1 TD 8 : SQL

(correction page ??)

Abordé lors de cette séance	
programmation	SQL
algorithme	index

Le langage SQL est utilisé pour manipuler des bases de données [1]. Pour faire simple, on utilise les bases de données quand on ne peut plus se servir d'Excel :
– le tableau dont on se sert est trop grand (comme trier 50000 lignes)
– on souhaite faire des opérations sur deux feuilles Excel (associer les lignes de l'une avec celles de l'autre)
Lorsque le volume de données est important, il est impossible de les voir dans leur ensemble. On peut en voir soit une partie soit une aggrégation. Par exemple, la société qui gère les vélib a ouvert l'accès à ses données. Il est possible de télécharger aussi souvent qu'on veut (toutes les minutes par exemple) un état complet des vélos et places disponibles pour toutes les stations de Paris : c'est une table qui s'enrichit de 1300 lignes toutes les minutes.

Première demi-heure : Récupérer les données et premier SELECT .

Ce TD nécessite les données suivantes :

```
from pyensae import download_data
download_data("SQLiteSpy.zip", website = 'xd')  # 2 secondes
download_data("td8_velib.zip", website = 'xd')  # 22 secondes (~7 Mo)
```

Après l'exécution de ce programme, vous devriez voir à côté de votre programme :

1. `SQLiteSpy.exe` : un programme qui permet d'explorer les bases de données SQLite.
2. `td8_velib.txt` : les données de ce TD.

Ensuite, on convertit les données au format texte en base de données au format SQLite :

```
from pyensae import import_flatfile_into_database
dbf = "td8_velib.db3"
import_flatfile_into_database(dbf, "td8_velib.txt")                   # 2 secondes
import_flatfile_into_database(dbf, "stations.txt", table="stations")  # 2 minutes
```

Vous devriez voir un fichier `td8_velib.db3` que vous pouvez ouvrir avec `SQLiteSpy.exe` (voir figure 1). Si ces étapes ne fonctionnent pas, vous pouvez directement obtenir le résultat en exécutant :

```
from pyensae import download_data
download_data("td8_velib.db3.zip", website = 'xd')  # 40 secondes (~12 Mo)
```

Lire une partie des données :

1. voir http://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es

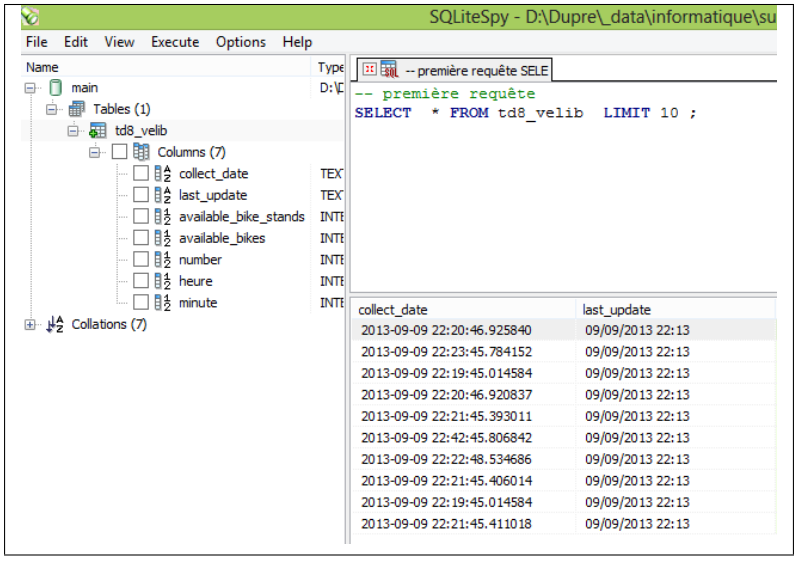

Figure 1 : *Copie d'écran du logiciel SQLiteSpy.exe avec les données du TD.*

```
-- sélectionner les données sur une plage horaire donnée
SELECT * FROM td8_velib WHERE last_update >= '2013-09-13 10:00:00' AND last_update <= '2013-09-13 11:00:00'

-- sélectionner certaines colonnes et ordonner les valeurs
SELECT available_bike_stands, available_bikes FROM td8_velib
WHERE last_update >= '2013-09-13 10:00:00' AND last_update <= '2013-09-13 11:00:00'
ORDER BY available_bike_stands DESC ;

-- compter le nombre d'emplacements de chaque station
SELECT last_update, available_bike_stands + available_bikes AS place, number FROM td8_velib
WHERE last_update >= '2013-09-13 10:00:00' AND last_update <= '2013-09-13 11:00:00'
ORDER BY place DESC ;
```

