TP n° 2 d'Informatique
Bases de données et manipulation avec Python

Objectif du TP

Pour un grand nombre d'applications, le stockage des données dans des structures dites « plates » n'est pas suffisant : il entraîne des répétitions, des pertes de places, des temps de recherche plus longs...

Pour obtenir une utilisation de données multidimensionnelles plus efficace, il est nécessaire d'utiliser un nouveau type de stockage : les bases de données. Ce type de stockage est utilisé dans un très grand nombre d'applications, qu'il s'agisse de sites web, de logiciels autonomes sur ordinateur ou sur téléphone, de systèmes embarqués ou pour certains capteurs ayant une mémoire. Dès que le système doit avoir des valeurs complexes ou massives stockées en mémoire, une base de données est utilisée.

Les bases de données sont manipulables de deux façons :
– à l'aide d'un logiciel dédié (ou d'un site web dédié), où l'on exécute les requêtes les unes après les autres, pour en vérifier ou simplement récupérer le résultat ;
– à l'aide d'un langage impératif permettant de récupérer les données et les traiter, pour les afficher dans un site web ou dans un logiciel.
Le but de ce TP est d'apprendre à extraire et manipuler des informations contenues dans des bases de données, dans ces deux cas d'utilisation.

1 Matériel

La base de données contient deux tables. La première table contient des stations météorologiques françaises, la deuxième contient les températures maximales quotidiennes relevées depuis 1893. Ces données sont extraites d'une base de données regroupant les mesures météorologiques récupérées dans plus de 10000 stations européennes, disponible sur le site de l'*European Climate Assessment and Dataset*, `<http //www.ecad.eu>`. Le site permet de récupérer en fichier zippé des collections de mesures, mais aussi d'interroger directement la base (voir le *KNMI Climate Explorer*).

Le schéma relationnel de chaque table est donné dans les fichiers explicatifs joints aux données téléchargées (et a été francisé). Pour la table `stations` :

```
id    : identifiant de station
nom   : nom de station
pays  : code ISO3116 du pays
lat   : latitude en degrés:minutes:secondes (+: Nord, -: Sud)
lon   : longitude en degrés:minutes:secondes (+: Est, -: Ouest)
alt   : altitude en mètres
```

Pour la table `temperatures` :

```
station : identifiant de station
jour    : date de relevé sous la forme AAAAMMJJ
temp    : temperature maximale relevée, en 0.1°C
qual    : code qualité (0='valide'; 1='suspecte'; 9='manquante')
```

La base de données est disponible sous forme d'un fichier `SQLite` appelé `TP2-temperatures.db`.

Le logiciel `SQLiteBrowser` est un logiciel libre qui permet d'ouvrir les fichiers `SQLite` et d'y exécuter des requêtes. Il est téléchargeable à l'adresse `https://sqlitebrowser.org/dl/`. Sous Windows, il peut être installé ou simplement utilisé sans installation en dézippant le fichier zip.

Le logiciel `SQLiteBrowser` pourra permettre de vérifier la cohérence des données.

2 Requêtes indépendantes

Dans un premier temps, on utilise `SQLiteBrowser` pour vérifier que l'on sait concevoir des requêtes classiques. On commence sur papier, puis on tapera ces requêtes dans le logiciel (voir ci-dessous).

