELECT Date FROM SEANCES GROUP BY Date;`
- `SELECT AVG(NbFautes) FROM SEANCES WHERE Id_client = n;`
- `SELECT MIN(NbFautes) FROM SEANCES;`
- `SELECT MAX(NbQuestions) FROM`
  `(SELECT COUNT(*) AS NbQuestions FROM SOLUTIONS GROUP BY Reponses) AS tablederivee;`
- `SELECT COUNT(*) FROM REPONSES AS R JOIN SOLUTIONS AS S`
  `ON R.Id_seance = S.Id_seance AND R.Question = S.Question`
  `WHERE Id_client = n AND R.Id_seance = s AND R.Reponses <> S.Reponses;`

Exercice 3.4

Le mieux est de construire les requêtes étape par étape.

On récupère donc les dates des trois dernières séances pour un client particulier d'identifiant noté `n` :

```
SELECT Date FROM SEANCES WHERE Id_client = n ORDER BY Date DESC LIMIT 3
```

Ensuite, pour les nombres de fautes associés, il suffit de voir si le maximum est inférieur ou égal à cinq, en vérifiant aussi qu'il y a au moins trois séances, donc on injecte :

```
SELECT COUNT(*), MAX(NbFautes) FROM
(SELECT * FROM SEANCES WHERE Id_client = n ORDER BY Date DESC LIMIT 3) AS tablederivee
```

À présent, on passe à la jointure pour se servir du nom :

```
SELECT COUNT(*) = 3 AND MAX(NbFautes) <= 5 AS Pret
FROM
(
  SELECT * FROM SEANCES
  JOIN CLIENTS ON Id = Id_client
  WHERE NomPrenom = nom
  ORDER BY Date DESC LIMIT 3
) AS tablederivee
```

En pratique, pour obtenir la liste des tels clients, le mieux est de faire une boucle dans un langage externe grâce auquel on peut faire des requêtes, et on peut même faire le traitement avec ce langage pour ne pas avoir à écrire des requêtes trop compliquées (suivant le degré de maîtrise dans les deux langages...).

Exercice 3.5

Pour obtenir le nombre de séances au total, on peut par exemple écrire

```
SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS
```

Pour filtrer les clients ayant effectué toutes les séances, on fait un regroupement et on utilise `HAVING` :

```
SELECT Id_client
FROM SEANCES
GROUP BY Id_client
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
```

De même, pour obtenir le nombre minimal de fautes au total parmi de tels clients, on récupère le nombre total de fautes pour chacun de ces clients :

```
SELECT SUM(NbFautes)
FROM SEANCES
GROUP BY Id_client
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
```

On obtient une table dérivée dont on recherche le minimum, qui doit être le nombre total de fautes du client sélectionné :

```
SELECT Id_client
FROM SEANCES
GROUP BY Id_client
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
AND SUM(NbFautes) =
(
  SELECT MIN(Total)
  FROM
  (
    SELECT SUM(NbFautes) AS Total
    FROM SEANCES
    GROUP BY Id_client
    HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
  ) AS tablederivee
)
```

Ici, on n'a pas profité de la possibilité de supposer l'existence et l'unicité, par ailleurs. Sinon c'est plus facile, au moins.