Gestion des données
TP1 : SQL en Python

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

2021-2022

L'objectif de ce TD est d'accéder à une base SQL à partir d'un programme Python, de façon à extraire des données et à effectuer divers traitements dessus.

Pour des raisons de simplicité, on utilise désormais SQLite, un système de base de données relationnelles qui a la particularité de ne pas s'utiliser avec un mécanisme client-serveur, mais directement comme une bibliothèque au sein d'un programme.

Les données utilisées ici proviennent du portail de données publiques de Météo-France, en particulier du jeu de données *Données SYNOP essentielles OMM* (https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=90&id_rubrique=32) qui regroupent des observations des principales stations météorologiques françaises. Une partie de ces observations (les stations, les dates et les températures) a été insérée dans une base de données SQLite.

La documentation du module sqlite3 est disponible à l'adresse https://docs.python.org/3/library/sqlite3.html pour Python3.

Remarque : il n'y a pas de type `date` en SQLite, on utilise donc un simple type `TEXT` pour stocker des dates. Il est possible d'interpréter une colonne contenant des dates comme de vraies dates avec les fonction `date` et `datetime` : c'est utile en particulier pour les comparaisons ou les tris.

Exercice 1 *Prise en main*

Téléchargez le fichier http://www-connex.lip6.fr/~schwander/enseignement/m2stat_gd/temperature.db et ouvrez-le avec l'interpréteur interactif SQLite :

```
sqlite3 temperature.db
```

L'affichage devrait ressembler à :

```
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
sqlite>
```

Question 1

Affichez la structure de la base à l'aide de la commande `.schema` (sans point-virgule à la fin, c'est une commande est spécifique à SQLite).

Correction

```
SQLite version 3.16.2 2017-01-06 16:32:41
Enter ".help" for usage hints.
sqlite> .schema
CREATE TABLE stations (
    id INT,
    name TEXT,
    PRIMARY KEY (id)
```

```
);
CREATE TABLE temperatures (
    id INT,
    date TEXT,
    station INT,
    temp REAL,
    PRIMARY KEY (id)
);
```

Question 2
Affichez tout le contenu de la base à l'aide de la commande `.dump` (sans point-virgule à la fin, c'est une commande est spécifique à SQLite).

Correction

```
SQLite version 3.16.2 2017-01-06 16:32:41
Enter ".help" for usage hints.
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE stations (
    id INT,
    name TEXT,
    PRIMARY KEY (id)
);
INSERT INTO "stations" VALUES(7005,'ABBEVILLE');
INSERT INTO "stations" VALUES(7015,'LILLE-LESQUIN');
INSERT INTO "stations" VALUES(7020,'PTE DE LA HAGUE');
INSERT INTO "stations" VALUES(7027,'CAEN-CARPIQUET');
INSERT INTO "stations" VALUES(7037,'ROUEN-BOOS');
INSERT INTO "stations" VALUES(7072,'REIMS-PRUNAY');
INSERT INTO "stations" VALUES(7110,'BREST-GUIPAVAS');
INSERT INTO "stations" VALUES(7117,'PLOUMANAC''H');
INSERT INTO "stations" VALUES(7130,'RENNES-ST JACQUES');
INSERT INTO "stations" VALUES(7139,'ALENCON');
INSERT INTO "stations" VALUES(7149,'ORLY');
INSERT INTO "stations" VALUES(7168,'TROYES-BARBEREY');
INSERT INTO "stations" VALUES(7181,'NANCY-OCHEY');
...
```

Question 3
Décrivez la structure de la base, sous forme de texte puis de schéma entité-association.

Correction

La base est constituée de deux tables :

- `STATIONS(_id_, name)`
- `TEMPERATURES(_id_, date, #station, temp)`

La table `STATIONS` contient une liste de stations, décrites seulement par leur nom. La table `TEMPERATURE` contient des mesures de températures (colonne `temp` en Kelvin) avec des dates (colonne `date` au format ISO 8601) et des stations (clé étrangère `station` pointant vers la table `STATIONS`).

Question 4
Affichez toutes les températures relevées par la station `ST-PIERRE`.

Correction

On réalise une jointure entre la table `stations` et la table `temperatures` et on filtre sur la colonne `name` pour n'avoir que les mesures de la stations `ST-PIERRE`.

```
SELECT temperatures.date, temperatures.temp FROM stations,temperatures
   WHERE stations.id = temperatures.station AND stations.name = 'ST-PIERRE';
```

Question 5

Affichez toutes les températures relevées par la station `ST-PIERRE`. en janvier 2016.

Correction

On rajoute un filtre sur les date.

```
SELECT temperatures.date, temperatures.temp FROM stations,temperatures
   WHERE stations.id = temperatures.station AND stations.name = 'ST-PIERRE'
     AND temperatures.date >= "2016-01-00"
     AND temperatures.date < "2016-02-01";
```

Ici, la comparaison des chaînes de caractères suffit. Pour des conditions plus compliquées, on pourrait utiliser les fonctions de manipulations de dates fournies par sqlite (par exemple, `date`, `datetime` ou `strftime`).

