Exercice 1 (6 points)

Cet exercice porte sur les bases de données et les requêtes SQL, les arbres binaires et les algorithmes sur les arbres binaires.

Partie A

Dans cet exercice, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT` , `FROM` , `WHERE` (avec les opérateurs logiques `AND` , `OR`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE` , `INSERT` , `DELETE`.

Une exoplanète est une planète située hors du système solaire. La plupart des exoplanètes découvertes à ce jour orbitent autour d'une unique étoile.

Une étoile est repérée précisément dans le ciel par son ascension droite et sa déclinaison (voir Figure 1). La direction de coordonnées `(0, 0)` est une direction fixe du ciel servant d'origine de ce système de coordonnées.

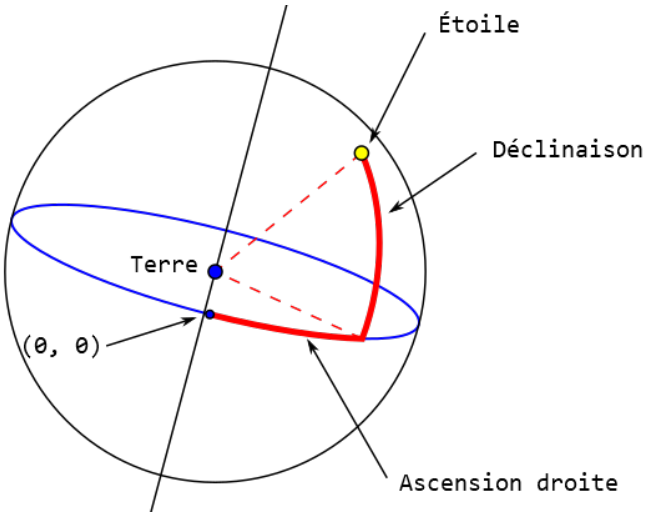

Figure 1. Coordonnées d'une étoile (adaptée depuis *https://commons.wikimedia.org/wiki/File:Coordonnees_equatoriales.svg*)

On considère dans cet exercice deux relations décrivant des étoiles et les exoplanètes orbitant autour d'elles :

- la relation `Etoiles` contient les informations décrivant des étoiles :
  - `id_etoile` : l'identifiant unique de l'étoile (nombre entier) ;
  - `nom` : le nom de l'étoile (chaîne de caractères) ;
  - `ascension` : l'ascension droite de l'étoile en degré (nombre réel) ;
  - `declinaison` : la déclinaison de l'étoile en degré (nombre réel).

- la relation `Exoplanetes` contient les informations décrivant des exoplanètes :
  - `id_exoplanete` : l'identifiant unique de l'exoplanète (nombre entier) ;
  - `masse` : la masse de l'exoplanète, exprimée sous la forme d'une fraction de la masse de la planète Jupiter (nombre réel) ;
  - `rayon` : le rayon de l'exoplanète, exprimée sous la forme d'une fraction du rayon de la planète Jupiter (nombre réel) ;
  - `id_etoile` : l'identifiant de l'étoile autour de laquelle orbite l'exoplanète (nombre entier).

Une exoplanète dont l'attribut `masse` est égal à `6.84` a une masse 6,84 fois plus grande que celle de la planète Jupiter.

On fournit ci-dessous des extraits de ces deux tables :

Etoiles			
id_etoile	nom	ascension	declinaison
1	109 Psc	26.23	20.08
2	beta Pic	86.82	-51.07
3	K2-21	340.30	-14.49
4	Kepler-11	297.12	41.91

Exoplanetes			
id_exoplanete	masse	rayon	id_etoile
1	6.84	1.15	1
2	11.90	1.65	2
3	8.89	1.20	2
4	0.01	0.16	3
5	0.02	0.22	3
6	0.01	0.16	4
7	0.01	0.26	4

L'attribut `id_exoplanete` est la clé primaire de la relation `Exoplanetes`. L'attribut `id_etoile` est la clé primaire de la relation `Etoiles`.

1. Expliquer pourquoi l'attribut `masse` de la relation `Exoplanetes` ne peut pas servir de clé primaire de cette relation.

2. Donner le nom de l'attribut pouvant être utilisé comme clé étrangère dans la relation `Exoplanetes`. Expliquer son rôle.

3. Donner le résultat de la requête SQL suivante :

```
SELECT masse, rayon
FROM Exoplanetes
WHERE id_exoplanete = 4;
```

4. Écrire une requête SQL permettant d'obtenir l'identifiant et le nom des étoiles dont l'ascension droite est supérieure ou égale à 100 degrés.

On souhaite insérer une nouvelle exoplanète de rayon égal à 0,37 fois celui de Jupiter et pesant 0,03 fois la masse de Jupiter. Cette exoplanète orbite autour de l'étoile Kepler-11 dont l'identifiant est `4`. On pourra attribuer à cette nouvelle exoplanète l'identifiant `9` qui n'apparaît pas dans la relation `Exoplanetes`.

5. Écrire une requête SQL permettant d'insérer cette nouvelle exoplanète dans la base de données.

6. Écrire une requête SQL permettant d'obtenir les rayons des exoplanètes orbitant autour de l'étoile nommée Kepler-11, dont l'identifiant est supposé non connu.

Partie B

On souhaite désormais écrire une application Python permettant de classer et de retrouver efficacement les étoiles selon leur position dans le ciel.

On rappelle qu'une étoile est repérée par son ascension droite et sa déclinaison. Par souci de simplicité, on considère désormais que deux étoiles ont toujours des coordonnées entières et distinctes. On représente en Python les coordonnées d'une étoile par un tuple d'entiers `(ascension, declinaison)`.

