Ingénierie des systèmes d'information.
(No)SQL. TP

Sergey Kirgizov et Amir Abdelkader

ESIREM, 2021

Vous êtes libres d'utiliser votre langage de programmation préféré.
Objectif : se familiariser avec les deux systèmes de gestion de bases de données : SQLite et MongoDB.

Mode du travail préférée : binômes.

L'exercice 1.4.1 de troisième TP sera évaluée, envoyez le code à l'enseignant de TP en fin de séance de TP3.

Nous travaillons avec le système d'exploitation GNU/Linux. Normalement, SQLite, MongoDB et Python sont disponibles dans les salles GR12, GR13 et GR19. De plus, depuis toutes les salles de l'Esirem, vous pouvez vous connecter sur le serveur GNU/Linux de la salle GR12 par ssh

```
ssh USERNAME@slocum.esirem-ad.ad.u-bourgogne.fr
```

ou bien

```
ssh USERNAME@10.169.20.12
```

1 SQLite (TP 1-3)

SQLite est une bibliothèque de gestion de base de données via le langage SQL. Vous pouvez télécharger le code source et les fichiers binaires de SQLite ici : `https://www.sqlite.org/download.html`. La documentation officielle est disponible également sur le net : `https://www.sqlite.org/`.

SQLite est installé dans les salles de TP. L'interface standard de SQLite est appelée `sqlite3`, sa documentation est disponible via le système de mans des systèmes Unix : `man sqlite3`. Un grand nombre de langages de programmation fournissent des liaisons avec la bibliothèque SQLite : Smalltalk, Scheme, Tcl, D, C, Lua, Perl, R, Python, Go, Java, Closure, etc.

1.1 Structure (TP 1)

Bitcoin ne stocke pas les états actuels des comptes dans sa blockchain, il enregistre uniquement les transactions [1]. Nous allons créer une petite base de transactions qui illustrera l'une des idées fondamentales du bitcoin : la liste des transactions.

1. Vous pouvez en savoir plus sur ce lien : `https://bitcoin.org/bitcoin.pdf`

☞ **EXERCICE 1.1.1.** Créer les deux tables suivantes et les connexions nécessaires.

VIREMENT

Colonne	Type
id	INTEGER PRIMARY KEY
from_pers_name	TEXT
to_pers_name	TEXT
amount	INTEGER
date	TEXT
country_code_from	TEXT
country_code_to	TEXT

COUNTRY

Colonne	Type
code	TEXT PRIMARY KEY
nom	TEXT

1.2 Jeu de tests (TP 1-2)

☞ **EXERCICE 1.2.1.** Remplir la base de données avec des données "aléatoires", générées par vous ou votre algorithme. Afin de tester confortablement la vitesse des requêtes, il est conseillé de générer plusieurs dizaines de millions de lignes.

ASTUCE :

- Pour insérer rapidement des données, utiliser *csv bulk-loading* : `https://sqlite.org/cli.html#csv_import`
- Les codes de pays sont disponibles ici `https://kirgizov.link/teaching/esirem/information-systems-2020/country.csv` ATTENTION : les codes ne sont pas uniques !

Voici un exemple de création et de remplissage de la table 'COUNTRY'. Nous supposons ici que les données des pays sont stockées dans le fichier 'country.csv'.

```
[user@computer /home/user/IngeSI/TP]$ sqlite3 tp.db
sqlite> CREATE TABLE COUNTRY (
   ...>     code TEXT PRIMARY KEY,
   ...>     name TEXT
   ...> );
sqlite>
sqlite> .mode csv
sqlite> .import country.csv country
sqlite> select count(*) from country;
208
sqlite> [CTRL-D]
[user@computer /home/user/IngeSI/TP]$
```

Voici un exemple de script en Python permettant de générer quelques millions de transactions sous format '.csv'.

```
import random
import datetime

PERS = ['TOTO', 'TATA', 'BULB']
COUN = ['FRA','BMU','BEL','HND']
```

```
date = datetime.datetime.now ()
delta = datetime.timedelta(0,50) # 50 sec
million = 1000000
f = open ("transactions.csv","w")
for i in range (10 * million):
  frm   = random.choice (PERS)
  to     = random.choice (PERS)
  amount = random.random () * 100
  date   += delta
  c1 = random.choice (COUN)
  c2 = random.choice (COUN)
  f.write ("{},{},{},{},{},{},{}\n".format( i, frm, to, amount, date, c1, c2))

f.close ()

## Import data to sqlite
## [user@ordin IngeSI/TP]$ sqlite3 tp.db
## sqlite> .mode csv
## sqlite> .import transactions.csv virement
```

ASTUCE : Vous pouvez utiliser logiciel Thonny pour l'exécution de scripts en Python. Ce logiciel est installé dans les salles de TP. Il est aussi disponible sur le net : `https://thonny.org/`.