Question 6

Affichez toutes les températures relevées en France métropolitaine.

Correction

On peut utiliser un filtre pour ne sélectionner que les stations en France métropolitaine, mais comme il y en a beaucoup plus que de stations hors France métropolitaine il est plus simple d'exclure les stations qui ne sont pas utiles.

```
SELECT stations.name,temperatures.date, temperatures.temp FROM stations,temperatures
   WHERE stations.id = temperatures.station
      AND stations.name != 'GLORIEUSES'
      AND stations.name != 'JUAN DE NOVA'
      AND stations.name != 'EUROPA'
      AND stations.name != 'TROMELIN'
      AND stations.name != 'GILLOT-AEROPORT'
      AND stations.name != 'NOUVELLE AMSTERDAM'
      AND stations.name != 'CROZET'
      AND stations.name != 'KERGUELEN'
      AND stations.name != 'PAMANDZI'
      AND stations.name != 'ST-PIERRE'
      AND stations.name != 'LA DESIRADE METEO'
      AND stations.name != 'ST-BARTHELEMY METEO'
      AND stations.name != 'LE RAIZET AERO'
      AND stations.name != 'TRINITE-CARAVEL'
      AND stations.name != 'LAMENTIN-AERO'
      AND stations.name != 'SAINT LAURENT'
      AND stations.name != 'CAYENNE-MATOURY'
      AND stations.name != 'SAINT GEORGES'
      AND stations.name != 'MARIPASOULA'
      AND stations.name != 'DUMONT D''URVILLE';
```

Cette question montre qu'il est extrêmement fastidieux de réaliser ce genre de requête directement en SQL. Il serait beaucoup pratique de pouvoir générer les conditions dans le `WHERE` avec un programme, et

de pouvoir utiliser ce programme pour déterminer où est une station d'après ces coordonnées.

\paragraph{Remarque} Attention aux doubles guillements `''` pour la station ='DUMONT D"URVILLE'.

Exercice 2 *Avec Python*

On doit commencer par charger la base de données (ceci ne sera pas rappelé dans les réponses aux questions).

```
import sqlite3
db = sqlite3.connect("temperature.db")
```

Question 1

Tracez toutes les températures relevées par la station `ST-PIERRE`.

```
query = """SELECT temperatures.date, temperatures.temp FROM stations,temperatures
           WHERE stations.id = temperatures.station AND stations.name = 'ST-PIERRE';"""
data = db.execute(query)
```

Attention, `data` n'est pas une vraie liste :

```
print(type(data))
```

On peut quand même le parcourir avec une boucle `for` :

```
for line in data:
    print(line)
```

Ou le convertir en une vraie liste :

```
data = list(data)
print(type(data))
```

Pour l'affichage, on peut se servir de ce qui précède pour appeler directement les fonctions de `matplotlib`. Pour une utilisation statistique, il peut être utile de connaître la fonction `pandas.read_sql_query` qui construit un dataframe à partir d'une requête sql :

```
import pandas as pd
df = pd.read_sql_query(query, db)
print(df.describe())
```

Question 2

Tracez toutes les températures sur un mois il y a un an à la station `ST-PIERRE`.

Remarque : les dates doivent êtres calculées avec Python, et non pas écrites directement.

Correction

```
import datetime
start = datetime.datetime.now() - datetime.timedelta(days=365)
end   = start + datetime.timedelta(days=30)

query = """SELECT temperatures.date, temperatures.temp FROM stations,temperatures
           WHERE stations.id = temperatures.station AND stations.name = 'ST-PIERRE'
               AND temperatures.date > '{}' AND temperatures.date < '{}';""".format(start, end)
```

\paragraph{Remarque} Notez les `{}` dans la chaîne de caractères : ils seront remplacés par la fonction `format` avec les valeurs données en argument.

En fait, on aurait pu faire les calculs de date directement avec sqlite3 en utilisant les fonctions décrites à la page https://www.sqlite.org/lang_datefunc.html

Question 3
Tracez toutes les températures sur un mois il y a un an, en France métropolitaine.

Correction

Il suffit d'utiliser la requête de la question **??**.

Question 4
Calculez la moyenne annuelle de chaque station, en utilisant Python pour le calcul de la moyenne.

Correction

```
def mean_python():
    query = """SELECT temperatures.date, stations.name, temperatures.temp FROM stations,temperatures
               WHERE stations.id = temperatures.station
                   AND temperatures.date >= date('2016-01-01') AND temperatures.date < date('2017-01
    df = pd.read_sql_query(query, db)
    import numpy as np
    df["temp"] = df["temp"].replace('mq', np.nan)
    # df["temp"].astype(float)
    df2 = df.groupby(["name"])["temp"].mean()
    print(df2)
mean_python()
```

Question 5
Calculez la moyenne annuelle de chaque station, en utilisant SQL pour le calcul de la moyenne.

