I. Interroger une base de données existante (sur ordinateur)

Exercice 1 *Exploiter une base existante (sur machine) : AOP et AOC*

Contexte : Un fichier à copier dans votre répertoire personnel est disponible dans l'espace d'échange des PC :
Enregistrez une copie du fichier `AOP.sqlite` sous un nouveau nom `mesAOP.sqlite` **dans votre répertoire personnel.**

N'éditez pas le fichier source ! ! ! !

Schéma relationnel retenu :

Les clés primaires des tables suivantes sont soulignées.

CI (Code Indication : département suivi du numéro dans le département)
Département (Département)
Commune (Commune)
Art (article L')
Airegéographique (Dénomination de l'AOP ou AOC)
IDA (Identifiant INSEE)

I) Requêtes

Ouvrez cette base de données avec un éditeur SQL.

1. Déterminez les entrées complètes de toutes les AOC de Quimper.
2. Déterminez toutes les communes d'AOC du Finistère.
3. Déterminez toutes les communes corre"spondant à l'IDA 1360. De quoi s'agit-il ?
4. Combien de communes du Finistère possèdent une AOP ?

II. Annales d'écrits de concours (sur papier)

Exercice 2 *Mines MP-PSI-PC 2017*

Base de données

On modélise ici un réseau routier par un ensemble de croisements et de voies reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés. La base de données du réseau routier est constituée des relations suivantes :

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (id) d'un croisement donné.

1. Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.
2. Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c.
3. Que renvoie la requête SQL suivante ?

```
SELECT V2.id_croisement_fin
FROM Voie as V1
JOIN Voie as V2
ON V1.id_croisement_fin = V2.id_croisement_debut
WHERE V1.id_croisement_debut = c
```

Correction.

1.
```
SELECT id_croisement_fin FROM voie
WHERE id_croisement_debut = c;
```

2.
```
SELECT longitude, latitude FROM croisement
JOIN voie ON id = id_croisement_fin
WHERE id_croisement_debut = c;
```

3. Cette requête renvoie les identifiants des croisement atteignables en utilisant deux voies à partir du croisement ayant l'identifiant c (donc avec un croisement intermédiaire).

Exercice 3 *Mines MP-PSI-PC 2016*

Partie I : tri

Dans le but ultérieur de réaliser des études statistiques, on souhaite se doter d'une fonction tri. On se donne la fonction `tri` suivante, écrite en Python :

```
def tri(L):
    n=len(L)
    for i in range(1,n):
        j=i
        x=L[i]
        while 0<j and x<L[j-1]:
            L[j]=L[j-1]
            j=j-1
        L[j]=x
```

1. Lors de l'appel `tri(L)` lorsque L est la liste $[5, 2, 3, 1, 4]$, donner le contenu de la liste L à la fin de chaque itération de la boucle for.
2. Soit L une liste non vide d'entiers ou de flottants. Montrer que « la liste `L[0:i+1]` (avec la convention Python) est triée par ordre croissant à l'issue de l'itération i » est un invariant de boucle. En déduire que `tri(L)` trie la liste L.
3. Évaluer la complexité dans le meilleur des cas et dans le pire des cas de l'appel `tri(L)` en fonction du nombre d'éléments de L. Citer un algorithme de tri plus efficace dans le pire des cas. Quelle est la complexité dans le meilleur et dans le pire des cas ?

   On souhaite, partant d'une liste constituée de couples (chaîne,entier), trier la liste par ordre croissant de l'entier associé suivant le fonctionnement suivant :

```
>>> L=[['Bresil',76],['Kenya',26017],['Ouganda',8431]]
>>> tri_chaine(L)
>>> L
[['Bresil',76],['Ouganda',8431],['Kenya',26017]]
```

4. Écrire en Python une fonction `tri_chaine` réalisant cette opération.

   Pour suivre la propagation des épidémies, de nombreuses données sont recueillies par les institutions internationales comme l'OMS. Par exemple, pour le paludisme, on dispose de deux tables :
   - La table `palu` recense le nombre de nouveaux cas confirmés et le nombre de décès liés au paludisme ; certaines lignes de cette table sont données en exemple (on précise que `iso` est un identifiant unique pour chaque pays) :

nom	iso	annee	cas	deces
Bresil	BR	2009	309 316	85
Bresil	BR	2010	334 667	76
Kenya	KE	2010	898 531	26 017
Mali	ML	2011	307 035	2 128
Ouganda	UG	2010	1 581 160	8 431
...				

