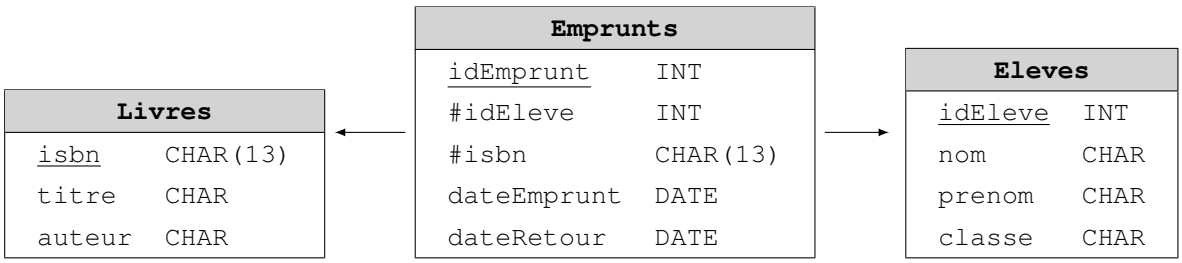
Métropole - sujet 2 - juin 2021

Exercice 1 (SQL - 4 points)
L'énoncé de cet exercice utilise les mots du langage SQL suivants : `SELECT FROM`, `WHERE`, `JOIN ON`, `INSERT INTO VALUES`, `UPDATE`, `SET`, `DELETE`, `COUNT`, `AND` et `OR`.

On considère dans cet exercice une gestion simplifiée des emprunts des ouvrages d'un CDI. La base de données utilisée sera constituée de trois relations (ou tables) nommées `Eleves`, `Livres` et `Emprunts` selon le schéma relationnel suivant :



Dans ce schéma relationnel, un attribut souligné indique qu'il s'agit d'une clé primaire.
Le symbole # devant un attribut indique qu'il s'agit d'une clé étrangère et la flèche associée indique l'attribut référencé. Ainsi, l'attribut `idEleve` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `idEleve` de la relation `Eleves`.

1. Expliquer pourquoi le code SQL ci-dessous provoque une erreur :

```
INSERT INTO Eleves VALUES (128, 'Dupont', 'Jean', 'T1')
INSERT INTO Eleves VALUES (200, 'Dupont', 'Jean', 'T1')
INSERT INTO Eleves VALUES (128, 'Dubois', 'Jean', 'T2')
```

2. Dans la définition de la relation `Emprunts`, qu'est-ce qui assure qu'on ne peut pas enregistrer un emprunt pour un élève qui n'a pas encore été inscrit dans la relation `Eleves` ?
3. Ecrire une requête SQL qui renvoie les titres des ouvrages de Molière détenus par le CDI.
4. Décrire le résultat renvoyé par la requête ci-dessous :

```
SELECT COUNT(*)
FROM Eleves
WHERE classe = 'T2'
```

5. Camille a emprunté le livre *Les Misérables*. Le code ci-dessous a permis d'enregistrer cet emprunt :

```
INSERT INTO Emprunts
VALUES (640, 192, '9782070409228', '2020-09-15', NULL)
```

Camille a restitué le livre le 30 septembre 2020.
Recopier et compléter la requête ci-dessous de manière à mettre à jour la date de retour dans la base de données.

```
UPDATE Emprunts SET ..... WHERE .....
```

6. Décrire le résultat renvoyé par la requête ci-dessous :

```
SELECT DISTINCT nom, prenom
FROM Eleves, Emprunts
WHERE Eleves.idEleve = Emprunts.idEleve AND Eleves.classe = 'T2'
```

7. Ecrire une requête SQL qui permet de lister les noms et prénoms des élèves qui ont emprunté le livre *Les Misérables*.