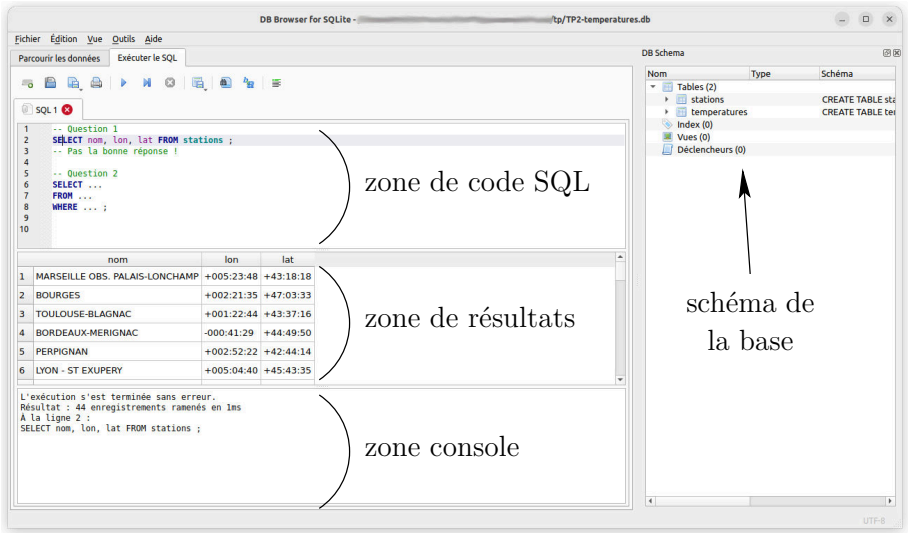
1. Quelle est le nom et la position de la station associée à Paris ?
   On pourra utiliser la condition `nom LIKE "%Paris%"` *pour obtenir les noms contenant « Paris ».*
2. Quelle a été la température à Paris le 1er janvier 2024 ?
   Remarque : la division de deux entiers donne un entier.
3. Quelles étaient les trois stations les plus froides le 1er janvier 1950 ?
4. Quelle est la température moyenne à Nancy en juillet 2024 ?
5. Dans quelles villes la température moyenne de l'été 2024 (juillet et août) est-elle supérieure à 29 °C ?

Sur les ordinateurs du lycée, le logiciel `SQLiteBrowser` est inclus au sein à du logiciel `EduPython` déjà installé. Il peut être ouvert en cliquant sur .

Vous pouvez ouvrir à la fois des fichiers `.db` contenant des bases de données et des fichiers `.sql` contenant des scripts en langage `SQL`. Ces deux types de fichiers sont très différents.

Une fois que vous avez ouvert le logiciel, vous devez cliquer sur « Ouvrir une Base de Données » puis choisir votre fichier `TP2-temperatures.db`. Le schéma de la table apparaît alors dans le premier des quatre onglets dans la fenêtre principale. Le deuxième onglet vous permet de « Parcourir les données », c'est-à-dire les consulter comme des tableaux Excel. Le troisième vous permet d'écrire et d'« Exécuter du SQL » : c'est dans celui-ci que vous travaillerez.

Remarque : les onglets visibles ainsi que la partie droite sont modifiables dans le menu « Vue ». Vous pouvez par exemple ne garder que les deux onglets « Parcourir... » et « Exécuter... », ainsi que le schéma dans la partie droite, comme sur la capture d'écran ci-dessous.



Vous avez intérêt à garder une trace de votre code `SQL`. Ne supprimez pas vos requêtes au fur et à mesure. Vous pouvez enregistrer votre fichier `.sql` (qui n'est donc que du texte) en cliquant sur .

Les commentaires sont précédés de « `--` ». Pratique pour indiquer de quelle question il s'agit.

Les requêtes sont à séparer par « `;` ». Rappel : les indentations et retours à la lignes ne comptent pas pour l'interpréteur, on fait ce qu'on veut donc on en met pour la lisibilité pour le lecteur humain.

Le logiciel lit et exécute toutes les commandes en cliquant sur (peu utile, seul le dernier résultat est affiché). Il n'exécute que la requête où se trouve le curseur en cliquant sur (bien plus utile).