   - la table `demographie` recense la population totale de chaque pays ; certaines lignes de cette table sont données en exemple :

pays	periode	pop
BR	2009	193 020 000
BR	2010	194 946 000
KE	2010	40 909 000
ML	2011	14 417 000
UG	2010	33 987 000
...		

5. Au vu des données présentées dans la table `palu`, parmi les attributs `nom`, `iso` et `annee`, quels attributs peuvent servir de clé primaire ? Un couple d'attributs pourrait-il servir de clé primaire ? (on considère qu'une clé primaire peut posséder plusieurs attributs). Si oui, en préciser un.
6. Écrire une requête en langage SQL qui récupère depuis la table `palu` toutes les données de l'année 2010 qui correspondent à des pays où le nombre de décès dus au paludisme est supérieur ou égal à 1 000. On appelle *taux d'incidence d'une épidémie* le rapport du nombre de nouveaux cas pendant une période donnée sur la taille de la population-cible pendant la même période. Il s'exprimer généralement en « nombre de nouveaux cas pour 100 000 personnes par année ». Il s'agit d'un des critères les plus importants pour évaluer la fréquence et la vitesse d'apparition d'une épidémie.
7. Écrire une requête en langage SQL qui détermine le taux d'incidence du paludisme en 2011 pour les différents pays de la table `palu`.
8. Écrire une requête en langage SQL permettant de déterminer le nom du pays ayant eu le deuxième plus grand nombre de nouveaux cas de paludisme en 2010 (on pourra supposer qu'il n'y a pas de pays *ex æquo* pour les nombres de cas). On considère la requête R qui s'écrit dans le langage de l'algèbre relationnelle :

$$R = \pi_{\texttt{nom,deces}}(\sigma_{\texttt{annee}=2010}(\texttt{palu}))$$

   On suppose que le résultat de cette requête a été converti en une liste Python stockée dans la variable `deces2010` et constituée de couples (chaîne,entier).
9. Quelle instruction peut-on écrire en Python pour trier la liste `deces2010` par ordre croissant du nombre de décès dus au paludisme en 2010 ?

Correction. III. Tri et bases de données

Q1. On obtient successivement
$n = 5$
$i = 1$ $x = 2$ $[2, 5, 3, 1, 4]$
$i = 2$ $x = 3$ $[2, 3, 5, 1, 4]$
$i = 3$ $x = 1$ $[1, 2, 3, 5, 4]$
$i = 4$ $x = 4$ $[1, 2, 3, 4, 5]$

Q2. Notons $P(k)$:" à l'issue de l'itération pour la valeur $i = k$, la valeur L_k de la variable L est telle que $L_k[0 : k+1]$ est la liste initiale $L_0[0 : k + 1]$ triée dans l'ordre croissant, et $L_k[k + 1 :] = L_0[k + 1 :]$."
• La liste $L_0[0 : 0 + 1]$ est la liste ayant une seule valeur $L_0[0]$ donc $P(0)$ est vrai (à l'issue de l'itération pour $i = 0$ signifiant avant d'entrer dans l'itération pour $i = 1$ i.e. la première).
• Supposons $P(k)$ vrai et $k + 1 \leqslant n$ (i.e. on fait encore une itération).
Comme j prend initialement la valeur $k + 1$ et ne peut que décroître, et comme on ne modifie que la valeur de $L[j]$, on aura bien $L_{k+1}[k + 2 :] = L_k[k + 2 :] = L_0[k + 2 :]$ (via $P(k)$).
On a aussi $L_k[0 : k + 1] = [x_0, \dots x_k]$ avec $x_0 \leqslant \cdots \leqslant x_k$. Au début de l'itération j vaut $k + 1$ et $x = L_k[k + 1] = L_0[k + 1]$. La boucle "while" permet de faire décroître j et de s'arrêter dès que $L_k[j] \leqslant x$. Quand on s'arrête, on a donc dans $L[0 : k+1]$ la valeur $[x_0, \cdots x_j, x_j, \cdots x_k]$, puis la ligne 9 assure $L_{k+1}[0 : k+1] = [x_0, \cdots x_{j-1}, x, x_j, \cdots x_k]$ i.e. c'est bien $L_k[0 : k] + [x] = L_k[0 : k + 1]$ trié par ordre croissant et via $P(k)$, c'est bien aussi $L_0[0 : k + 1]$ trié par ordre croissant.
• Ainsi $P(k)$ est bien un invariant de boucle.