Exercice 2 (Processus - 4 points)

1. Les états possibles d'un processus sont : *prêt*, *élu*, *terminé* et *bloqué*.
   (a) Expliquer à quoi correspond l'état *élu*.
   (b) Proposer un schéma illustrant les passages entre les différents états.
2. On suppose que quatre processus C1, C2, C3 et C4 sont créés sur un ordinateur, et qu'aucun autre processus n'est lancé sur celui-ci, ni préalablement ni pendant l'exécution des quatre processus. L'ordonnanceur, pour exécuter les différents processus prêts, les place dans une structure de données de type file. Un processus prêt est enfilé et un processus élu est défilé.
   (a) Parmi les propositions suivantes, recopier celle qui décrit le fonctionnement des entrées/sorties dans une file :
      i. Premier entré, dernier sorti
      ii. Premier entré, premier sorti
      iii. Dernier entré, premier sorti
   (b) On suppose que les quatre processus arrivent dans la file et y sont placés dans l'ordre C1, C2, C3 et C4.
      - Les temps d'exécution totaux de C1, C2, C3 et C4 sont respectivement 100 ms, 150 ms, 80 ms et 60 ms.
      - Après 40 ms d'exécution, le processus C1 demande une opération d'écriture disque, opération qui dure 200 ms. Pendant cette opération d'écriture, le processus C1 passe à l'état bloqué.
      - Après 20 ms d'exécution, le processus C3 demande une opération d'écriture disque, opération qui dure 10 ms. Pendant cette opération d'écriture, le processus C3 passe à l'état bloqué.

   Sur la frise chronologique donnée en annexe (à rendre avec la copie), les états du processus C2 sont donnés.
   Compléter la frise avec les états des processus C1, C3 et C4.
3. On trouvera ci- dessous deux programmes rédigés en pseudo-code.
   Verrouiller un fichier signifie que le programme demande un accès exclusif au fichier et l'obtient si le fichier est disponible.

Programme 1	Programme 2
Verrouiller fichier_1	Verrouiller fichier_2
Calculs sur fichier_1	Verrouiller fichier_1
Verrouiller fichier_2	Calculs sur fichier_1
Calculs sur fichier_1	Calculs sur fichier_2
Calculs sur fichier_2	Déverrouiller fichier_1
Calculs sur fichier_1	Déverrouiller fichier_2
Déverrouiller fichier_2	
Déverrouiller fichier_1	

   (a) En supposant que les processus correspondant à ces programmes s'exécutent simultanément (exécution concurrente), expliquer le problème qui peut être rencontré.
   (b) Proposer une modification du programme 2 permettant d'éviter ce problème.

Exercice 3 (Arbre binaire de recherche et POO - 4 points)

On rappelle qu'un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit. Un nœud sans sous-arbre est appelé feuille. La taille d'un arbre est le nombre de nœuds qu'il contient ; sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles. Ainsi la hauteur d'un arbre réduit à un nœud, c'est-à-dire la racine, est 1.

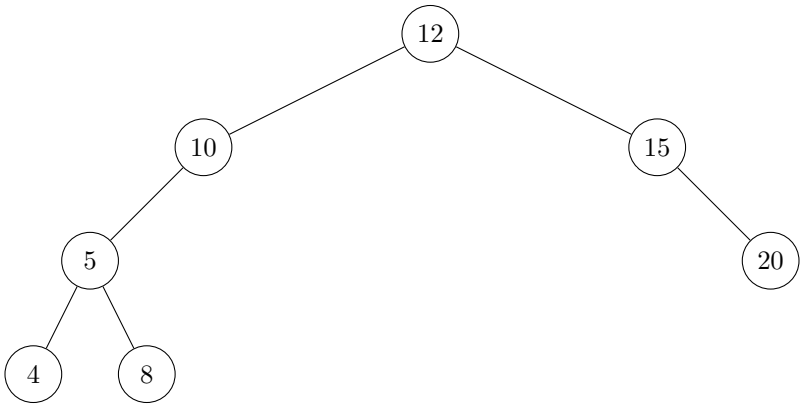
Dans un arbre binaire de recherche, chaque nœud contient une clé, ici un nombre entier, qui est :

- strictement supérieure à toutes les clés des nœuds du sous-arbre gauche ;
- strictement inférieure à toutes les clés des nœuds du sous-arbre droit.

Ainsi les clés de cet arbre sont toutes distinctes.

Un arbre binaire de recherche est dit « bien construit » s'il n'existe pas d'arbre de hauteur inférieure qui pourrait contenir tous ses nœuds.

