# 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 ou Q représente le nombre de partition horizontale(shards) N qui représente les réplications 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 estsuivie d'effet pour tous les nœuds du système

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

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

## 2 Partie Neo4J (8 points)

### 2.1 Création du graphe

```
1 CREATE (louis:Personne:Homme {nom:'Claudel', prenom:'Louis', naissance
      :1826})
2  -[:MARIE_A {annee_mariage:1866}]->
```

```
(louise:Personne:Femme {nom:'Cerveaux', prenom:'Louise', naissance:1840}),
(louis) <-[:A_PARENT]- ( amille:Personne:Femme {nom:'Claudel', prenom:'
    Camille', naissance:1864}),
( amille) -[:A_PARENT]-> (louise), (louis) <-[:A_PARENT]-
(paul:Personne:Homme {nom:'Claudel', prenom:'Paul', naissance:1868})
RETURN louise, louis, amille, paul


MATCH (paul:Homme {nom:'Claudel', prenom:'Paul', naissance:1868})
CREATE (marie:Personne:Femme {nom:'Perrin', prenom:'Marie'}) -[:MARIE_A {
    annee_mariage:1906}]->
(paul),
(louise:Personne:Femme {nom:'Vet h', prenom:'Louise', naissance:1905}) -[:
    A_PARENT]->
(paul), (louise) -[:A_PARENT]->
(rosalie:Personne:Femme {nom:'Vet h', prenom:'Rosalie', naissance:1871}),
(henri:Personne:Homme {nom:'Vet h', prenom:'Henri', naissance:1898}) -[:
    A_PARENT]->
(rosalie), (pierre:Personne:Homme {nom:'Claudel', prenom:'Pierre',
    naissance:1908}) -[:A_PARENT]->
(paul), (pierre) -[:A_PARENT]-> (marie),
(reine:Personne:Femme {nom:'Paris', prenom:'Reine', naissance:1910})
-[:A_PARENT]-> (marie), (reine) -[:A_PARENT]-> (paul),
(reinemarie:Personne:Femme {nom:'Paris', prenom:'Reine-Marie'})
-[:A_PARENT]-> (reine), (renee:Personne:Femme {nom:'Nantet', prenom:'Renee'
    , naissance:1917})
-[:A_PARENT]-> (marie), (renee) -[:A_PARENT]-> (paul),
(victoire:Personne:Femme {nom:'Nantet', prenom:'Marie-victoire'})-[:
    A_PARENT]-> (renee);
RETURN
```

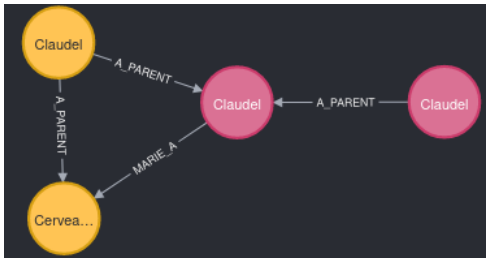


FIGURE 1 – requête 1

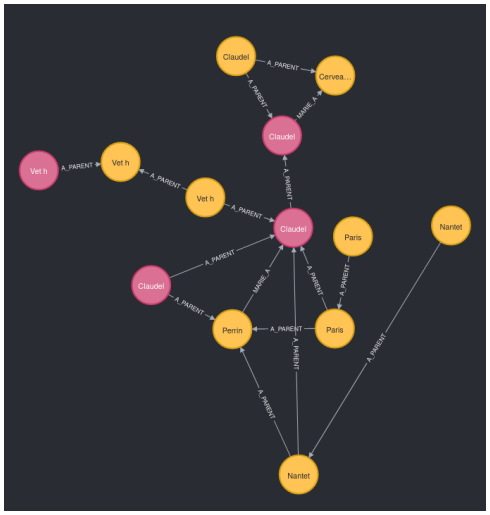


FIGURE 2 – requête Final

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

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

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

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

Cette request retourne les frères et soeur de Camille Claudel, les neuveux et niéce 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. 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 "Claudel";
  neovoc:prenom "Camille" .
```

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

Le résulats est affiché sous forme rdf.

neovoc :Femme et neovoc :Personne indique que l'objet retoruner est d'objet Personne et femme.
neovoc :naissance ”1864” indique que l'attribut naissance contient la valeur 1864
neovoc :nom ”Claudel” indique que l'attribut nom contient la valeur ”Claudel”
neovo :prenom ”Camille” indique que l'attribut prenom contient la valeur ”Camille”

# 3 Partie CouchDB

## 3.1

Vous donnerez votre per eption sur l'organisation des do uments retenue par le modélisateur. Quelle autre stru turation vous aurait semblé pertinente ? Vous donnerez des exemples.

Le défault 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 objet de type salle. Exemple :

```
1
```

```
{
    "_id" : "A36.03",
    "type" : "amphi",
    "capcite":120,
    "accessibilite": "oui",
    "video": "oui"

}

{ "_id" :"triolet_36",
" ode" : "t_b36",
" ampus" : "Triolet",
"type": "batiment",
"destination" : "enseignement",
"annee_ onstru tion" : 2019,
"salles": ["A36.03","A36.02","A36.01","TD36.202","SC36.04"]
},
```

### 3.2

```
function(doc){
    if(doc.type =='batiment' && doc.destination =='enseignement')
        emit(doc.code, doc.annee_construtions);
}
```

Parcrourt 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

```
function (doc) {
    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   }
```

parcrourt 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], do.salles[s].
    capaite);
7               }
8           }
9   }
10
11  function(keys, values, reredue) {
12      return sum(values);
13  }
```

Fait la même chose que la requet précédente mais en addittionnant 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..