Compte rendu du TP1- TP1bis NoSql

Appel à l'ordre

Le but de ce travail pratique est de plonger l'ensemble des étudiants à une prise en main de mongoDB. Ceci est réalisé à partir 06 Mars 2018 à l'Université de Pau et encadré par Annig Le Parc LACAYRELLE , maître de conférences au département informatique de l'Université.

Participants

GNEBEHI Bagre & KOUAME-KODIA Maryline

1 Installation et démarrage de mongoDB

1-a Installation

Il est d'abord nécessaire de créer un répertoire dans Home qui servira d'espace de travail. Ce répertoire porte le nom de bgnebehi.

Les instructions contenu dans le fichier InstallMongo.txt issu du répertoire install/NoSQL sont d'excellent conducteurs.

1-b Lancer le serveur mongoGB

Dans un terminal on lance la commande ci dessous pour lancer le serveur .

```
./mongod --dbpath /Home/bgnebehi/data/db/
```

--dbpath permet de se positionner dans le répertoire de stockage

1-c Lancer le shell

Il s'agit de lancer le client. Après avoir lancé le serveur , dans un autre terminal et en parallèle nous lançons le shell de la façon suivante :

```
./mongo
```

S'en suit alors l'ouverture du shell ou l'on commencera la prise en main.

```
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

1-d Arrêter le serveur mongoDB

On se positionne sur la base de donnée admin , puis on arrête le serveur.

```
> use admin
switched to db admin
> db.shutdownServer()
```

1-e Quitter le shell , Puis relancer le serveur et le shell

Ceci se fait via la commande suivante :

```
> quit()
bgnebehi@scinfe052:/Home/bgnebehi/mongodb-linux-x86_64-ubuntu1804-4.0.6/bin$
```

Il faut donc lancer un serveur puis un client pour le démarrage de mongoDB.

Pour quitter le shell , il est nécessaire de stopper le serveur en premier.

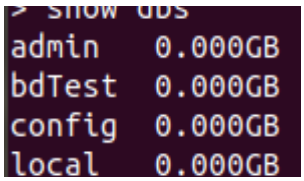
2 Création d'une base de données

2-a Créer une base de données nommée bdTest

La commande suivante est l'outil idéal :

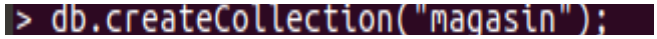
```
> use bdTest
switched to db bdTest
```

2-b Afficher les bases de données du serveur mongod

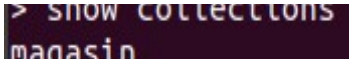


On constate que notre base bdTest est créée. En effet la commande *use* permet de se positionner sur une base et d'en créer si elle n'existe pas.

2-c Créer une collection nommée magasin



2-d Afficher la liste des collection de bdTest



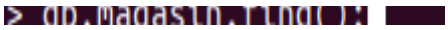
Magasin devient donc une collection de bdTest

3 Insertion de données


db.magasin.insert({"id": 1,
"libelle":"ChaussureXX",
"rayon": "chaussure",
"stock":[{"pointure":40,
"qstock":5},
{"pointure":41,
"qstock":3}]
});
db.magasin.insert({"id": 2,
"libelle":"PantalonXXZ",
"rayon": "vêtement",
"stock":[{"taille":"S",
"qstock":10},
{"taille":"M",
"qstock":4},
{"taille":"L",
"qstock":15}]
});
db.magasin.insert({"id": 3,
"libelle":"TennisWW",
"rayon": "chaussure",
"stock":[{"pointure":43,
"qstock":0}]
});
db.magasin.insert({"id": 4,
"libelle":"CasquetteBB",
"rayon": "accessoire",
"stock":[{"taille":"U",
"qstock":50}]
});
db.magasin.insert({"id": 5,
"libelle":"Sac à mainVW",
"rayon": "accessoire",
"stock":[{"taille":"U",
"qstock":0}]

4 Interrogation de la base

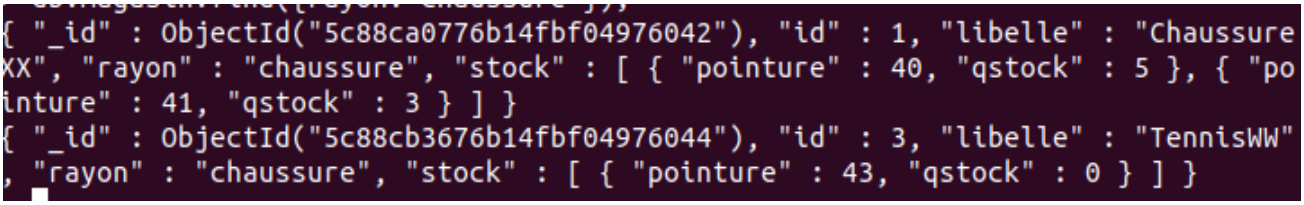
4-a Obtenir la liste des produits



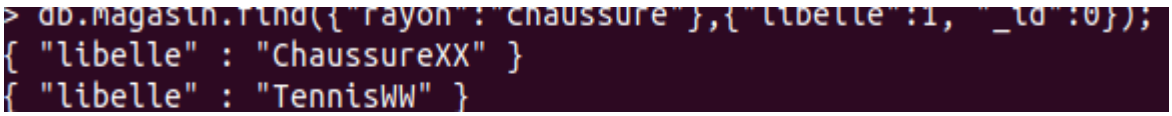
4-b Obtenir le nombre de produit