Correction

```
def mean_sql():
    query = """SELECT stations.name, avg(temperatures.temp) FROM stations,temperatures
               WHERE stations.id = temperatures.station
                   AND temperatures.date >= date('2016-01-01') AND temperatures.date < date('2017-01
               GROUP BY stations.name;"""
    df = pd.read_sql_query(query, db)
    print(df)
mean_sql()
```

Question 6
Comparez les temps d'exécutions pour les deux questions précédentes.

Correction

Dans `ipython`, on peut utiliser `%timeit` :

```
%timeit mean_python()
[...]
1 loop, best of 3: 286 ms per loop

%timeit mean_sql()
[...]
10 loops, best of 3: 132 ms per loop
```

Le calcul coté SQL est beaucoup plus rapide qu'en Python (même en utilisant Pandas, ce serait encore pire avec du Python pure) : on aura donc toujours intérêt à faire le plus de calcul coté SQL que du coté Python (ou n'importe quel autre langage accédant à la base de données).

Question 7
Tracez toutes les températures sur un mois il y a un an, en France métropolitaine, en déterminant les stations à partir des informations sur les stations données à l'adresse https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=90&id_rubrique=32.

Correction

Question 8
Modifiez la structure de la base pour insérer les coordonnées des stations, puis insérez les coordonnées.

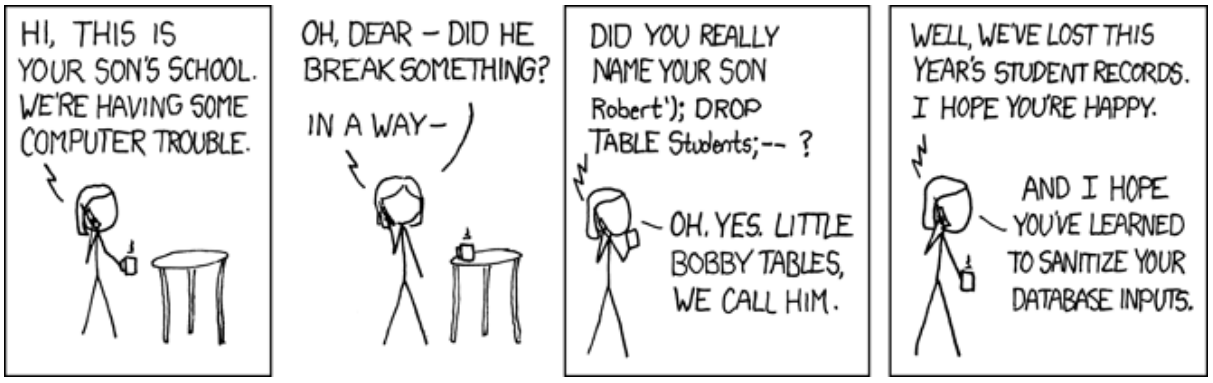
Correction

Exercice 3 *Insertion et sécurité*

Question 1
Écrivez une fonction Python qui insère une nouvelle observation dans la base, en prenant les valeurs à insérer comme arguments de la fonction.

Correction

Question 2
Lisez la bande dessinée suivante :



Source : XKCD 327 https://xkcd.com/327/, CC BY-NC 2.5

On pourra aussi consulter la page suivante : https://www.explainxkcd.com/wiki/index.php/Little_Bobby_Tables.

Correction

Question 3
À l'aide d'arguments bien choisis pour la fonction écrite à la question **??**, supprimez ou modifiez des données dans la base.

Correction

Question 4
Imaginez ce qui peut se passer si cette fonction est utilisée par un utilisateur malveillant (par exemple, à partir d'un site web).

REMARQUE : on appelle ce genre d'attaque des *injections SQL*. C'est un problème très grave auquel on doit faire face lorsqu'une requête SQL est construite à partir d'une entrée extérieure.

La meilleure stratégie pour se protéger des injections consiste à utiliser des requêtes préparées : au lieu de construire directement la requête en concaténant des chaînes de caractères, on utilise un mécanisme spécifique pour passer des arguments aux requêtes, appelé *requête préparée*.

Ainsi, plutôt que :

```
conn.execute("SELECT * FROM toto WHERE X = " + str(X))
```

on utilisera :

```
conn.execute("SELECT * FROM toto WHERE X = ?", X)
```

REMARQUE : il n'est pas toujours possible d'utiliser des requêtes préparées, dans ce cas on peut utiliser un mécanisme d'échappement qui permet de transformer une chaîne de caractère en garantissant qu'elle en contient plus aucun code SQL risquant d'être interprété. C'est en général plus risqué et plus compliqué à utiliser, on préférera donc les requêtes préparées quand c'est possible.