Travaux pratiques

informatique commune

Bases de données

Rappelons qu'une *base de données relationnelle* (BDR) est un ensemble de *tables* contenant des données reliées entre elles par des *relations* ; on y extrait de l'information par le biais de *requêtes* exprimées dans un langage appelé SQL (*Structured Query Language*).

Les architectures les plus répandues des systèmes de gestion des bases de données (SGBDR) comportent deux ou trois niveaux. Dans une architecture client/serveur (architecture à *deux tiers*), la base de données est gérée sur un serveur et l'application tourne sur une autre machine, le client. Dans une architecture à trois niveaux (architecture *trois tiers*) s'intercale entre le client et le serveur une machine qui gère l'interface (typiquement un navigateur web). Ces SGBDR permettent une gestion fine des accès concurrents (ils garantissent des transactions cohérentes même lorsque plusieurs clients donnent des ordres de lecture ou d'écriture sur les mêmes données) [1].
Ainsi, dans mon cours et mes travaux pratiques de première année j'utilise un navigateur web pour interroger à distance une base de données géographique. N'hésitez pas à vous y référer pour vous entraîner à la rédaction de requêtes SQL.

Mais toutes les SGBDR ne fonctionnent pas sur le modèle client/serveur. SQLite est une bibliothèque écrite en C qui propose un moteur de BDR ne reproduisant pas ce schéma : ici l'intégralité de la base de données est stockée dans un fichier et le système de gestion directement intégré aux programmes l'utilisant. Ce modèle est très populaire sur les systèmes embarqués, en particulier sur les smartphones ; iOS et Android utilisent tous deux SQLite dans de nombreuses applications.

SQLite et Python

Pour utiliser SQLite en Python sur votre ordinateur, vous devez disposer du module `sqlite3` et d'une base de données dans votre répertoire de travail. Commencez donc par récupérer le fichier `dots.sqlite` [2] contenant cette BDR à l'adresse suivante : `http://info-llg.fr/commun-mp/dots.sqlite` ou directement à partir de ma clé USB (environ 19 Mio).
Placez ce fichier dans le répertoire courant puis écrire en Python les lignes suivantes :

```
import sqlite3
bdr = sqlite3.connect('./dots.sqlite')  # connexion à la base de donnée
cur = bdr.cursor()
```

L'objet créé à la troisième ligne est un *curseur* ; c'est par l'intermédiaire de deux méthodes appliquées à cet objet que nous allons interagir avec la base de données : `execute` et `fetchall`.

- `cur.execute("SELECT ... FROM ...")` exécute une requête SQL passée en paramètre sous la forme d'une chaîne de caractères ;
- `result = cur.fetchall()` récupère les résultats de la requête sous la forme d'une liste de tuples.

1. Interrogations usuelles de la base de données

Nous allons pour l'instant travailler avec deux tables : `depts` qui décrit les départements de France métropolitaine et `communes` qui comme son nom l'indique en décrit les communes. Les attributs de ces deux tables sont :

- Pour la table `depts` :
  - `id` (un entier) la clé primaire de l'enregistrement ;
  - `code` (une chaîne de caractères) le code administratif du département ;
  - `nom` (une chaîne de caractères) le nom du département ;
  - `chef_lieu` (un entier) l'identifiant du chef-lieu du département.

1. Cependant, la question de la concurrence d'accès n'est pas abordée par le programme.
2. Un grand merci à Émeric Tourniaire qui en est le créateur.

– Pour la table `communes` :
  – `id` (un entier) la clé primaire de l'enregistrement ;
  – `code` (une chaîne de caractères) le code administratif de la commune ;
  – `postal` (une chaîne de caractères) le code postal de la commune ;
  – `nom` (une chaîne de caractères) le nom de la commune ;
  – `superficie` (un flottant) la superficie de la commune, en hectares [3] ;
  – `altitude` (un entier) l'altitude de la commune, en mètres ;
  – `population` (un flottant) la population d'une commune, en millier d'habitants ;
  – `dep_id` (un entier) l'identifiant du département où se trouve la commune.

Question 1. Rédiger des requêtes SQL pour répondre aux questions suivantes [4] :

a) Donner la liste des dix communes les plus vastes de France, classées par ordre décroissant de taille, en précisant pour chacune d'elles le nom du département où elles se situent.

b) Quelle est la commune de plus de 10 000 habitants la plus haute de France ? Dans quel département se situe-t-elle ?

c) Quel est le chef-lieu de département le moins peuplé de France ?

d) Quelle est la superficie moyenne (en hectares) des communes corses ? Rappelons que la Corse est constituée de deux départements de codes administratifs 2A et 2B.

e) En considérant que la superficie d'un département est égal à la somme des superficies des communes qu'il abrite, calculer la superficie moyenne (en km^2) des départements métropolitains.

f) Quel est le département le plus vaste ?

g) Parmi les communes de plus de 1 000 habitants, quel est celle dont la densité est la plus faible ? (c'est-à-dire dont le nombre d'habitants par mètre carré est le plus petit). Dans quel département se trouve-t-elle ? On exprimera sa densité en nombre d'habitants par kilomètre carré.

h) Existe-t-il dans le Pas-de-Calais (code administratif 62) deux communes ayant même superficie et situées à la même altitude ?

i) Enfin, déterminer le code postal attribué au nombre le plus élevé de communes différentes (on en précisera ce nombre).