```
> db.magasin.find().count();
5
```

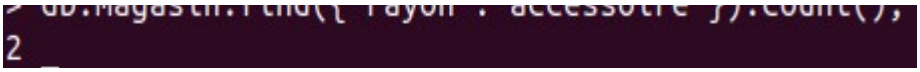
4-c Obtenir la liste des produits du rayon chaussure



4-d Obtenir le libellé des produits du rayon chaussure



4-e Compter la liste des articles du rayon accessoire



4-f Obtenir le libellé et le rayon des produits en rupture de stock

```
{ "_id" : ObjectId("5c88d5cfdf3002e0dc2ae5cc"), "libelle" : "TennisWW", "rayon"
: "chaussure" }
{ "_id" : ObjectId("5c88d5dedf3002e0dc2ae5ce"), "libelle" : "Sac à mainVW", "ray
on" : "accessoire" }
```

➔ **stock.qstock précise la profondeur de la structure json**

5 Modification

5-a Ajout d'une taille au produit 2

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

➔ **.update modifie la base de donnée**

5-b Modifier la quantité en stock de la 2ieme taille du produit 1

```
db.magasin.update({"id":1},{$set:{"stock.1.qstock":"45"}});
```

5-c Ajouter un prix de 40€ à tous les produits du rayon chaussure

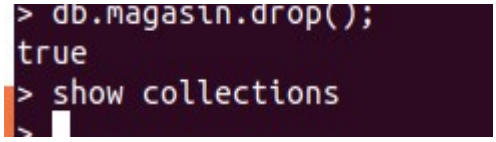
```
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
```

6 Suppression

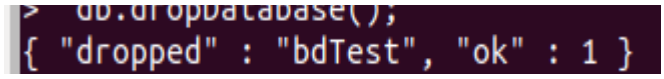
6-a Supprimer les articles du rayon accessoire