A la sortie du programme, L est à la valeur L_{n-1} (la dernière valeur prise par i est $n - 1$ donc via $P(n - 1)$, L contient bien la liste initiale triée par ordre croissant.

Q3. La ligne 1 compte 2 opérations élémentaires (une affectation et un appel à la fonction len). *On pourrait compter $n + 1$ si on pense que* len *est plutôt de complexité linéaire en la longueur de la liste passée en paramètre.*
La variable i prend toutes les valeurs de 1 à $n - 1$ et pour une valeur i fixée, on a : - (ligne 4) une opération élémentaire (affectation)
- (ligne 5) deux opérations élémentaires (affectation, appel à un élément d'une liste)
- (lignes 6-7-8) : 4 opérations élémentaires dans le test (2 comparaisons, une opération booléenne et un appel à un élément d'une liste) et pour chaque passage dans la boucle (dans le meilleur des cas : aucun, dans le pire i) il y a 3+2=5 opérations élémentaires. - (ligne 9) une opération élémentaire (affectation)

Finalement, dans le meilleur des cas (liste déjà triée dans l'ordre croissant), on fait de l'ordre de $2 + \sum_{i=1}^{n-1}(1 + 2 + 4 + 0 + 1) = 8n - 5 = O(n)$ opérations élémentaires d'où une complexité linéaire ;
dans le pire des cas (liste triée dans l'ordre décroissant), on fait $2+\sum_{i=1}^{n-1}(1+2+4+5i+1)$ i.e. $8n-5+\frac{5(n-1)n}{2} = O(n^2)$ opérations élémentaires d'où une complexité quadratique.

Dans le pire comme dans le meilleur des cas, le tri fusion est plus efficace car de complexité logarithmique en $O(\ln(n))$.

Q4. À partir de tri, on peut proposer

```
def tri_chaine(tab):
    n=len(tab)
    for i in range(1,n):
        j=1
        x=tab[i]
        while 0<j and x[1]<L[j-1][1]:
            L[j]=L[j-1]
            j=j-1
        L[j]=x
```

Q5. Une clé primaire est un ensemble minimal d'attributs dont la valeur définit de manière unique tout enregistrement.
Ainsi aucune attribut seul n'est une clé primaire pour la table palu : un même pays (nom ou iso) apparaissant dans plusieurs enregistrements (années différentes), pour une même année plusieurs pays apparaissant dans palu,..etc.
En revanche, le couple (annee,nom) (ou (annee,iso)) est une clé primaire.

Q6. On peut écrire

```
SELECT * FROM palu WHERE annee=2010 AND deces >=1000;
```

Q7. On peut écrire (on garde pays pour savoir de quel pays il s'agit même si ce n'est pas explicitement demandé)

```
SELECT pays, 100 000 *cas/pop as taux incidence
    FROM palu JOIN demographie ON pays=iso AND periode=annee
    WHERE annee=2011;
```

Q8. Pour l'année 2010, on cherche le plus grand nombre de nouveaux cas de paludisme, on crée une nouvelle table virtuelle sans ce dernier et on cherche le pays ayant eu le maximum de cas (restants)...

```
SELECT nom FROM palu
    WHERE annee=2010
      AND cas= (SELECT max(cas) FROM palu
       WHERE annee=2010 AND
           cas< (SELECT max(cas) FROM palu WHERE annee=2010));
```

Q9. D'après la requête utilisée, deces201 contient la liste des couples (nom du pays, nombre de décès en 2010). Ainsi, on peut trié par ordre croissant du nombre de décès par

```
tri_chaine(deces2010)
```

Remarquons qu'avant la conversion python, on aurait pu faire directement trier le résultat via la base de données....