SELECT, MIN, MAX :

```
-- maximum de vélos disponibles sur toutes la base
SELECT MAX(available_bike_stands) FROM td8_velib

-- minimum, maximum de vélos disponibles sur toutes la base
SELECT MIN(available_bike_stands) FROM td8_velib
UNION ALL
SELECT MAX(available_bike_stands) FROM td8_velib

-- ajouter des informations
SELECT "min" AS label, MIN(available_bike_stands) FROM td8_velib
UNION ALL
SELECT "max" AS label, MAX(available_bike_stands) FROM td8_velib
```

SELECT, DISTINCT, COUNT, FROM :

```
-- tous les numéros de stations de façon unique
SELECT DISTINCT number FROM td8_velib

-- compter le nombre de stations (1230)
```

```
SELECT COUNT(*) FROM (
    SELECT DISTINCT number FROM td8_velib
)
```

Exercice 1

1. Déterminer le nombre de valeur distinctes pour la colonne `last_update`.
2. Déterminer la première et dernière date.

Seconde demi-heure : GROUP BY .

L'instruction `GROUPBY` permet d'aggréger des valeurs (min, max, sum) sur un ensemble de ligne partageant le même ensemble de valeurs (ou clé).

```
SELECT last_update, SUM(available_bikes) AS velo_disponible
FROM td8_velib
GROUP BY last_update
ORDER BY last_update
```

Le résultat est un tableau avec de petites valeurs au début et de grandes vers la fin. Cela est dû au processus de création de la base de données [2]. Il ne faudra considérer que les dates au-delà de 2013-09-10 11 :30 :19.

last_update	velo_disponible
...	...
2013-09-10 11:00:19	74
2013-09-10 11:05:19	74
2013-09-10 11:10:19	74
2013-09-10 11:15:19	118
2013-09-10 11:20:19	2495
2013-09-10 11:25:19	8809
2013-09-10 11:30:19	12813
2013-09-10 11:35:19	12817
2013-09-10 11:40:19	12744
...	...

Que fait la requête suivante ? Que se passe-t-il si vous enlevez les symboles "−−" (on *décommente* la condition `WHERE`).

```
SELECT last_update, SUM(available_bikes) AS velo_disponible, COUNT(DISTINCT number) AS stations
FROM td8_velib
--WHERE last_update >= "2013-09-10 11:30:19"
GROUP BY last_update
ORDER BY last_update
```

2. Certaines stations sont hors service et la dernière arrivée ou le dernier départ remonte à plusieurs jours. A chaque fois qu'on récupère les données velib, on dispose pour chaque station de la dernière arrivée ou du dernier départ de vélo. Le champ *last_ update* correspond à cette date.

Et celle-ci ?

```
SELECT last_update,
       CASE  WHEN available_bikes>0 THEN 1 ELSE 0 END AS vide,
       COUNT(*) AS nb
FROM td8_velib
WHERE last_update >= "2013-09-10 11:30:19"
GROUP BY last_update, vide
ORDER BY last_update
```

Exercice 2 :

Pour chaque station, compter le nombre de plages horaires de cinq minutes où il n'y a aucun vélo disponible.

Exercice 3 :

Si on note $X(s)$ le nombre de plages horaires de cinq minutes où il n'y a aucun vélo disponible, construire le tableau suivant : $k \to \#\{s|X(s) = k\}$.