```
> db.magasin.deleteMany({"rayon":"accessoire"});
{ "acknowledged" : true, "deletedCount" : 2 }
```

6-b Supprimer la collection magasin.



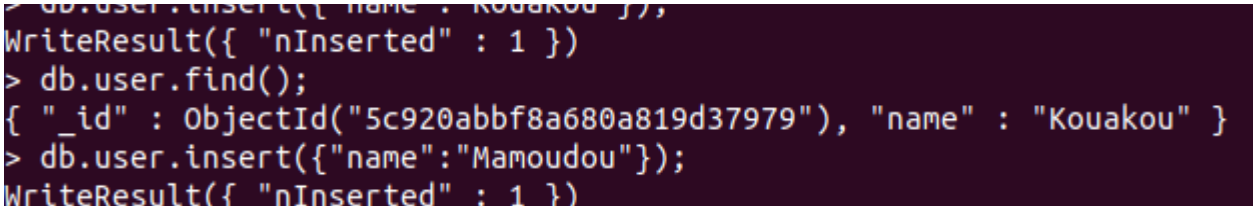
6-c Supprimer la base de donnée bdTest



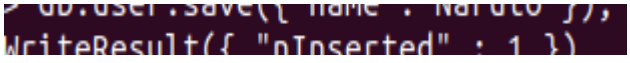
7 Question Supplémentaire - TP1bis

A partir de la commande db.user.insert(); on insère deux utilisateurs de noms Kouakou et Mamoudou.

Leur id est généré automatiquement dans la collection user.

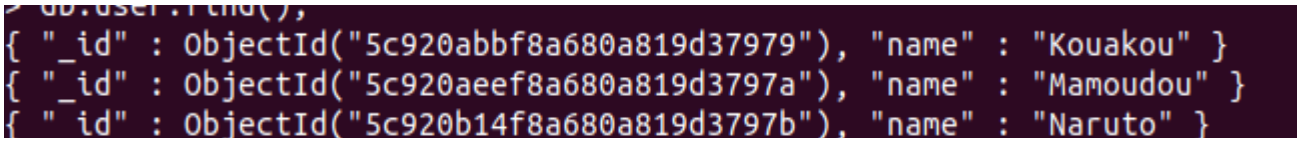


Avec la commande db.user.save() on ajoute un nouvel utilisateur du nom de Naruto.

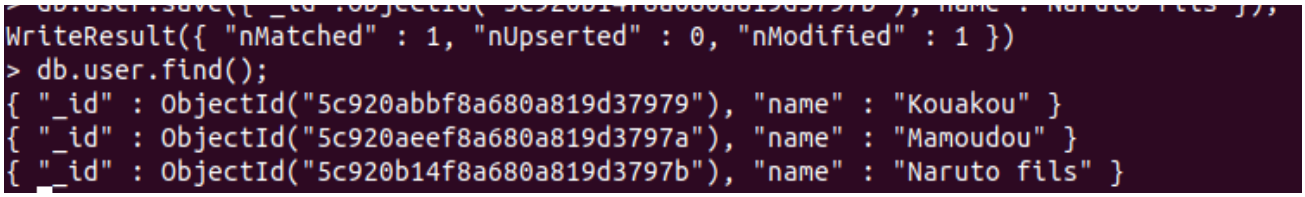


Jusque là les comportement des deux commandes sont les mêmes.

On obtient :



Lorsqu'on appelle la commande db.user.save() avec un id déjà existant , la commande sera comme save sera appelée comme une commande de modification.



Comme le montre l'exemple ci-dessus , l'utilisateur du nom Naruto est remplacé par l'utilisateur du nom Naruto fils.

Compte rendu du TP2 NoSql

1 Appel à l'ordre

Le but de ce travail pratique est de plonger l'ensemble des étudiants à une **interrogation de mongodb.** Ceci est réalisé à partir **20 Mars 2018** à **l'Université de Pau** et encadré par **Annig Le Parc LACAYRELLE** , maître de conférences au département informatique de l'Université.

2 Import d'une base de données

Dans un terminal on lance la commande ci dessous pour lancer le serveur .

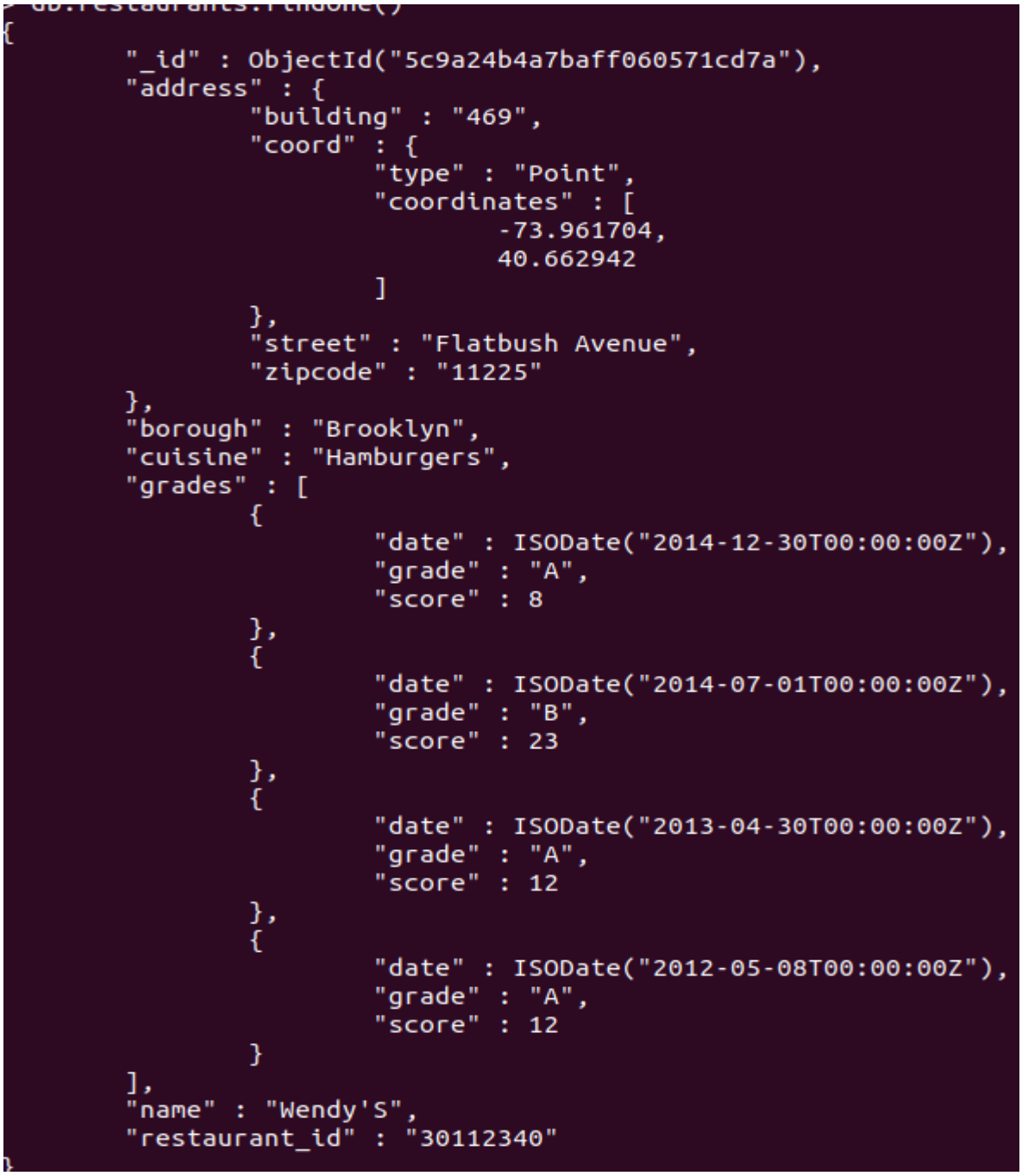
```
./mongoimport -d new_york -c restaurants restaurants.json_/restaurants.json
```

3 Interrogation de la base de données

3a_ Obtenir le nombre de restaurants

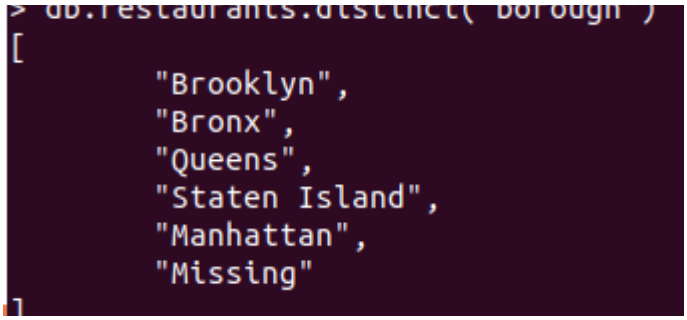
```
> db.restaurants.find().count()
25357
```

3b Obtenir un restaurant (utiliser `findOne`)



Nous renvoi le premier document de la collection restaurants

3c_ Obtenir la liste des quartiers.



Le distinct est utilisé pour éviter les répétitions.

3d_ Obtenir la liste des différents types de cuisine