3 Utilisation de Python

Il est aussi possible d'accéder et d'interroger des bases de données avec Python, ce qui permet d'en réaliser un affichage, un traitement. Pour ce faire, il faut utiliser un module spécifique, `sqlite3`. La syntaxe générale est la suivante :

```
# Importation du module nécessaire
import sqlite3

# Connexion à la base de données
connexion = sqlite3.connect("fichier.db")                    # ouverture de la base
connexion.row_factory = sqlite3.Row              # accès aux colonnes par leur nom
curseur = connexion.cursor()                          # obtention d'un « curseur »

# Exécution d'une requête sur la base
# Tout reste "dans" le curseur : pas d'affectation ici
curseur.execute("SELECT * FROM table")
# Remarque : si on veut conserver le formatage de la requête SQL, on peut
# ajouter des retours à la ligne en mettant un \ en bout de ligne
curseur.execute("\
SELECT * \
FROM table")

# S'il n'y a qu'un seul résultat à récupérer, on peut utiliser fetchone()
ligne = curseur.fetchone()

# Dans tous les cas et surtout si on ne sait pas trop le nombre de résultats,
# on peut utiliser le caractère itérable du curseur
for ligne in curseur:
    ...instructions...

# Dans les deux cas, ligne est un dictionnaire dont les clés sont les colonnes
# de la table produite :
curseur.execute("SELECT nom, lon, lat FROM stations")
for ligne in curseur:
    print( ligne['nom'] + ' : ' + ligne['lon'] + ligne['lat'] )

# Fermeture obligatoire de la connexion, permet de libérer le fichier
connexion.close()
```

6. Écrire une fonction `affichage` qui reçoit un dictionnaire `ligne` contenant les clés `nom`, `jour` et `temp`, qui affiche les résultats sous un formatage particulier, permettant par exemple d'obtenir pour les questions 2 et 3 :

```
PARIS - ORLY                              - 01/07/2024 - 22.1°C
STRASBOURG-ENTZHEIM                       - 01/01/1950 - 0.6°C
BASEL-MULHOUSE                            - 01/01/1950 - 0.6°C
CLERMONT-FERRAND                          - 01/01/1950 - 0.9°C
```

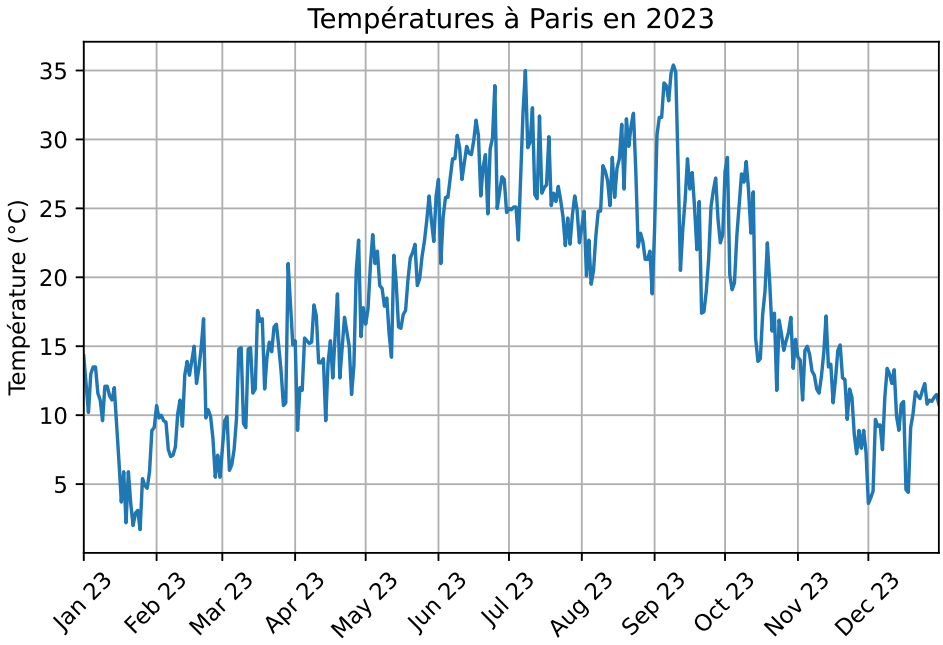
On pourra utiliser la méthode de chaîne de caractères `chaine.ljust(n)` qui *justifie à gauche* la `chaine`, c'est-à-dire la complète d'autant de blancs qu'il le faut pour que le résultat ait `n` caractères (40 sur l'exemple). Pour la date, il s'agit d'une chaîne de caractères de la forme « `AAAAMMJJ` », pour laquelle la syntaxe d'extraction des listes est utilisable (`ligne['jour'][4:6]` permet de récupérer le mois). On notera que la température est stockée comme un entier, égal à 10 fois la température.