EXERCICE 1.2.2. Concevoir quelques requêtes SQL. Tester les requêtes, en mesurant le temps d'exécution. Ajouter les indices nécessaires et retester à nouveau les requêtes. C'est devenu plus rapide ?

Exemple :

```
sqlite> select count(*) from virement where amount > 90;
[réponse lente]
999346
sqlite> create index virement_amount_ind on virement (amount);
sqlite> select count(*) from virement where amount > 90;
[réponse rapide]
999346
sqlite>
```

EXERCICE 1.2.3. Avec ce système nous ne stockons que des transactions. Comment savoir combien d'argent une certaine personne a sur son compte ? Rédiger la requête SQL correspondante. Quels index SQL peut-on créer dans ce cas ?

1.3 Triggers (TP 2)

EXERCICE 1.3.1. Créer une table "PERSONNE" pour les soldes des comptes des personnes.

PERSONNE

Colonne	Type
pers_name	TEXT PRIMARY KEY
solde	INTEGER

☞ **EXERCICE 1.3.2.** En utilisant les données de la table "VIREMENT", mettre à jour la table "PERSONNE".

La mise à jour manuelle des données après chaque nouvelle transaction n'est pas très pratique. C'est une activité ennuyeuse qui n'est pas à l'abri des erreurs. C'est beaucoup mieux de mettre à jour automatiquement la valeur du champ "solde" de la table "PERSONNE" lorsque quelqu'un met à jour la table "VIREMENT". Pour ça on peut utiliser les triggers SQL. Voici la documentation de SQLite : `https://www.sqlite.org/lang_createtrigger.html`.

☞ **EXERCICE 1.3.3.** Écrire des triggers permettant la mise à jour automatique des valeurs du champ "solde" de la table "PERSONNE" lorsque quelqu'un met à jour la table "VIREMENT".

Voici un exemple de trigger pour l'opération "INSERT". Ce trigger contient une erreur. Laquelle ?

```
CREATE TRIGGER insert_transaction
 AFTER INSERT ON virement
 BEGIN
   -- create PERSONNEs if someone not exist
   INSERT INTO PERSONNE (pers_name, solde)
   SELECT new.from_pers_name, 0
   WHERE NOT EXISTS (SELECT 1
                     FROM PERSONNE
                     WHERE pers_name = new.from_pers_name);
   INSERT INTO PERSONNE (pers_name, solde)
   SELECT new.from_pers_name, 0
   WHERE NOT EXISTS (SELECT 1
                     FROM PERSONNE
                     WHERE pers_name = new.from_pers_name);
   UPDATE PERSONNE SET solde = solde + new.amount WHERE pers_name = new.to_pers_name ;
   UPDATE PERSONNE SET solde = solde - new.amount WHERE pers_name = new.from_pers_name ;
END;
```

☞ **EXERCICE 1.3.4.** Tester bien les triggers dans tous les cas : insert, update, delete, nouvelle personne, personne existante, etc.

1.4 Ajout de nouvelles fonctions dans SQLite (TP 3)

Il est possible d'étendre SQLite par les fonctions définies par l'utilisateur. Voici la documentation officielle, qui utilise le langage C : `https://sqlite.org/c3ref/create_function.html`.

Si vous écrivez en python, ce sera plus pratique de lire cela :

- `https://docs.python.org/3.8/library/sqlite3.html#sqlite3.Connection.create_function`
- `https://docs.python.org/3.8/library/sqlite3.html#sqlite3.Connection.create_aggregate`

Voici un exemple de création et d'utilisation de deux nouvelles fonctions :
- `mode` : une fonction d'agrégation statistique, la valeur dominante.
- `fib` : le nième élément de la séquence de Fibonacci.

Ce code contient une erreur. Laquelle ?

```
import sqlite3

# SQLite is an embeded database system.
# While any trigger resides in a database file,
# the user defined functions are stored in the client code only, not in the database itself !

# In a client-server database (like Postgres, MariaDB, Oracle)
```

```
# user defined functions and triggers are both stored in the database.

class Mode:
    def __init__(self):
        self.tbl = []
    def step(self, value):
        self.tbl.append (value)
    def finalize(self):
        """ Returns first most common element """
        return max (set (self.tbl), key = self.tbl.count)

def fibonacci (n):
    if n == 0 : return 0
    if n == 1 : return 1
    return fibonacci (n-1) + fibonacci (n-2)

conn = sqlite3.connect('tp.db')
cur = conn.cursor()
conn.create_aggregate ("mode", 1, Mode)
conn.create_function ("fib", 1, fibonacci)

cur.execute("SELECT mode(amount) FROM VIREMENT")
rows = cur.fetchall()
for row in rows : print(row)

cur.execute("select fibo(13)")
rows = cur.fetchall()
for row in rows : print(row)

cur.close()
conn.close()
```

L'exercice 1.4.1 sera évaluée, envoyez le code à l'enseignant de TP en fin de séance.