```
> db.restaurants.distinct("cuisine");
```

3e_ Obtenir le nom des restaurants du quartier du Queens

```
> db.restaurants.find({"borough" : "Queens"},{ _id : 0, "name" : 1})
```

3f_ Combien y-a-t ’il de restaurants dans le Queens ?

```
> db.restaurants.find({ "borough" : "Queens" }).count();
5656
```

3g _ Obtenir le nom des restaurants Italien de Brooklyn situés sur la « 5 Avenue ».

db.restaurants.find({“cuisine”: “Italian”,”address.street”:”5 Avenue”,”borough”:”Brooklyn”},{“_id”:0,”name”:1})

3h_ Obtenir le nom des restaurants Italien de Brooklyn situés sur la « 5 Avenue » et qui contiennent le mot pizza dans leur nom.

db.restaurants.find({“cuisine”: “Italian”,”address.street”:”5 Avenue”,”borough”:”Brooklyn”,”name:/Pizza/”},{“_id”:0,”name”:1})

3i_ Obtenir les restaurants grecques du Queens ayant eu une note inférieure à 10 lors d’une évaluation. Vous afficherez le nom et la liste des scores

3j_ Obtenir les restaurants grecques du Queens ayant eu une note inférieure à 6 et pas de notes supérieures à 15. Vous afficherez le nom et la liste des scores.

db.restaurants.find({"cuisine" : "Greek","borough":"Queens","grades.score":{$lte:6,$not:{$gte:10}}}, {"_id":0,"name":1,"grades.score":1})

3k_ Obtenir les restaurants grecques du Queens ayant eu une note inférieure à 35 avec une évaluation C. Vous afficherez le nom et la liste des scores.

db.restaurants.find({"cuisine" : "Greek","borough":"Queens","grades.score":{$lte:35}, "grades.grade":"C"}, {"_id":0,"name":1,"grades.score":1,"grades.grade":1})

3l_ Obtenir le nom et quartier des restaurants ayant eu un C à leur dernière évaluation(elle est en première position dans la liste). La liste résultat sera triée par nom des restaurants

```
> db.restaurants.find({"grades.0.grade":"C"},{"_id":0,"name":1,"borough":1})
```

3m_ Obtenir le nombre de restaurant par quartier ayant eu un C à leur dernière évaluation. Vous afficherez le résultat par ordre alphabétiques des quartiers.

```
varGroup2 = { $group : {"_id" : "$borough", "total" : {$sum : 1}}};
varMatch = { $match : { "grades.0.grade":"C"} };
varSort2 = { $sort : {"_id":1} };
db.restaurants.aggregate( [ varMatch, varGroup2, varSort2 ] );
```

```
{ "_id" : "Brooklyn", "total" : 56 }
{ "_id" : "Manhattan", "total" : 83 }
{ "_id" : "Queens", "total" : 47 }
{ "_id" : "Staten Island", "total" : 7 }
```

3n_ Donner le score moyen des restaurants par quartier. Vous afficherez le résultat par ordre décroissant.

```
varUnwind = {$unwind : "$grades"};
varGroup3 = { $group : {"_id" : "$borough", "moyenne" : {$avg : "$grades.score"} } };
varSort3 = { $sort : { "moyenne" : -1 } };
db.restaurants.aggregate( [ varUnwind, varGroup3, varSort3 ] );
```

```
{ "_id" : "Queens", "moyenne" : 11.634865110930088 }
{ "_id" : "Brooklyn", "moyenne" : 11.447723132969035 }
{ "_id" : "Manhattan", "moyenne" : 11.41823125728344 }
{ "_id" : "Staten Island", "moyenne" : 11.370957711442786 }
{ "_id" : "Bronx", "moyenne" : 11.036186099942562 }
{ "_id" : "Missing", "moyenne" : 9.632911392405063 }
```

4 Plan d’exécution de indexation

4 a Afficher le plan d’exécution

```
> db.restaurants.explain().find( { "borough": "Queens"},{_id:0,"name":1})
```

4 b Créer un index sur *borough*

```
> db.restaurants.createIndex( { "borough": 1 } );
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Compte rendu du TP3 NoSql

Appel à l'ordre

Le but de ce travail pratique est de plonger l'ensemble des étudiants à une **prise en main de robot3t** .L'accent est mis sur le fonctionnement du **mapReduce** et de la **jointure** avec mongoDB Ceci est réalisé à partir à **l'Université de Pau** et encadré par **Annig Le Parc LACAYRELLE** , maître de conférences au département informatique de l'Université.