4 Évolution de la température sur une année

7. Récupérer dans une liste les températures de 2023 à Paris, en affichant uniquement celles du premier jour de chaque mois.
8. Afficher ces températures sur un graphe. Soigner l'affichage : titre de graphe, titres des axes... On peut dans un premier temps tracer l'évolution de la température sans préciser d'abscisses (avec `plt.plot(temperatures)`).

On souhaite en particulier afficher les dates en abscisse sous la forme « janv. 2023 ». Pour cela, on doit
- charger la classe `datetime` dans le module `datetime`, qui permet de transformer les chaînes de caractères `YYYYMMJJ` en objet de type « date » :
  ```
  from datetime import datetime
  ```
- charger les fonctions `MonthLocator` et `DateFormatter` du module `matplotlib.dates`, qui permettent de choisir les mois à afficher (tous) et de choisir le format de l'affichage :
  ```
  from matplotlib.dates import DateFormatter, MonthLocator
  ```
- transformer chaque date (`'%Y%m%d'` correspond au format `YYYYMMJJ`) :
  ```
  datetime.strptime(ligne['date'],'%Y%m%d')
  ```
- utiliser ces dates comme abscisses des points :
  ```
  plt.plot(dates,temperatures)
  ```
- limiter l'axe horizontal, pour éviter à `matplotlib` de prendre une marge à gauche et à droite :
  ```
  plt.xlim(dates[0],dates[-1])
  ```
- choisir d'afficher une étiquette sur chaque mois sur l'axe :
  ```
  plt.gcf().axes[0].xaxis.set_major_locator(MonthLocator())
  ```
- modifier l'affichage des étiquettes en abscisse (`'%b %y'` correspond au format « janv. 2023 ») :
  ```
  plt.gcf().axes[0].xaxis.set_major_formatter(DateFormatter('%b %y'))
  ```
- incliner les dates de 30° pour mieux les lire :
  ```
  plt.xticks(rotation=45)
  ```

Une fois ceci réalisé, le graphe demandé en question 8 doit ressembler à celui-ci :



9. Modifier le graphe de la question 8 en écrivant le code pour obtenir le graphe ci-dessus.

5 Filtrage

On observe sur le graphique précédent de grandes variations rapides d'un jour à l'autre. On souhaite « lisser » ces courbes. Une méthode classique est de réaliser un filtrage appelé « moyenneur » qui réalise une moyenne glissante sur, par exemple, 10 jours successifs. On parle aussi de filtre « passe-bas » car il supprime les variations rapides et conserve les variations lente. En appelant x_i les valeurs brutes données en entrée du filtre et y_i les valeurs filtrées récupérées en sortie, on a alors

$$y_i = \frac{1}{10}\left(x_{i-5} + x_{i-4} + x_{i-3} + \cdots + x_{i+3} + x_{i+4}\right)$$

10. Définir la fonction `filtre(liste)`, qui prend une `liste` de n valeurs en argument et renvoie une liste de $n-9$ valeurs (le balayage commence à $i=5$ et finit à $i =$ `len(liste)`-4).
Généraliser cette fonction à l'aide d'un deuxième paramètre `p`, de valeur par défaut égale à 10, correspondant au nombre (toujours supposé pair) de valeurs utilisées pour le filtrage.
11. Afficher sur un même graphe les températures brutes et filtrées à Paris en 2023. Pour améliorer la lisibilité, on pourra simplement pointer les températures brutes (avec `plot(dates,temperatures,'+')`, les points ne sont alors plus liés entre eux).

6 Évolution globale

12. Récupérer la liste des températures moyennes annuelles parisiennes depuis le début des relevés : il faut, en langage `SQL`, faire la moyenne des températures pour chaque année. On peut obtenir les 4 premiers caractères d'une chaîne de caractères par l'expression `SUBSTR(champ,1,4)`.
Tracer l'évolution de la température moyenne annuelle brute et de la courbe filtrée. Comme à la question 11, on pourra tracer simplement des points pour les valeurs brutes et une vraie courbe pour les valeurs filtrées.