Troisième demi-heure : JOIN

L'instruction `JOIN` sert à associer des lignes d'une table avec les lignes d'une autre table à partir du moment où elles partagent une information commune.

```
SELECT A.*, B.name  -- ajout du nom au bout de chaque ligne
FROM td8_velib AS A
JOIN stations AS B
ON A.number == B.number
```

On peut s'en servir pour calculer un ratio en associant les deux instructions `GROUP BY` et `JOIN`. L'instruction suivante permet d'obtenir la distribution des vélos disponibles sur la période d'étude pour chaque station.

```
SELECT A.*, 1.0 * A.available_bikes / B.nb_velo  AS distribution_temporelle
FROM td8_velib AS A
JOIN (
  SELECT number, SUM(available_bikes) AS nb_velo
  FROM td8_velib
        WHERE last_update >= "2013-09-10 11:30:19"
  GROUP BY number
) AS B
ON A.number == B.number
WHERE A.last_update >= "2013-09-10 11:30:19"
```

Exercice 4 :

Pour chaque station, déterminer la distribution du nombre de vélos disponibles pour chaque période horaire d'une journée (par station, il y aura donc 24 * 12 valeurs comprises entre 0 et 1.) Le résultat que vous devriez obtenir est illustré par la figure 2.

Quatrième demi-heure : Zones de travail et zones de résidences

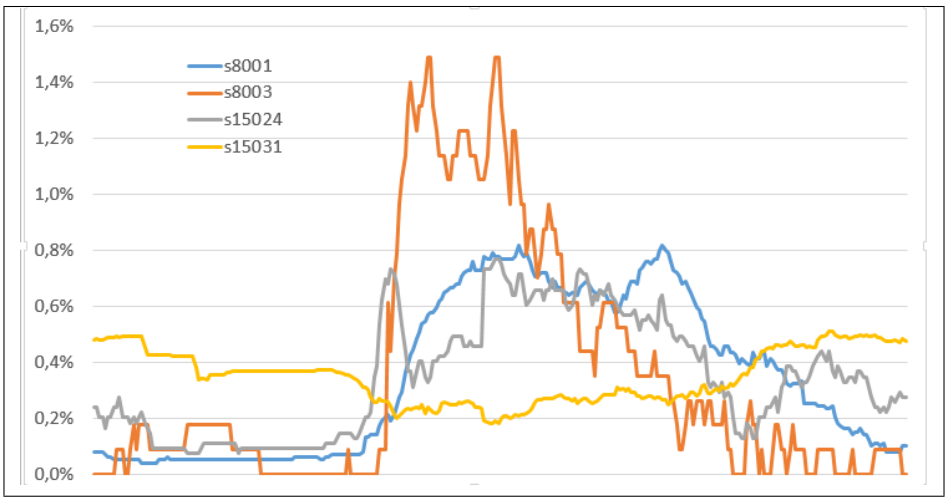

Figure 2 : *Distribution horaire du nombre de vélos pour 4 stations.*

On souhaite déterminer si une station se situe plutôt dans une zone de travail ou plutôt dans une zone de résidence. On part de l'hypothèse que, dans une zone de travail, les gens arrivent en vélib et repartent en vélib. C'est sans doute le cas de la station 8003 (voir figure 2). Les vélos seront plutôt disponibles dans la journée. A l'opposé, dans une zone de résidence, les vélos seront disponibles plutôt la nuit. Comment faire à partir de la distribution des vélos disponibles construite à la question précédente ?

On considère que la plage diurne s'étend de 10h à 16h. Vous trouverez une illustration du résultat dans cet article http://www.xavierdupre.fr/blog/2013-09-26_nojs.html.

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

On repart de la requête précédente pour effectuer un JOIN avec la table *stations* pour récupérer les coordonnées (lat, long). Après un copier/coller dans Excel, on peut situer les zones de travail sur la région parisienne.

Remarques .

Pour récupérer les données d'une base de données de type sqlite3 depuis *Python*, il suffit d'utiliser le module `sqlite3`[3]. Le court exemple suivant permet de récupérer dans *Python* le résultat d'une requête SQL :

```
import sqlite3
conn = sqlite3.connect(dbf)  # on ouvre une connexion sur la base de données
data = conn.execute("SELECT * FROM stations")  #on exécute une requête SQL
for d in data :    # on affiche le résultat
    print (d)      #
conn.close()       # on ferme la connexion
```

3. http://docs.python.org/2/library/sqlite3.html#module-sqlite3