Dans la suite, on considère les étoiles dont les coordonnées sont contenues dans la liste de tuples `etoiles` définie par `etoiles = [(29, 21), (17, 14), (10, 30), (35, 13), (30, 63), (15, 20)]`.

On cherche à construire un arbre binaire de recherche à partir des coordonnées présentes dans la liste `etoiles` afin d'accélérer les opérations de traitement sur celles-ci. Pour cela :

- on commence par trier la liste `etoiles` par ordre croissant, afin que l'arbre résultant soit de hauteur minimale ;
- pour construire l'arbre binaire de recherche à partir des éléments de la liste `etoiles` compris entre les indices `debut` (inclu) et `fin` (exclu) :
  - la racine de l'arbre est l'élément d'indice `milieu` définit par

    `milieu = (debut + fin)//2` ;

– on construit récursivement le sous arbre gauche à l'aide des éléments de la liste `etoiles` compris entre les indices `debut` (inclu) et `milieu` (exclu) ;
– on construit récursivement le sous arbre droit à l'aide des éléments de la liste `etoiles` compris entre les indices `milieu + 1` (inclu) et `fin` (exclu).

Pour implémenter cet algorithme, on représente en Python les arbres binaires non vides à l'aide de tuples de trois éléments `(sag, position, sad)` dans lesquels :

- `position` est la valeur de la racine. Cette valeur est le couple de coordonnées permettant de repérer l'étoile ;
- `sag` et `sad` sont respectivement les sous-arbres gauche et droit de l'arbre.

L'arbre vide est quant à lui représenté par `None`.

On rappelle que l'on peut comparer des tuples en Python à l'aide de l'opérateur < : on compare tout d'abord les valeurs à l'indice `0` de chaque couple puis, en cas d'égalité, celles à l'indice `1`.

Ainsi, les expressions `(1, 4) < (2, 3)` et `(1, 4) < (1, 6)` s'évaluent toutes les deux à `True`.

La fonction `sorted` de Python prend en argument une liste et renvoie une nouvelle liste contenant les mêmes valeurs triées dans l'ordre croissant à l'aide de l'opérateur <.

7. Donner la liste renvoyée par l'instruction `sorted(etoiles)`.

8. Dessiner l'arbre binaire représenté par le tuple `(((None, (1, 34), None), (2, 35), None), (11, 36), (None, (17, 30), None))`.

L'arbre construit à partir de la liste `etoiles` a donc pour représentation Python :

```
(((None, (10, 30), None), (15, 20), (None, (17, 14), None)),
(29, 21), ((None, (30, 63), None), (35, 13), None))
```

Il est représenté sur la Figure 2 ci-après.

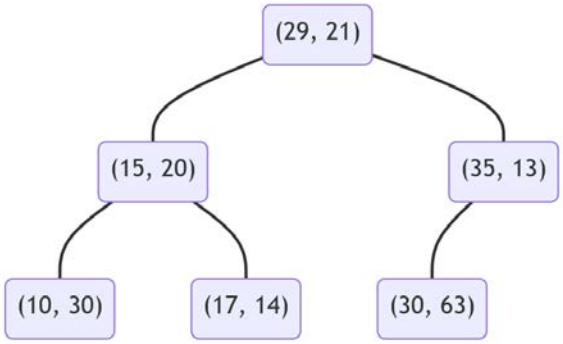

Figure 2. Arbre associé à la liste `etoiles`

9. Dessiner l'arbre binaire de recherche obtenu à partir de la liste :

   `[(1, 33), (2, 30), (2, 33), (4, 30), (8, 39)]`

10. Recopier et compléter les lignes 3, 7, 8, 9 et 11 du code de la fonction `construction` qui prend en paramètres une liste `etoiles` supposée triée par ordre croissant, ainsi que deux entiers `debut` et `fin`. Cette fonction renverra l'arbre binaire de recherche associé aux coordonnées présentes entre les indices `debut` (inclus) et `fin` (exclu) de la liste `etoiles`.

    Par exemple, l'appel initial permettant de construire l'arbre associé à la liste `etoiles` est `construction(etoiles, 0, 6)`.

    L'indice du milieu est 3, le sous-arbre gauche est renvoyé par l'appel `construction(etoiles, 0, 3)` et le sous-arbre droit par `construction(etoiles, 4, 6)`.

```
 1 def construction(etoiles, debut, fin):
 2     if debut == fin:
 3         return ...
 4
 5     milieu = (debut + fin) // 2
 6
 7     sag = construction(...)
 8     racine = ...
 9     sad = ...
10
11     return ...
```

11. Écrire le code de la fonction `en_arbre` qui prend en paramètre une liste `etoiles` de couples de coordonnées non triés et renvoie l'arbre construit selon la démarche décrite plus haut. On pourra utiliser la fonction `construction` de la question précédente.

On souhaite désormais écrire une fonction `contient` qui prend en paramètres un arbre binaire de recherche `arbre` tel que renvoyé par la fonction `construction` ainsi

qu'un tuple d'entiers `position` représentant les coordonnées d'une étoile. Cette fonction renvoie `True` si l'arbre contient cette étoile, `False` dans le cas contraire.

12. Recopier et compléter les lignes 3, 8, 9, 10 et 12 du code de la fonction `contient`.

```
 1 def contient(arbre, position):
 2     if arbre is None:
 3         return ...
 4
 5     sag, valeur, sad = arbre
 6
 7     if position < valeur:
 8         return contient(..., ...)
 9     elif ...:
10         return ...
11     else:
12         return ...
```