

NoSQL Correction

1 Questions de cours (4points)

Vous expliquerez très brièvement (moins de 10 lignes) le système QNRW abordé lors du cours autour du système CouchDB (mais qui dépasse le cadre de ce système), en exploitant au besoin les notions suivantes :

- scalabilité
- réplication
- partitionnement
- postulat CAP et système distribué

un système QNRW est un système où Q représente le nombre de partition horizontale(shards) N qui représente les répliquions des données R : quorum de noeuds à consulter en lecture avant réponse au client W : quorum de noeuds à consulter en écriture avant réponse au client

Consistency (cohérence) : toute modification de donnée est suivie d'effet pour tous les noeuds du système

Availability (disponibilité) : toute requête émise et traitée par un noeud du système, reçoit une réponse (même en situation d'échec à produire une réponse)

Partition tolerance (recouvrement des noeuds) : assurer une continuité du fonctionnement en cas d'ajout/suppression de noeuds du système

2 Partie Neo4J (8 points)

2.1 Création du graphe

```
1 CREATE (louis:Personne:Homme {nom:'Claudel', prenom:'Louis', naissance:1826})
2 -[:MARIEA {annee_mariage:1866}]->
3 (louise:Personne:Femme {nom:'Cerveaux', prenom:'Louise', naissance:1840}),
```

```

4  (louis) <-[:A_PARENT]- ( amille:Personne:Femme {nom:'Caudel', prenom:'
    Camille', naissance:1864}),
5  ( amille) -[:A_PARENT]-> (louise), (louis) <-[:A_PARENT]-
6  (paul:Personne:Homme {nom:'Caudel', prenom:'Paul', naissance:1868})
7  RETURN louise, louis, amille, paul
8
9
10 MATCH (paul:Homme {nom:'Caudel', prenom:'Paul', naissance:1868})
11 CREATE (marie:Personne:Femme {nom:'Perrin', prenom:'Marie'}) -[:MARIE_A {
    annee_mariage:1906}]->
12 (paul),
13 (louise:Personne:Femme {nom:'Vet h', prenom:'Louise', naissance:1905}) -[:
    A_PARENT]->
14 (paul), (louise) -[:A_PARENT]->
15 (rosalie:Personne:Femme {nom:'Vet h', prenom:'Rosalie', naissance:1871}),
16 (henri:Personne:Homme {nom:'Vet h', prenom:'Henri', naissance:1898}) -[:
    A_PARENT]->
17 (rosalie), (pierre:Personne:Homme {nom:'Caudel', prenom:'Pierre',
    naissance:1908}) -[:A_PARENT]->
18 (paul), (pierre) -[:A_PARENT]-> (marie),
19 (reine:Personne:Femme {nom:'Paris', prenom:'Reine', naissance:1910})
20 -[:A_PARENT]-> (marie), (reine) -[:A_PARENT]-> (paul),
21 (reinemarie:Personne:Femme {nom:'Paris', prenom:'Reine-Marie'})
22 -[:A_PARENT]-> (reine), (renee:Personne:Femme {nom:'Nantet', prenom:'Renee',
    , naissance:1917})
23 -[:A_PARENT]-> (marie), (renee) -[:A_PARENT]-> (paul),
24 (victoire:Personne:Femme {nom:'Nantet', prenom:'Marie-victoire'}) -[:
    A_PARENT]-> (renee);
25 RETURN

```

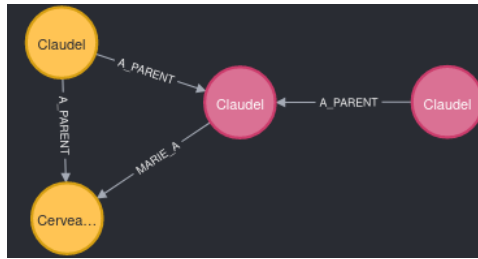


FIGURE 1 – requête 1



FIGURE 2 – requête Final

2) Vous écrirez en langage Cypher, la requête : "donner les petits-enfants de Paul Claudel"

```
1 MATCH (f2:Personne) -[:A_PARENT]-> (f1:Personne) -[:A_PARENT]-> (p:Personne {
    nom:"Claudel", prenom: "Paul" } )
2 RETURN f2.nom, f2.prenom
```

3) Une requête de consultation en langage Cypher, vous est donnée qui porte sur le graphe qui vient d'être réé. Vous donnerez la signification de cette requête, ainsi que le résultat renvoyé par cette requête

```
1 MATCH (cc:Femme {nom:'Claudel', prenom:'Camille'}) -[:A_PARENT]-> (pe:
    Personne) <-[:A_PARENT]-
2 (fs:Personne) <-[:A_PARENT]- (nn:Personne)
3 WHERE cc.prenom <> fs.prenom
4 RETURN *
```