2. Interaction avec la base de données

L'intérêt majeur de SQLite est de permettre une interaction très simple entre un langage de programmation (ici Python) est une base de données.

Question 2.

a) Rédiger une fonction `chef_lieu` qui prend en argument le code administratif d'un département et renvoie le nom de celui-ci ainsi que de son chef-lieu.

b) Rédiger une fonction `prefecture` qui prend en argument le nom d'une commune et retourne la liste des chef-lieux des départements où se situe une commune portant ce nom.

c) Combien de réponses renvoie cette fonction pour la commune de Sainte-Colombe ?

Question 3.

a) A l'instar des annuaires inversés, rédiger en Python une fonction `code_postal_inverse` qui prend en argument un code postal et qui retourne un couple formé du nom du département et de la liste des communes correspondant à ce code postal. S'il n'en existe pas, cette fonction retournera la valeur `None`.

b) Utiliser votre fonction pour déterminer les communes qui se partagent le code postal `42440`.

c) Votre fonction retourne-t-elle un résultat pour le code postal `08320` ? Si ce n'est pas le cas, corriger votre fonction pour qu'elle renvoie le nom des quatre communes qui se partagent ce code.

d) À la ville de Nantes est associée le code postal `44000`. Votre fonction marche-t-elle pour ce dernier ? Si ce n'est pas le cas, interroger la base de données pour visualiser l'enregistrement associé à cette ville et comprendre la raison de cet échec. Modifier alors votre fonction pour prendre en compte ce type de situation. On pourra utiliser la fonction **LIKE** : si s est une chaîne de caractères, alors `s` **LIKE** `'%string%'` retourne `True` si et seulement si la chaîne de caractère `'string'` est un facteur de s (le caractère `%` désigne une chaîne quelconque).

3. On rappelle qu'un hectare est égal à 10 000 mètres carrés.
4. Chacune de ces questions peut trouver sa réponse à l'aide d'une seule requête SQL.

Question 4. Représentation graphique
La fonction `pie` de MATPLOTLIB.PYPLOT permet le tracé d'un diagramme circulaire (ou « camembert », un disque représentant un petit nombre de valeurs par des angles proportionnels à celles-ci). Consulter l'aide associée à cette fonction, puis tracer un diagramme circulaire dans lequel seront représentés :

- le nombre de communes situées à une altitude inférieure à 50 m ;
- le nombre de communes situées entre 50 et 100 m d'altitude ;
- le nombre de communes situées entre 100 et 500 m d'altitude ;
- le nombre de communes situées entre 500 et 1000 m d'altitude ;
- le nombre de communes situées à plus de 1000 m d'altitude.

3. Tracé de contour

Une troisième table de la base, nommée `pointsdep`, décrit les contours des différents départements. Chacun d'eux est représenté par un ou plusieurs polygones décrits par les coordonnées de leurs sommets. Les attributs de cette table sont :

- `id` (un entier) la clé primaire de l'enregistrement ;
- `iddept` (un entier) l'identifiant du département ;
- `poly` (un entier) le numéro du polygone ;
- `ordre` (un entier) le numéro du point ;
- `x` et `y` (deux flottants) la longitude et la latitude du point.

En effet, tous les départements ne sont pas connexes (en particulier les départements maritimes qui possèdent des îles), et ceux-ci sont décrits par autant de polygones qu'il y a de composantes connexes. Chaque sommet de chaque polygone est ainsi décrit par un enregistrement de la table.

Question 5.

a) Rédiger une fonction `polygone` qui prend en arguments le code administratif d'un département et le numéro d'un polygone et qui retourne les listes `X` et `Y` des abscisses et ordonnées des sommets de ce polygone.

b) En déduire une fonction `dessine_departement` qui prend en argument le code administratif d'un département et trace en retour les contours de ce dernier.

Remarque. Si vous utilisez la fonction `plot` de MATPLOTLIB.PYPLOT vous obtiendrez le contour du département ; si vous la remplacez par la fonction `fill` (avec la même syntaxe) l'intérieur du polygone sera coloré. Dans ce cas, ajoutez un argument à votre fonction pour vous permettre de choisir la couleur à utiliser.

c) Illustrer l'utilisation de cette fonction en dessinant les départements bretons (il s'agit des départements de codes administratifs 22, 29, 35, 56).
On utilisera la commande `plt.axes().ser_aspect(1.5)` pour obtenir un aspect plus conforme à nos habitudes.

Question 6. Enfin, dessinez une carte de France sur laquelle la couleur de chaque département est proportionnelle à sa population.