On considère l'arbre binaire de recherche ci-dessous :



1. (a) Quelle est la taille de l'arbre ci-dessus ?
   (b) Quelle est la hauteur de l'arbre ci-dessus ?
2. Cet arbre binaire de recherche n'est pas « bien construit ». Proposer un arbre binaire de recherche contenant les mêmes clés et dont la hauteur est plus petite que celle de l'arbre initial.
3. Les classes `Noeud` et `Arbre` ci-dessous permettent de mettre en oeuvre en `Python` la structure d'arbre binaire de recherche. La méthode `insere` permet d'insérer récursivement une nouvelle clé.

```
class Noeud :
    def __init__(self, cle):
        self.cle = cle
        self.gauche = None
        self.droit = None

    def insere(self, cle):
        if cle < self.cle :
            if self.gauche == None :
                self.gauche = Noeud(cle)
            else :
                self.gauche.insere(cle)
        elif cle > self.cle
            if self.droit == None :
                self.droit = Noeud(cle)
            else :
                self.droit.insere(cle)


class Arbre :
    def __init__(self, cle):
        self.racine = Noeud(cle)

    def insere(self, cle):
        self.racine.insere(cle)
```

Donner la représentation de l'arbre codé par les instructions ci-dessous :

```
a = Arbre(10)
a.insere(20)
a.insere(15)
a.insere(12)
a.insere(8)
a.insere(4)
a.insere(5)
```

4. Pour calculer la hauteur d'un arbre non vide, on a écrit la méthode ci-dessous dans la classe `Noeud` :

```
def hauteur(self):
    if self.gauche == None and self.droit == None:
        return 1
    if self.gauche == None:
        return 1+self.droit.hauteur()
    elif self.droit == None:
        return 1+self.gauche.hauteur()
    else:
        hg = self.gauche.hauteur()
        hd = self.droit.hauteur()
        if hg > hd:
            return hg+1
        else:
            return hd+1
```

Ecrire la méthode `hauteur` de la classe `Arbre` qui renvoie la hauteur de l'arbre.

5. Ecrire les méthodes `taille` des classes `Noeud` et `Arbre` permettant de calculer la taille d'un arbre.
6. On souhaite écrire une méthode `bien_construit` de la classe `Arbre` qui renvoie la valeur `True` si l'arbre est « bien construit » et `False` sinon.
   On rappelle que la taille maximale d'un arbre binaire de recherche de hauteur h est $2^h - 1$.
   (a) Quelle est la taille minimale, notée t_{min}, d'un arbre binaire de recherche « bien construit » de hauteur h ?
   (b) Ecrire la méthode `bien_construit` demandée.

Exercice 4 (Programmation et récursivité - 4 points)

On s'intéresse dans cet exercice à un algorithme de mélange des éléments d'une liste.

1. Pour la suite, il sera utile de disposer d'une fonction echange qui permet d'échanger dans une liste `lst` les éléments d'indice `i1` et `i2`. Expliquer pourquoi le code `Python` ci-dessous ne réalise pas cet échange et en proposer une modification.

```
def echange(lst, i1, i2):
    lst[i2] = lst[i1]
    lst[i1] = lst[i2]
```

2. La documentation du module `random` de `Python` fournit les informations ci-dessous concernant la fonction `randint(a,b)` :

```
# Renvoie un entier aléatoire N tel que a <= N <= b.
# Alias pour randrange(a,b+1).}
```

Parmi les valeurs ci-dessous, quelles sont celles qui peuvent être renvoyées par l'appel `randint(0, 10)` ?

`0` `1` `3.5` `9` `10` `11`

3. Le mélange de Fischer Yates est un algorithme permettant de permuter aléatoirement les éléments d'une liste. On donne ci-dessous une mise en oeuvre récursive de cet algorithme en `Python`.

```
from random import randint
def melange(lst, ind):
    print(lst)
    if ind > 0:
        j = randint(0, ind)
        echange(lst, ind, j)
        melange(lst, ind-1)
```

(a) Expliquer pourquoi la fonction `melange` se termine toujours.

