Bases de données (1/3) : écosystème et découverte de SQL

Les 17 et 23 avril, et le 14 mai 2014
http://www.mp933.fr/ - stephane@gonnord.org
fayard.prof@gmail.com

Buts du TP

- Apprendre a accéder à une base de données, via une application graphique et via la bibliothèque `sqlite3`.
- Faire quelques requêtes élémentaires, à nouveau via une application graphique et depuis un scrip python.
- Commencer à s'habituer à la syntaxe SQL.
- Apprendre à créer/alimenter une base.

Exercice 1 Créer (au bon endroit) un dossier associé à ce TP. Y placer les bases de données généreusement fournies ainsi que le fichier `csv` :

`base_triangles.db` `communes-regions-departements.db` `cafes.csv`

1 Clic-clic-clic vs. tip-tip-tip : des triangles

Où on apprend à consulter une base de données...

Exercice 2 Lancer `sqliteman` (qui se trouve sous scribe dans le répertoire commun `T:`) ; charger la base de données `base_triangles.db`, consitituée d'une seule table, dont le schéma relationnel est :

```
triangle( idt:integer, ab:integer, ac:integer, bc:integer)
```

Chaque enregistrement/ligne représente les longueurs d'un triangle ABC, ainsi qu'un identificateur unique (on parle de *clé primaire*).

1. Lancer (successivement !) les requêtes SQL suivantes, en réfléchissant à leur signification/objectif :

```
SELECT COUNT(*) FROM triangles
SELECT * FROM triangles WHERE ab+ac+bc=100
SELECT ab*ac*bc FROM triangles WHERE ab+ac+bc>=100
```

2. Déterminer, à l'aide de requêtes SQL :
   (a) la plus petite valeur des produits $AB.AC.BC$, pour les triangles (ABC) de périmètre supérieur ou égal à 100 ;
   (b) les longueurs correspondants au(x) triangle(s) pour le(s)quel(s) le minimum précédent est atteint ;
   (c) tous les triangles rectangles en A ;
   (d) le nombre de tels triangles ;
   (e) le maximum des périmètres des triangles rectangles en A ;
   (f) tous les triangles équilatéraux.
3. Optionnel : déterminer les triangles tels que $\frac{AB+AC+BC}{3} = 42$, et placer les longueurs de ces triangles dans un fichier [1] (texte ou csv).

Exercice 3 Lancer Spyder, sauvegarder immédiatement le fichier édité au bon endroit. Écrire une commande absurde, de type `print(5*3)` dans l'éditeur, sauvegarder et exécuter après un petit coup de F6.

1. Il y a un bouton pour ça : *Export Data*, juste dessus les résultats.

1. Taper les lignes suivantes dans le script python. Exécuter.

```
import sqlite3

base = sqlite3.connect('base_triangles.db')
curseur = base.cursor()

res = curseur.execute("""SELECT * FROM triangles WHERE ab+bc+ac=100""")
foo = res.fetchall()

base.close()
```

2. Que vaut maintenant `res` ? Et `foo` ?
3. Déterminer, à l'aide d'une requête SQL, l'ensemble des triangles plats.
4. Optionnel : depuis Python, déterminer les triangles tels que $\frac{AB+AC+BC}{3} = 42$, et placer les longueurs de ces triangles dans un fichier. Comparer avec la méthode directe depuis `sqliteman`

2 Communes, départements et régions

Où on découvre des jonctions de tables.

EXERCICE 4 Lancer `sqliteman` ; charger la base de données `communes-regions-departements.db`, consituée de trois tables appelées `communes`, `departements` et `regions`.

