

TP3/4 – Base de données NoSQL

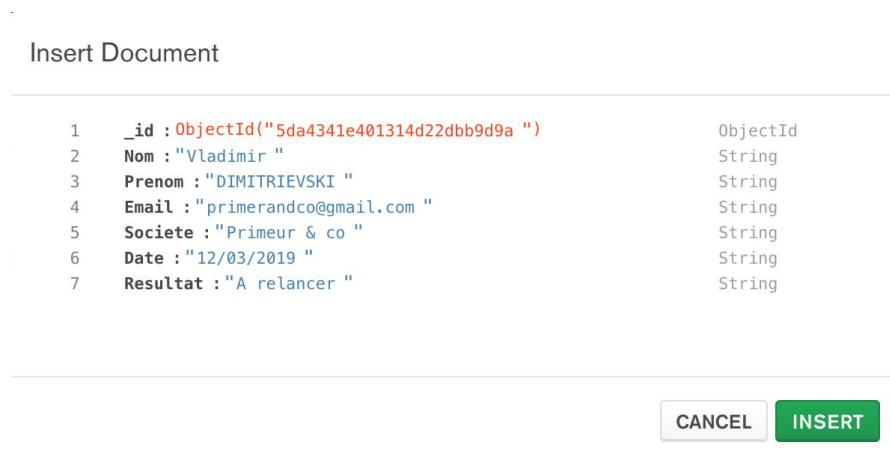
Introduction

Dans ce TP, nous allons utiliser une base de données noSQL type “Document”(MongoDB) et une autre de type “colonne” (Cassandra). Nous allons aussi utiliser les diagrammes de CHEBOTKO pour construire le schéma de la base Cassandra.

1) MongoDB

a) Insertion des données

Nous avons utilisé MongoDB Compass pour insérer les informations clients. Pour chaque collaborateur, nous avons créé une collection, qui contient les fiches (document) extraites d’Excel. Exemple d’insertion de données pour Pascal Lelievre :



The screenshot shows the 'Insert Document' dialog in MongoDB Compass. It contains a list of document fields and their types:

Field	Type
<code>_id : ObjectId("5da4341e401314d22dbb9d9a ")</code>	ObjectId
<code>Nom : "Vladimir "</code>	String
<code>Prenom : "DIMITRIEVSKI "</code>	String
<code>Email : "primerandco@gmail.com "</code>	String
<code>Societe : "Primeur & co "</code>	String
<code>Date : "12/03/2019 "</code>	String
<code>Resultat : "A relancer "</code>	String

At the bottom right, there are two buttons: 'CANCEL' and 'INSERT'.

b) Connexion à la base avec Python

Nous avons utilisé python pour connecter à la base(Merigold),la collection(collaborateur), puis faire une requête de connexion.

```
from pymongo import MongoClient  
  
client = MongoClient('localhost', 27017)  
db = client.merigold
```

1) Nombre de fiches

Pour calculer le nombre de fiche, nous utilisons méthode “count.documents({})” dans chaque collection puis nous faisons une boucle “for” pour faire la somme. L’utilisation d’un document vide

("{}") nous permet de sélectionner tous les documents d'une collection.

Le code Python et le résultat sont les suivants :

```
##### Nombre de fiches
compteur = 0
for collection_name in db.list_collection_names():
    collection = db.get_collection(collection_name)
    compteur += collection.count_documents({}) # Document vide pour tout sélectionner
print(f'Nombre de fiches : {compteur}')
```

Nombre de fiches : 6

2) Fiche de Jacques WEBER

Pour obtenir l'information de Jacques WEBER, nous pouvons juste rechercher dans la collection de Alice DUMOND(Nous savions où il est).