Exercice 4 *Exploitation d'une base de données, info communes Centrale 2015*

À partir de mesures régulièrement effectuées par différents observatoires, une base de données des caractéristiques et des états des corps célestes de notre Système solaire est maintenue à jour. L'objectif de cette partie est d'extraire de cette base de données les informations nécessaires à la mise en oeuvre des fonctions développées dans les parties précédentes, puis de les utiliser pour prévoir les positions futures des différentes planètes. Les données à extraire sont les masses des corps étudiés et leurs états (position et vitesse) à l'instant t_{min} du début de la simulation. Une version simplifiée, réduite à deux tables, de la base de données du Système solaire est donnée figure 3. Les masses sont exprimées en kilogrammes, les distances en unités astronomiques (1 au $= 1,5 \times 10^{11}$ m) et les vitesses en kilomètres par seconde. Le référentiel utilisé pour exprimer les composantes des positions et des vitesses est galiléen, orthonormé et son centre est situé à proximité du Soleil.

CORPS

id_corps	nom	masse
⋮	⋮	⋮

ETAT

id_corps	datem	x	y	z	vx	vy	vz
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

La table CORPS répertorie les corps étudiés, elle contient les colonnes
— id_corps (clé primaire) entier identifiant chaque corps ;
— nom, chaine de caractères, désigne le nom usuel du corps ;
— masse de type flottant, contient la masse du corps.

La table `ETAT` rassemble l'historique des états successifs (positions et vitesses) des corps étudiés. Elle est constituée de huit colonnes :
— id_corps de type entier, identifie le corps concerné ;
— datem est la date de la mesure, sous forme d'un entier donnant le nombre de secondes écoulées depuis un instant d'origine ;
— trois colonnes de type flottant pour les composantes de la position x, y, z ;
— trois colonnes de type flottant pour les composantes de la vitesse vx, vy, vz.

A- Écrire une requête `SQL` qui renvoie la liste des masses de tous les corps étudiés.

B- Les états des différents corps ne sont pas forcément tous déterminés exactement au même instant. Nous allons assimiler l'état initial (à la date t_{min}) de chaque corps à son dernier état connu antérieur à t_{min} . Dans toute la suite, on supposera que la valeur de t_{min} , sous le format utilisé dans la table `ETAT`, est accessible à toute requête `SQL` via l'expression `tmin()`.

B.1- On souhaite d'abord vérifier que tous les corps étudiés disposent d'un état connu antérieur à `tmin()`. Le nombre de corps présents dans la base est obtenu grâce à la requête `SELECT count(*) FROM corps`. Écrire une requête `SQL` qui renvoie le nombre de corps qui ont au moins un état connu antérieur à `tmin()`.

B.2- Écrire une requête `SQL` qui renvoie, pour chaque corps, son identifiant et la date de son dernier état antérieur à `tmin()`.

B.3- Le résultat de la requête précédente est stocké dans une nouvelle table date_mesure à deux colonnes :
— id_corps de type entier, contient l'identifiant du corps considéré ;
— date_der de type entier, correspond à la date du dernier état connu du corps considéré, antérieur à `tmin()`.

Pour simplifier la simulation, on décide de négliger l'influence des corps ayant une masse strictement inférieure à une valeur fixée masse_min() et de ne s'intéresser qu'aux corps situés dans un cube, centré sur l'origine du référentiel de référence et d'arête arete() donnée. Les faces de ce cube sont parallèles aux plans formés par les axes du référentiel de référence. Écrire une requête `SQL` qui renvoie la masse et l'état initial (sous la forme masse, x, y, z, vx, vy, vz) de chaque corps retenu pour participer à la simulation. Classez les corps dans l'ordre croissant par rapport à leur distance à l'origine du référentiel.

Correction.

A- `SELECT masse FROM corps`

B-B.1- `SELECT COUNT(DISTINCT id_corps) FROM etat WHERE datem < tmin()`

B.2- `SELECT id_corps,MAX(datem) FROM etat WHERE datem < tmin() GROUP BY id_corps`

B.3-
```
SELECT masse,x,y,z,vx,vy,vz FROM corps AS c
  JOIN etat AS e ON c.id_corps = e.id_corps
  JOIN date_mesure AS d ON (datem = date_der AND e.id_corps = d.id_corps)
  WHERE masse > masse_min() AND ABS(x) < arete()/2
    AND ABS(y) < arete()/2 AND ABS(z) < arete()/2
  ORDER BY x*x+y*y+z*z
```