1. Écrire *sur papier* les schémas relationnels de ces tables.
2. Lancer la requête :

```
SELECT C.nom, D.nom FROM
communes C JOIN departements D
ON C.dep = D.id
```

   Cette requête nous a permis de joindre les deux tables `communes` et `departements`. Au passage, on a utilisé des alias (renommage). On aurait également pû écrire :

```
SELECT communes.nom, departements.nom FROM
communes JOIN departements
ON communes.dep = departements.id
```

3. En s'inspirant du modèle précédent, donner la liste de toutes les communes avec pour chacune, son département, sa région et sa population.
   La première ligne est :

```
L'Abergement-Clémenciat,Ain,Rhône-Alpes,784
```

4. (a) Donner la liste des villes de plus de 100000 habitants, ainsi que leur population et leur région.
   (b) Trier la liste précédente par ordre décroissant de la population. On pourra pour cela faire une recherche Google pour trouver la bonne fonction `SQL` à utiliser.
5. Donner la liste des communes (nom et population) dont le nom commence par `Pa` et se finissant par `is`. On pourra pour cela utiliser la commande `LIKE` (Google est encore votre ami).
6. Déterminer les communes qui ont strictement plus de lettres dans leur nom que leur nombre d'habitants.

NB : une jonction peut également être réalisée via la syntaxe suivante :

```
SELECT C.nom, D.nom FROM communes C, departements D WHERE C.dep = D.id
```

3 Création/alimentation d'une base de données

Où on apprend à créer et alimenter une base de données.

Exercice 5 Toujours dans `sqliteman`, créez un nouvelle base de données dans un fichier nommé `cinema.db`.

1. Créez une table dont le schéma relationnel est :

   `films( nom:text, realisateur:text, annee:text)`

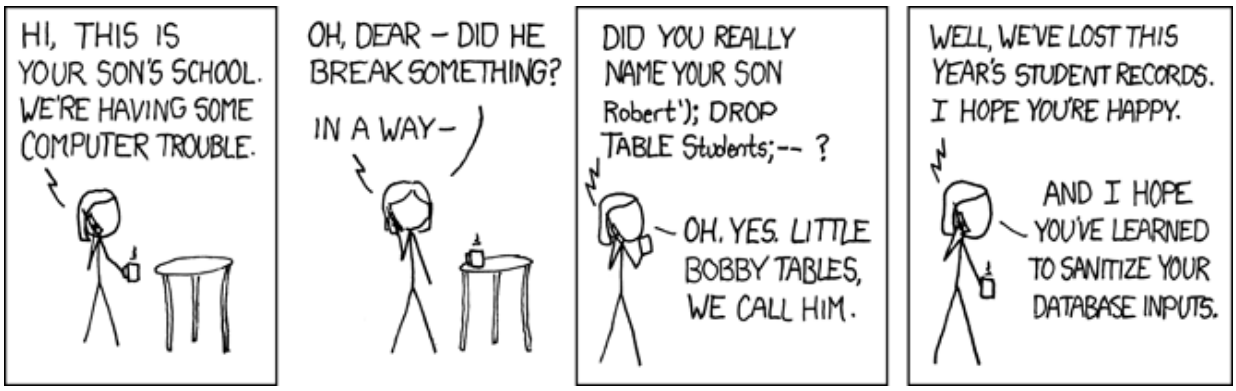
2. En vous aidant d'internet, entrez dans la table les films suivants : Sacré Graal, La vie de Brian, Brazil
3. En vous inspirant de la deuxième partie de ce TP, dire comment modifier la table `films` et créer une nouvelle table, pour éviter les doublons.

Exercice 6 Afin de retrouver votre professeur d'informatique sur Paris, vous vous munissez de la liste des lieux parisiens où le café est encore servi pour moins d'un euro. La liste [2] des noms et des adresses de ces lieux se trouve dans le fichier `cafes.csv`, dont l'extension signifie *comma separated values*.

1. Dans `sqliteman`, créez une nouvelle base de données dans un fichier nommé `cafes.db`.
2. Créez la table dont le schéma relationnel est :

   `cafes( nom:text, adresse:text)`

3. En utilisant la fonction `Import Table Data` du menu `Database`, importez les données du fichier `cafe.csv`.
4. Sachant qu'il aime retrouver son cantal natal, à quelle adresse le retrouverez-vous ?
5. Combien de cafés de la base ont leur adresse dans le sixième arrondissement ?
6. Traduisez « comma » et « semicolon » en français `:-)`



2. Récupérée sur `http://opendata.paris.fr`