(b) Lors de l'appel de la fonction `melange`, la valeur du paramètre `ind` doit être égal au plus grand indice possible de la liste `lst`. Pour une liste de longueur n, quel est le nombre d'appels récursifs de la fonction `melange` effectués, sans compter l'appel initial ?

(c) On considère le script ci-dessous :

```
lst = [v for v in range(5)]
melange(lst, 4)
```

On suppose que les valeurs successivement renvoyées par la fonction `randint` sont `2`, `1`, `2` et `0`.
Les deux premiers affichages produits par l'instruction `print(lst)` de la fonction `melange` sont :

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
```

Donner les affichages suivants produits par la fonction `melange`.

(d) Proposer une version itérative du mélange de Fischer Yates.

Exercice 5 (Programmation - 4 points)

Etant donné un tableau non vide de nombres entiers relatifs, on appelle sous-séquence une suite non vide d'éléments voisins de ce tableau. On cherche dans cet exercice à déterminer la plus grande somme possible obtenue en additionnant les éléments d'une sous-séquence.

Par exemple, pour le tableau ci-dessous, la somme maximale vaut 18. Elle est obtenue en additionnant les éléments de la sous-séquence encadrée en gras ci-dessous `(6 ; 8 ; -6 ; 10)`.

-8	-4	**6**	**8**	**-6**	**10**	-4	-4

1. (a) Quelle est la solution du problème si les éléments du tableau sont tous positifs ?
   (b) Quelle est la solution du problème si tous les éléments sont négatifs ?
2. Dans cette question, on examine toutes les sous-séquences possibles.
   (a) Ecrire le code `Python` d'une fonction `somme_sous_sequence(lst, i, j)` qui prend en argument une liste `lst` et deux entiers `i` et `j` et qui renvoie la somme de la sous-séquence délimitée par les indices `i` et `j` (inclus).
   (b) La fonction `pgsp` ci-dessous permet de déterminer la plus grande des sommes obtenues en additionnant les éléments de toutes les sous-séquences possibles du tableau `lst` :

```
def pgsp(lst):
    n = len(lst)
    somme_max = lst[0]
    for i in range(n):
        for j in range(i, n):
            s = somme_sous_sequence(lst, i, j)
            if s > somme_max :
                somme_max = s
    return somme_max
```

   Parmi les quatre choix suivants, quel est le nombre de comparaisons effectuées par cette fonction si le tableau `lst` passé en paramètre contient 10 éléments ?

   10 55 100 1055

   (c) Recopier et modifier la fonction `pgsp` pour qu'elle renvoie un tuple contenant la somme maximale et les indices qui délimitent la sous-séquence correspondant à cette somme maximale.
3. On considère dans cette question une approche plus élaborée. Son principe consiste, pour toutes les valeurs possibles de l'indice `i`, à déterminer la somme maximale `S(i)` des sous-séquences qui se terminent à l'indice `i`. En désignant par `lst[i]` l'élément de `lst` d'indice `i`, on peut vérifier que
   - `S(0) = lst[0]`
   - et pour `i` $\geqslant 1$:

     `S(i) = lst[i]` si `S(i-1)` $\leqslant 0$;

     `S(i) = lst[i] + S(i-1)` si `S(i-1)` > 0.

   (a) Recopier et compléter le tableau ci-dessous avec les valeurs de `S(i)` pour la liste considérée en exemple.

i	0	1	2	3	4	5	6	7
lst[i]	-8	-4	6	8	-6	10	-4	-4
S(i)								

   (b) La solution au problème étant la plus grande valeur des `S(i)`, on demande de compléter la fonction `pgsp2` ci-dessous, de sorte que la variable `sommes_max` contienne la liste des valeurs `S(i)`.

```
def pgsp2(lst):
    sommes_max = [lst[0]]
    for i in range(1, len(lst)):

        # à compléter

    return max(sommes_max)
```

   (c) En quoi la solution obtenue par cette approche est-elle plus avantageuse que celle de la question 2.b. ?

Annexe de l'exercice 2 à rendre avec la copie.

Question 2.b.