1 Installation et démarrage de robot3t

1-a Installation

Les instructions contenu dans le fichier `InstallMongo.txt` issu du répertoire `install/NoSQL` sont d'excellent conducteurs.

Le principe est le même qu'avec mongoDB.
Il faut lancer le serveur avant de commencer à utiliser l'environnement

2 Exécuter ce script Que permet-il de calculer?

```
var mapFunction = function () {emit(this.borough, 1);};
var reduceFunction = function (key, values) {return Array.sum(values);};
var queryParam = {query : {}, out : "result_set"}
db.restaurants.mapReduce(mapFunction, reduceFunction, queryParam);
db.result_set.find();
```

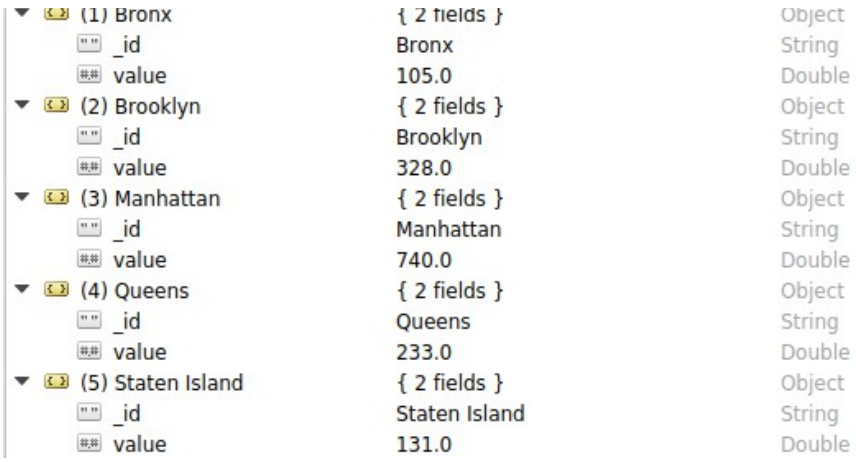
result_set 0 sec.

	_id	value
1	Bronx	2338.0
2	Brooklyn	6085.0
3	Manhattan	10258.0
4	Missing	51.0
5	Queens	5656.0
6	Staten Isl...	969.0

Ce script permet de calculer le nombre de restaurants par quartier.

3 Modifier ce script pour qu'il s'applique uniquement aux restaurants italiens

```
var mapFunction = function () {emit(this.borough, 1);};
var reduceFunction = function (key, values) {return Array.sum(values);}
var queryParam = {query : {"cuisine":/italian/i}, out : "result_set"}
db.restaurants.mapReduce(mapFunction, reduceFunction, queryParam);
db.result_set.find();
```



4 Proposer les scripts pour calculer

4-a La moyenne des notes pour chaque restaurant situé sur la 5ième avenue

```
var map = function () {
        for (var i=0; i<this.grades.length; i++){
                var key = this.name;
                var value = { compte : 1,moy:this.grades[i].score, score:this.grades[i].score};
                emit(key,value);
        }
}
```

```
var reduce = function (key,values) {
        reduction = {
                compte:0,
                score:0,
                moy:0
        };
        for (var i=0; i<values.length; i++){
                reduction.score += values[i].score;
                reduction.compte += values[i].compte;
                reduction.moy += reduction.score/reduction.compte;
        }
        return reduction;
};
```

```
var queryParam = {query : {"address.street":/5 Avenue/i}, out : "result_set"}

db.restaurants.mapReduce(
        map,
        reduce,
        queryParam
);
db.result_set.find();
```

```
/* 1 */
{
    "_id" : "15 St Cafe",
    "value" : {
        "compte" : 1.0,
        "moy" : 11.0,
        "score" : 11.0
    }
}

/* 2 */
{
    "_id" : "200 Fifth Avenue Restaurant & Sports Bar",
    "value" : {
        "compte" : 6.0,
        "score" : 119.0,
        "moy" : 119.033333333333
    }
}
```