Le code Python et le résultat sont les suivants :

```
##### Fiche de Jacques WEBER
collection = db.get_collection('Alice Dumond')
fiche = collection.find_one({'Nom-Prenom': 'Jacques WEBER'})
pprint.pprint(fiche)
```

```
{'Email': 'Jacques.Weber@fruitcompany.com',
 'Nom-Prenom': 'Jacques WEBER',
 'Resultat': 'Attente avis',
 'Societe': 'My Fruit Company',
 'Tel': '+33 6 65 89 56 34',
 '_id': ObjectId('5da43304401314d22dbb9d97')}
```

3) Fiche d'entreprise "Primeur & co"

Le code Python et le résultat sont les suivants :

```
##### Fiche d'entreprise "Primeur & co"
collection = db.get_collection('Pascal Lelievre')
fiche = collection.find_one({'Societe': 'Primeur & co'})
pprint.pprint(fiche)
```

```
{'Date': '12/03/2019',
 'Email': 'primerandco@gmail.com',
 'Nom': 'Vladimir',
 'Prenom': 'DIMITRIEVSKI',
 'Resultat': 'A relancer',
 'Societe': 'Primeur & co',
 '_id': ObjectId('5da4341e401314d22dbb9d9a')}
```

4) Nombre de fiche de chaque collaborateur

Le code est similaire au 1). Au lieu d'incrémenter un compteur, on affiche à chaque itération le nom de la collection ainsi que le nombre de fiches dans la collection.

Le code Python et le résultat sont les suivants :

```
### Nombre de fiches de chaque collaborateur
for collection_name in db.list_collection_names():
    collection = db.get_collection(collection_name)
    nb_fiches = collection.count_documents({})
    print(f'{collection_name} a {nb_fiches} fiches')
```

Pascal Lelievre a 2 fiches
Laetitia Dupond a 2 fiches
Alice Dumond a 2 fiches

2) Cassandra

a) Diagrammes de CHEBOTKO

Première requête :

On souhaite, pour chaque camion, obtenir les position GPS et l'heure de relevé. Le diagramme proposé est le suivant :

camion_id	text	K
heure	timestamp	C
gps_x	double	
gps_y	double	

Pour effectuer cette requête, on effectue un partitionnement suivant **camion_id** ce qui explique que cette colonne soit utilisée comme Primary Key. De cette manière, toutes les données relatives à un camion seront au sein d'une même partition dans la base. La colonne **heure** est une colonne de clustering car elle fait partie de la clé primaire du schéma (un relevé GPS est identifié de manière unique par **camion_id, heure**) mais n'est pas utile pour le tri des lignes dans notre requête.

Deuxième requête :

On souhaite, à chaque instant, pour chaque camion, obtenir les position GPS. Le diagramme proposé est le suivant :

heure	timestamp	K
camion_id	text	C
gps_x	double	
gps_y	double	

Pour effectuer cette requête, on effectue un partitionnement suivant **heure** ce qui explique que cette colonne soit utilisée comme Primary Key. De cette manière, toutes les données relatives à une heure

seront au sein d'une même partition dans la base. La colonne **camion_id** est une colonne de clustering.

b) Connexion à la base et requêtes

Camions présents dans la base :

On commence par lister les camions présents dans la base ainsi que le nombre de lignes associé à chaque camion. Le code Python et le résultat sont les suivants :

```
### Lister tous les camions et le nombre de données dans la base
rows = session.execute("SELECT camion_id, COUNT(*) FROM camion GROUP BY camion_id")
for camion in rows:
    print(camion)
```

```
Row(camion_id='BA-865-PF', count=761)
Row(camion_id='CN-225-AB', count=216)
Row(camion_id='AD-671-KA', count=1115)
Row(camion_id='AC-543-AG', count=1076)
```

Liste des données de tous les camions :

On affiche les données de toute la base avec le code Python suivant :

```
### Lister les données de tous les camions
rows = session.execute("SELECT * FROM camion")
for user_row in rows:
    print (user_row)
```

L'affichage est trop long pour être inséré ici.