☞ **EXERCICE 1.4.1.** Ajoutez les fonctions statistiques dans SQLite : la médiane et l'écart-type. Inspirez-vous de l'exemple précédent de la fonction mode.

☞ **EXERCICE★ 1.4.2.** Ajoutez les fonctions statistiques dans SQLite : coefficient d'asymétrie (skewness), kurtosis.

☞ **EXERCICE 1.4.3.** Que se passera-t-il si nous déclarons une fonction en python, créons un trigger qui utilise cette fonction, puis déclenchons ce trigger via une autre interface, par exemple, via "`sqlite3`" ?

2 MongoDB (TP 4)

Documentation :
- `https://docs.mongodb.com/`
- `man mongod`
- `man mongo`

Installation `https://www.mongodb.com/download-center/community`.
Pour lancer un serveur de MongoDB, Il faut d'abord créer un répertoire vide dans lequel le MongoDB va créer les fichiers de la base de données.

```
# Créer un répertoire
mkdir chemin/vers/un/repertoire/des/donnees

# Lancer le serveur
mongod --dbpath chemin/vers/un/repertoire/des/donnees --bind_ip 127.0.0.1 --port 27017
```

Dans un autre terminal, lancer un client

```
# Lancer le client mongo
mongo --host 127.0.0.1 --port 27017
```

ASTUCE : Dans la salle GR12 tous les écrans, claviers et souris sont connectés au même ordinateur-serveur. Je vous conseille d'utiliser les différentes adresses IP et les différentes ports liées au réseau local pour lancer plusieurs serveurs MongoDB pour différents binômes :

```
# Binôme 1
mongod --dbpath chemin1/vers/un/repertoire/des/donnees --bind_ip 127.0.0.1 --port 27017

# Binôme 2
mongod --dbpath chemin2/vers/un/repertoire/des/donnees --bind_ip 127.0.0.2 --port 27018

# Binôme 3
mongod --dbpath chemin3/vers/un/repertoire/des/donnees --bind_ip 127.0.0.3 --port 27019
...
```

Cette information doit également être prise en compte lorsque vous vous connectez à ces serveurs :

```
# Binôme 1
mongo chemin1/vers/un/repertoire/des/donnees --bind_ip 127.0.0.1 --port 27017

# Binôme 2
mongo chemin2/vers/un/repertoire/des/donnees --bind_ip 127.0.0.2 --port 27018

# Binôme 3
mongo chemin3/vers/un/repertoire/des/donnees --bind_ip 127.0.0.3 --port 27019
...
```

EXERCICE 2.0.1. Voici un petit tutoriel sur MongoDB. Essayer ces commandes dans votre terminal.

```
// Show documentation
help

// Show current database
db
// List all databases
```

```
show dbs

// Switch to the database 'coincoin'.
// If a database does not exist it will be created
use coincoin

// Create some collections in the current database
db.createCollection("virement")
db.createCollection("country")

// Insert some data
db.country.insert([
{ code : "FRA", name : "France" },
{ code : "BEL", name : "Belgique" },
{ code : "CUB", name : "Cuba"},
{ code : "MLT", name : "Malete"}
])

db.virement.insert({
"from_pers_name" : "TOTO",
"to_pers_name"   : "TATA",
"amount"         : 20,
"date"           : "3999-12-31 23:59:59",
"country_code_from" : "BEL",
"country_code_to"   : "FRA"
})

// Select all data
db.virement.find()

// Select some data
db.virement.find( { from_pers_name: "TOTO" } )

// Select some data (amount > 10)
db.virement.find( { amount: { $gt: 10 } } )

// Filter & Sum
db.virement.aggregate([
    { $match: { to_pers_name: "TOTO"}},
    { $group: { _id : null, "total": { $sum: "$amount" }}}
])

// We can write fuctions in js !
function solde (pers) {
  let input = 0;
  let output = 0;
  let c = db.virement.aggregate([
    { $match: { to_pers_name: pers}},
    { $group: { _id : null, "total": { $sum: "$amount" }}}
  ])
  if ( c.hasNext() ) input = c.next().total;
  c = db.virement.aggregate([
    { $match: { from_pers_name: pers}},
    { $group: { _id : null, "total": { $sum: "$amount" }}}
  ])
  if ( c.hasNext() ) output = c.next().total
  return input - output;
}
```

```
// then run it
solde ('TOTO')
```

La documentation détaillée est disponible sur le net. Voici, par exemple, comment créer des index : `https://docs.mongodb.com/manual/indexes/`.

EXERCICE 2.0.2. Lire la documentation sur les index. Quelle structure de données mongodb utilise pour les index ?

EXERCICE 2.0.3. Créer quelques index.

3 Cadre juridique

EXERCICE 3.0.1. Trouver et lire les licences et les conditions d'utilisation de SQLite et MongoDB.