4-b La moyenne des notes obtenues lors des évaluations A par type de cuisine

```
var map = function () {
        for (var i=0; i<this.grades.length; i++){
                var key = this.cuisine;
                var value = {compte : 1,moy:this.grades[i].score, score:this.grades[i].score}
                emit(key,value);
        }
}
```

```
var reduce = function (key,values) {
        reduction = {
                compte:0,
                score:0,
                moy:0
        };

        for (var i=0; i<values.length; i++){
                reduction.score += values[i].score;
                reduction.compte += values[i].compte;
                reduction.moy += reduction.score/reduction.compte;
        }
        return reduction;
};

var queryParam = {query : {"grades.grade":/A/i}, out : "result_set"}
```

```
var queryParam = {query : {"grades.grade":/A/i}, out : "result_set"}

db.restaurants.mapReduce(
       map,
       reduce,
       queryParam
);
db.result_set.find();
```

```
/* 1 */
{
    "_id" : "Afghan",
    "value" : {
        "compte" : 50.0,
        "score" : 563.0,
        "moy" : 129.304146311666
    }
}

/* 2 */
{
    "_id" : "African",
    "value" : {
        "compte" : 256.0,
        "score" : 3387.0,
        "moy" : 404.361646966535
    }
}
```

Compte rendu du TP4 NoSql

Appel à l'ordre

Le but de ce travail pratique est de plonger l'ensemble des étudiants à une **prise en main de Neo4j** .L'accent est mis sur le fonctionnement des bases de données orientées graphes dans la cadre de l'apprentissage de la gestion des données en masse que nous enseigne le module NoSQL.

Ce TP est réalisé à **l'Université de Pau** et encadré par **Annig Le Parc LACAYRELLE** , maître de conférences au département informatique de l'Université.

1 Installation et démarrage de Neo4j

Les instructions contenu dans le fichier InstallMongo.txt issu du répertoire install/NoSQL furent d'excellents conducteurs qui nous ont permis de prendre en main plus aisément l'outils informatique.

Neo4j se lance via une adresse url qu'on obtient en exécutant la commande *./neo4j console*

2 Documentation pour commencer

Neo4j permet de représenter les données en tant que nœuds reliés par un ensemble d'arcs, ces objets possédant leurs propres propriétés. Les propriétés sont constitués d'un couple de clé-valeurs de type simple tel que chaînes de caractères ou numérique; celles-ci peuvent êtres indexées.

Cypher est un langage informatique de requête orienté graphe utilisé par Neo4j.
On s'est référé pour l'ensemble du TP à la documentation de la page :
https://neo4j.com/docs/getting-started/current/graphdb-concepts/

3 Insertion des données

Nous avons créé une base de données ***MonReseauSocial*** qui contient les informations suivantes :

➔ Ajout de Alain , Florence , Kévin et les employeurs ou instituts auxquels ils y sont rattachés, dans notre base de données.

```
create (n:Personne:Homme{prenom:"Alain",âge:"55 ans"}) return n
create (n:Personne:Femme{prenom:"Florence",âge:35,specialite:"informatique"}) return n
create (n:Personne:Homme{prenom:"Kevin", âge:18}) return n
create (n:EntrepriseInstitut{nom:"Total",adresse:"Pau"}) return n
```

➔ Cette commande nous permet de visualiser notre base de données dans son

ensemble

```
$ match (n) return n
```

"n"
{"nom":"Alain","age":55}
{"nom":"Kevin","age":18}
{"nom":"Total","ville":"Pau"}
{"specialite":"jardinage","nom":"Bruno","age":46}
{"specialite":"informatique","nom":"Florence","age":55}
{"specialite":"informatique","nom":"Florence","age":55}
{"nom":"UPPA","ville":"Pau"}

4 Opérations sur les données

4-a Modifier l'âge d'une personne

➔ **On modifie l'âge de Kevin**

```
1 match(n{nom:"Kevin"})
2 SET n.age = 20;
```

4-b Ajouter un attribut travaille aux personnes. Cet attribut prend la valeur «oui» si la personne possède une relation TRAVAILLE-A et «non» sinon

➔ **On ajoute un attribut travaille à Alain et on fait de même avec Florence et Alain**

```
match(n{nom:"Alain"})
set n.travaille = "non"
return n
```

"n"
{"nom":"Alain","travaille":"non","age":55}

"n"
{"nom":"Alain","travaille":"non","age":55}
{"travaille":"oui","specialite":"informatique","nom":"Florence","age":55}
{"nom":"Total","ville":"Pau"}
{"travaille":"oui","specialite":"jardinage","nom":"Bruno","age":46}
{"nom":"UPPA","ville":"Pau"}
{"nom":"Kevin","travaille":"non","age":20}
{"travaille":"oui","specialite":"informatique","nom":"Florence","age":55}

5 Interrogation

5-a Afficher tous les travailleurs

```
match(n{travaille:"oui"})
return n.nom, n.travaille
```

"n.nom"	"n.travaille"
"Florence"	"oui"
"Bruno"	"oui"
"Florence"	"oui"

5-b Afficher tous les étudiants de la base

```
1 match(n)<-[:ETUDIE_A]-(e)
2 return e.nom
```

"e.nom"
"Kevin"

5-c Afficher les hommes de plus de 50 ans qui travaillent

- **On fait un premier filtre sur l'attribut “travaille”, on précise ensuite le sexe, puis on exige d’avoir des personnes de plus de 50 ans.**

```
1 MATCH(n{travaille:"oui"})
2 MATCH(n:Homme)
3 WHERE  n.age >= 50
4 RETURN n.age, n.nom, n.travaille
```