Liste des données associées à un camion en particulier :

On choisit de lister les données du camion AD-671-KA. Pour ne garder que les données de ce camion, on utilise le mot-clé WHERE. On peut ensuite insérer nous-même le chaîne de caractères correspondant à l'immatriculation du camion dans notre requête SQL, sans oublier les guillemets simples ou utiliser un dictionnaire contenant le **camion_id** et le donner en paramètre à la fonction **execute**. La deuxième méthode est préférable car elle insère automatiquement les guillemets et se cherche d'échapper les guillemets présents dans les valeurs du dictionnaire.. Le code Python est le suivant :

```
### Lister les données d'un camion donné
data = {"camion_id": 'AD-671-KA'}
rows = session.execute("SELECT * FROM camion WHERE camion_id = %(camion_id)s", data)
compteur = 0
for camion_row in rows:
    print(camion_row)
    compteur += 1
print("Nombre de lignes :" + str(compteur))
```

Quelques lignes affichées :

```
Row(camion_id='AD-671-KA', heure=datetime.datetime(2019, 7, 26, 8, 5, 28),  
gps_x=5.850850068119164, gps_y=43.44510125978476)  
Row(camion_id='AD-671-KA', heure=datetime.datetime(2019, 7, 26, 8, 10, 28),  
gps_x=5.795596105270221, gps_y=43.45100581753382)  
Row(camion_id='AD-671-KA', heure=datetime.datetime(2019, 7, 26, 8, 15, 28),  
gps_x=5.7342551445673005, gps_y=43.45756084501381)  
Row(camion_id='AD-671-KA', heure=datetime.datetime(2019, 7, 26, 8, 20, 28),  
gps_x=5.676169484946878, gps_y=43.46376800399277)
```

L’affichage du compteur est le suivant :

Nombre de lignes :1115

D’après le résultat de la première requête, on a bien récupéré toutes les données du camion.

Vérifier l’accès aux données des GPS sur une plage horaire donnée

On choisit de lister les données GPS entre 8h00 et 8h30 le 2019-07-01. On utilise “ALLOW FILTERING” dans la requête CQL pour pouvoir utiliser “<=” et “>=”. Le code Python est le suivant :

```
##%% Données GPS entre 8h00 et 9h00 le 2019-07-01  
data = {"heure_debut": "2019-07-01 08:00:00", "heure_fin": "2019-07-01 08:30:00"}  
rows = session.execute("SELECT * FROM position WHERE heure >= %(heure_debut)s \\  
                        AND heure <= %(heure_fin)s ALLOW FILTERING", data)  
  
for row_heure in rows:  
    print(row_heure)
```

Résultat :

```
Row(heure=datetime.datetime(2019, 7, 1, 6, 11, 16), camion_id='BA-865-PF', gps_x=-0.5720828857536058, gps_y=43.377875085145554)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 21, 16), camion_id='BA-865-PF', gps_x=-0.6781263749825133, gps_y=43.417609049691016)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 16, 16), camion_id='BA-865-PF', gps_x=-0.6232230525203907, gps_y=43.397037049618206)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 26, 16), camion_id='BA-865-PF', gps_x=-0.7382175611505158, gps_y=43.44012491632324)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 1, 16), camion_id='BA-865-PF', gps_x=-0.461849, gps_y=43.336571)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 11, 32), camion_id='AC-543-AG', gps_x=6.135056035542229, gps_y=49.11463603243542)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 6, 16), camion_id='BA-865-PF', gps_x=-0.5106005661333186, gps_y=43.354837967772966)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 16, 32), camion_id='AC-543-AG', gps_x=6.0694540181216645, gps_y=49.11482489193362)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 6, 32), camion_id='AC-543-AG', gps_x=6.18995, gps_y=49.114478)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 26, 32), camion_id='AC-543-AG', gps_x=5.957862806435342, gps_y=49.11514614821615)  
Row(heure=datetime.datetime(2019, 7, 1, 6, 21, 32), camion_id='AC-543-AG', gps_x=6.013481627084294, gps_y=49.11498602901599)
```

Conclusion

Dans ce TP, nous avons mis en oeuvre des connexions à plusieurs types de bases NoSQL afin de lire et d’écrire des données. Nous avons aussi découvert les diagrammes de CHEBOTKO qui permettent de trouver le schéma d’une base Cassandra à partir des requêtes que l’on souhaite faire.