4. Une nouvelle requête Cypher vous est donnée (toujours sur le graphe créé). Vous donnerez la signification de cette requête et et vous expliquerez le résultat obtenu (vous pouvez illustrer en dessinant le graphe associé).

```
:POST /rdf/cypher { "cypher":"MATCH (c:Personne {prenom:'Camille'}) RETURN c" , "format" : "N3"}

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix neovoc: <neo4j://vocabulary#> .
@prefix neoind: <neo4j://individuals#> .

neoind:1069 a neovoc:Femme, neovoc:Personne;
  neovoc:naissance "1864"^^<http://www.w3.org/2001/XMLSchema#long>;
  neovoc:nom "Caudel";
  neovoc:prenom "Camille" .
```

Cette request retourne les frères et soeur de Camille Claudel, les neveux et nièce de Camille Claudel, le père de Camille Claudel ainsi que camille Claudel elle même.

louis Claudel

Paul Claudel

Camille Claudel

Reine Paris

Renee Paris

Pierre Claudel

Louis vernhet

4) Cette requête Cypher retourne sous forme RDF une requête qui retourne toutes les personnes qui ont comme prénom "Camille"

Le résultats est affiché sous forme rdf.

neovoc :Femme et neovoc :Personne indique que l'objet retourner est d'objet Personne et femme.

neovoc :naissance "1864" indique que l'attribut naissance contient la valeur 1864

neovoc :nom "Caudel" indique que l'attribut nom contient la valeur "Caudel"

neovo :prenom "Camille" indique que l'attribut prenom contient la valeur "Camille"

3 Partie CouchDB

3.1

Vous donnerez votre perception sur l'organisation des documents retenue par le modélisateur. Quelle autre structuration vous aurait semblé pertinente ? Vous donnerez des exemples.

Le défaut que je trouve sur la structure du document est sur le stockage de la liste de salle. Il serait pour moi plus malin de faire un passage par référence en créant des objets de type salle. Exemple :

```
1
2 {
3   "_id" : "A36.03",
4   "type" : "amphi",
5   "capcite":120,
6   "accessibilite": "oui",
7   "video": "oui"
8
9 }
10
11 { "_id" : "triolet_36",
12   "ode" : "t_b36",
13   "ampus" : "Triolet",
14   "type": "batiment",
15   "destination" : "enseignement",
16   "annee_ onstru tion" : 2019,
17   "salles": [ "A36.03", "A36.02", "A36.01", "TD36.202", "SC36.04" ]
18 },
```

3.2

```
1
2 function(doc){
3   if(doc.type == 'batiment' && doc.destination == 'enseignement')
4     emit(doc.code, doc.annee_construction);
5 }
```

Parcroure tout les batiments ne selectionne que les ceux à destination de "enseignement".

Retourne son nom de code ainsi que son année de construction. Key | Value

t_b36 2019

t_b16 1966

t_b05 1964

3.3

```
1 function (doc) {  
2     if (doc.type == 'batiment' && doc.destination == 'enseignement') {  
3         if (Array.isArray(doc.salles)) {  
4             for (var s in doc.salles) {  
5                 emit([doc.code, doc.salles[s].type], doc.salles[s].capacite  
6             });  
7         }  
8     }  
9 }
```

parcroure tout les batiments a destination "enseignement" et parcourt chaque salle de chaque batiment. Retourne le nom de code du batiment, le type de salle ainsi que la capacité de la salle. Key | Value

[t_b36,amphi] 120

[t_b36,amphi] 120

[t_b36,amphi] 120

[t_b36,TDInfo] 40

[t_b36,amphi] 80

[t_b16,amphi] 120

[t_b16,TD] 18

etc..

3.4

```

1 function (doc) {
2     if (doc.type == 'batiment' && doc.destination == 'enseignement')
3     {
4         if (Array.isArray(doc.salles)){
5             for (var s in doc.salles)
6                 emit([doc.code, doc.salles[s].type], doc.salles[s].
7                     capaité);
8             }
9         }
10    }
11 function(keys, values, rereduce) {
12     return sum(values);
13 }

```

Fait la même chose que la requet précédente mais en additionnant a chaque fois les capacités des salles de classe. Key | Value

```

[t_b36,amphi] 120
[t_b36,amphi] 240
[t_b36,amphi] 360
[t_b36,TDInfo] 400
[t_b36,amphi] 480
[t_b16,amphi] 600
[t_b16,TD] 618
etc..

```