- **résultat non visible , j'insère alors un autre homme à la base tel que :**

```
$ create(n:Personne:Homme{nom:"Bastien",age:52, specialite : "petrole", travaille:"oui"}) return n
```

- **je relance ma commande et là j’obtiens le seule homme de la base qui**

a plus de 50 ans

"n.age"	"n.nom"	"n.travaille"
52	"Bastien"	"oui"

5-d Afficher les personnes qui occupent un emploi d'ingénieur et qui ont au moins 40 ans . Vous afficherez les personnes ainsi que les entreprises dans lesquelles elles travaillent **

```
MATCH (n)-[r{poste:"Ingenieur"}]->(p)
WHERE n.age >= 40
RETURN p.nom, n.nom
```

"n.nom"	"f.nom"
"Florence"	"Total"

5-e Afficher tous les amis de Bruno

création de relation pour faire fonctionner la requête

- J'ajoute le fait que Bruno et Bastien soit amis

```
MATCH (n{nom:"Bruno"})
MATCH (m{nom:"Bastien"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

- commande

```
MATCH (n)-[r:EST_AMI_AVEC]->(p{nom:"Bruno"})
RETURN n.nom
```

"n.nom"
"Bastien"

5-f Afficher la liste des personnes connectées à Bruno par au maximum 2 intermédiaires

création de relation pour faire fonctionner la requête

- **Je lie d'amitié Alain et Florence , Bruno et Kevin ainsi que Kevin et Bastien**

```
MATCH (n{nom:"Alain"})
MATCH (m{nom:"Florence"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Bruno"})
MATCH (m{nom:"Kevin"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Kevin"})
MATCH (m{nom:"Bastien"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

commande

```
1 MATCH (n:Personne)-[*1..2]->(p{nom:"Bruno"})
2 RETURN n.nom
```

"n.nom"
"Bastien"
"Kevin"

5-g Afficher le nom des personnes qui ont des amis en communs avec Bruno

=> création de relation pour faire fonctionner la requête

➔ **J'ajoute Evelyne à ma base**

```
$ create(n:Personne:Femme{nom:"Evelyne",age:38, specialite : "couture", travaille:"oui"}) return n
```

➔ **Je lie d'amitié Bruno et Evelyne**

```
1 MATCH (n{nom:"Bruno"})
2 MATCH (m{nom:"Evelyne"})
3 CREATE(m) - [:EST_AMI_AVEC] -> (n)
4
```

➔ **Puis Evelyne et Bastien**

```
MATCH (n{nom:"Bastien"})
MATCH (m{nom:"Evelyne"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

➔ **Bruno et Alain se lie d'amitié**

```
MATCH (n{nom:"Bruno"})
MATCH (m{nom:"Alain"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

➔ **Je lie ainsi Alain à Florence**

```
MATCH (n{nom:"Florence"})
MATCH (m{nom:"Alain"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

➔ **Je lie Florence à Bruno**

```
MATCH (n{nom:"Florence"})
MATCH (m{nom:"Bruno"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

➔ **Puis Florence à Kevin**

```
MATCH (n{nom:"Florence"})
MATCH (m{nom:"Kevin"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

=> commande

```
MATCH (n{nom:"Bruno"})-[:EST_AMI_AVEC]->(p)
MATCH (o)-[:EST_AMI_AVEC]->(p)
WHERE NOT o.nom = "Bruno"
RETURN o.nom
```

o.nom
"Kevin"
"Alain"

5-h Afficher le nom des personnes qui ont des amis en communs avec Bruno mais qui ne le connaissent pas

=> création de personnes pour faire fonctionner la requête

➔ **J'ajoute Marie**

```
$ create (n:Personne:Femme{nom:"Marie", âge:29, specialite:"coiffure" , travaille:"non"}) return n
```

n


```
{
  "travaille": "non",
  "âge": 29,
  "specialite": "coiffure",
  "nom": "Marie"
}
```


➔ **Liam**

```
$ create (n:Personne:Homme{nom:"Liam", âge:21, specialite:"athetisme" , travaille:"non"}) return n
```

n


```
{
  "travaille": "non",
  "âge": 21,
  "specialite": "athetisme",
  "nom": "Liam"
}
```


=> création de relations pour faire fonctionner la requête

➔ **Je crée d'autres liens d'amitiés**

```
1 MATCH (n{nom:"Liam"})
2 MATCH (m{nom:"Marie"})
3 CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Liam"})
MATCH (m{nom:"Kevin"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

=> commande

```
MATCH (n{nom:"Bruno"})-[:EST_AMI_AVEC]->(p)
MATCH (o)-[:EST_AMI_AVEC]->(p)
WHERE NOT o.nom = n.nom AND NOT (o)-[:EST_AMI_AVEC]->(n{nom:"Bruno"})
RETURN o.nom , n.nom, p.nom
```

o.nom	n.nom	p.nom
"Kevin"	"Bruno"	"Florence"

5-i Afficher le nombre de personnes liées à chaque institution (ou entreprise)

=> création de relations pour faire fonctionner la requête

➔ **J'ajoute à Evelyne et à Liam une formation**

```
match(n{nom:"Evelyne"})
match(f{nom:"UPPA"})
create(n)-[:ETUDIE_A{formation:"M1 TI"}]->(f)
```

```
match(n{nom:"Liam"})
match(f{nom:"UPPA"})
create(n)-[:ETUDIE_A{formation:"L3 AES"}]->(f)
```

=> commande

```
MATCH (n) - [:TRAVAILLE_A|ETUDIE_A] -> (p)
RETURN p.nom, COUNT(n)
```

p.nom	COUNT(n)
"Total"	4
"UPPA"	4

5-j Afficher le nombre de personnes liées à chaque institution (ou entreprise). Vos résultats devront être ordonnées selon le nombre de personnes liées (ordre croissant).

```
MATCH (n) - [:TRAVAILLE_A|ETUDIE_A] -> (p)
RETURN p.nom, COUNT(n)
ORDER BY COUNT(n)
```

5-k Afficher le nom des personnes qui ont plus de deux amis qui travaillent

=> création de personnes pour faire fonctionner la requête

➔ **J'ajoute Annick**

```
$ create (n:Personne:Femme{nom:"Annick", âge:26,
  specialite:"escalade", travaille:"oui" }) return n
```

```
{
  "travaille": "oui",
  "âge": 26,
  "specialite": "escalade",
  "nom": "Annick"
}
```


➔ **J'ajoute Pierre**

```
$ create (n:Personne:Homme{nom:"Pierre", âge:31,
  specialite:"natation", travaille:"non"}) return n
```


```
{
  "travaille": "non",
  "âge": 31,
  "specialite": "natation",
  "nom": "Pierre"
}
```


➔ **J'ajoute Ange**

```
$ create (n:Personne:Femme{nom:"Ange", âge:50,
  specialite:"science_de_la_terre", travaille:"oui"}) return n
```

```
{
  "travaille": "oui",
  "âge": 50,
  "specialite":
"science_de_la_terre",
  "nom": "Ange"
}
```


➔ **J'ajoute Mamadou**

```
$ create (n:Personne:Homme{nom:"Mamadou", âge:24,
  specialite:"biologie", travaille:"oui"}) return n
```


```
{
  "travaille": "oui",
  "âge": 24,
  "specialite": "biologie",
  "nom": "Mamadou"
}
```


➔ **J'ajoute une autre institution**

```
$ create(n:EntrepriseInstitut{nom:"Laboration d'analyse
médicale",ville:"Pau"}) return n
```

=> création de relation pour faire fonctionner la requête

➔ **Je renseigne le poste de Annick**

```
match(n{nom:"Annick"})
match(f{nom:"UPPA"})
create(n)-[:TRAVAILLE_A{poste:"Intervenante"}]->(f)
```

➔ **Celui de Ange**

```
match(n{nom:"Ange"})
match(f{nom:"Total"})
create(n)-[:TRAVAILLE_A{poste:"Chercheur"}]->(f)
```

➔ **Et celui de Mamadou**

```
match(n{nom:"Mamadou"})
match(f{nom:"Laboration d'analyse médicale"})
create(n)-[:TRAVAILLE_A{poste:"Laboratin"}]->(f)
```

➔ **Je crée de nouveaux liens d'amitié**

```
MATCH (n{nom:"Annick"})
MATCH (m{nom:"Florence"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Ange"})
MATCH (m{nom:"Florence"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Bastien"})
MATCH (m{nom:"Mamadou"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Annick"})
MATCH (m{nom:"Mamadou"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

```
MATCH (n{nom:"Florence"})
MATCH (m{nom:"Mamadou"})
CREATE(m) - [:EST_AMI_AVEC] -> (n)
```

=> commande

```
MATCH (n)-[r:EST_AMI_AVEC]->(p{travaille:"oui"})
WITH n, COUNT(p) AS amiTravail
WHERE amiTravail > 2
RETURN DISTINCT n.nom , amiTravail
```

n.nom	amiTravail
"Mamadou"	5

5-l Afficher les plus courts chemins connectant Bruno a des personnes ayant informatique comme spécialité.

```
1 MATCH plusCourt =allShortestPaths ((n)-[*]-(m) )
2 WHERE n.nom="Bruno" and m.specialite = "informatique"
3 RETURN plusCourt;
```

plusCourt


```
[
{
  "travaille": "oui",
  "specialite": "jardinage",
  "nom": "Bruno",
  "age": 46
}
,
{

}
```

,


```
{
  "travaille": "oui",
  "specialite": "informatique",
  "nom": "Florence",
  "age": 55
}
```


]

[


```
{
  "travaille": "oui",
  "specialite": "jardinage",
  "nom": "Bruno",
  "age": 46
}
```


}

,


```
{


}
```


,


```
{
  "travaille": "oui",
  "specialite": "informatique",
  "nom": "Florence",
  "age": 55
}
```


]

5-m Afficher le plan d'exécution pour la requête précédente

```
1 EXPLAIN MATCH plusCourt =allShortestPaths ((n)-[*]-(m) )
2 WHERE n.nom="Bruno" and m.specialite = "informatique"
3 RETURN plusCourt
```