TD06

Base de données non traditionnelles

1 Génération de données d'index

1.1Proposer la modélisation de la collection index en écrivant un fragment de JSON

aire 1044 1063 2078 2520

aires 262 263 264 267 268 273 277 279 280 281 282

airiau 906 2095 2096 2097

ais 793 805 1527 2053 2711 3488 3612

aisance 3269

aise 198 205 206 233 234 235 236 237 238 240

aisé 1333 1334 1606 1607

aisée 595 597 3419

aisément 152 755 1766 1930

Quel modèle se dégage de cet extrait ?

1.1Proposer la modélisation de la collection index en écrivant un fragment de JSON

```
[
   { mot,
     index  [ ]
   },
   { mot,
     index  [ ]
   },
   ...
   { mot,
     index  [ ]
   }
]
```

aire 1044 1063 2078 2520

aires 262 263 264 267 268 273 277 279 280 281 282

airiau 906 2095 2096 2097

ais 793 805 1527 2053 2711 3488 3612

aisance 3269

aise 198 205 206 233 234 235 236 237 238 240

aisé 1333 1334 1606 1607

aisée 595 597 3419

aisément 152 755 1766 1930

1 Génération de données d'index

```
[
   { mot,
     index  [ ]
   },
   { mot,
     index  [ ]
   },
   ...
   { mot,
     index  [ ]
   }
]
```

aire 1044 1063 2078 2520

aires 262 263 264 267 268 273 277 279 280 281 282

airiau 906 2095 2096 2097

ais 793 805 1527 2053 2711 3488 3612

aisance 3269

aise 198 205 206 233 234 235 236 237 238 240

aisé 1333 1334 1606 1607

aisée 595 597 3419

aisément 152 755 1766 1930

La donnée au format JSON

```
[
  [
    "aire",
    "1044",
    "1063",
    "2078",
    "2520"
  ],
…
  [
    "aisément",
    "152",
    "755",
    "1766",
    "1930"
  ]
]
```

On reprend l'exemple introduit précédemment :

1. la réparation 1 utilise un filtre à air et 5l d'huile.
2. la réparation 2 utilise 4 pneus.

Un de vos camarades, débutant en bases de données traditionnelles, propose la structuration suivante pour les réparations :

```
[
  {
    "reparation": 1,
    "pieces": [
      {
        "filtreAir": 1
      },
      {
        "huile1litre": 5
      }
    ]
  },
  {
    "reparation": 2,
    "pieces": [
      {
        "pneu": 4
      }
    ]
  }
]
```

Cette modélisation vous semble-elle correcte ?
Pourquoi ?
Si non, proposez une modélisation correcte

On reprend l'exemple introduit précédemment :

1. la réparation 1 utilise un filtre à air et 5l d'huile.
2. la réparation 2 utilise 4 pneus.

Un de vos camarades, débutant en bases de données traditionnelles, propose la structuration suivante pour les réparations :

```
[
  {
    "reparation": 1,
    "pieces": [
      {
        "filtreAir": 1
      },
      {
        "huile1litre": 5
      }
    ]
  },
  {
    "reparation": 2,
    "pieces": [
      {
        "pneu": 4
      }
    ]
  }
]
```

Pas d'information dans la clé.
L'information est à mettre dans la valeur

```
[
 { ‘‘répartition’’:1,
      ’’pièces’’ : [
      {
       ‘‘nom’’ :’’filtreAir’’,
      ’’Qte’’ : ‘‘1’’
      },
      {
       ‘‘nom’’ :’’huiler’’,
      ’’Qte’’ : ‘‘5’’
      }
      ]
 },
 {’’répartition’’:2,
      ’’pièces’’ : [
      {
       ‘‘nom’’ :’’pneu’’,
      ’’Qte’’ : ‘‘4’’
      }
      ]
 }
]
```


1.2 Proposer un script jq permettant de transformer les données du fichier texte en BSON que l'on peut insérer dans MongoDB

- Avant d'injecter un fichier data MongoDB, il faut s'assurer qu'il est bien indenté
- MongoDB :
  - Demande en entrée un BSON = Binary JSON
  - L'enregistrement dans un fichier est dit DOCUMENT
  - La BDD n'est pas constituée de tables mais de collections
- On va utiliser l'utilitaire jq
  - Est un est un outil en ligne de commande pour manipuler du JSON
  - Syntaxe : $ jq ‘ requete ‘ nomFichiersurlequelAppliquerReq.json
    Il prend une entrée et produit une sortie
  - https://stedolan.github.io/jq/manual/

1.2 Accès aux données selon des sélecteurs

- $ jq . immo.json
  On vérifie que fichier immo.json est bien indenté
  Le . ici indique la racine de la donnée
- $ jq .[ ] immo.json
  Demande l'ensemble des documents dans imm.json
  Dans les deux cas : Si le fichier n'est pas au format JSON alors erreur
- Ces deux commandes ne font aucune sélection, elles ne font qu'afficher l'entrée. Il faut ajouter un sélecteur pour effectuer une sélection

1.2 Accès aux données selon des sélecteurs

- $ jq .[1] immo.json
  Demande le document d'index 1 dans immo.json (le 1er index est 0)
  Ici le . Pointe sur l'élément actuellement parcourue
- $ jq .[2,4] immo.json
  Demande les documents d'index 2 et 4 dans imm.json
  Si on demande l'index k et qu'il n'existe pas renvoie null
  Pour ne pas se tromper vérifier le nombre de documents en mode compact
  $ jq -c .[ ] data1.json | wc -l on obtient le nombre de ligne

1.2 Accès aux données selon des sélecteurs

- $ jq .[ ].id immo.json
  - Demande la valeur de la propriété id dans les documents de imm.json
  - si id n'existe pas ou mal hiérarchisé retourne null
  - $ jq 'map(has("isActive"))' data1.json renvoie true
  - $ jq 'map(has("foo"))' data1.json renvoie false

1.2 Accès aux données selon des sélecteurs


```
[{
    "id": "600ab7774fe33cf22a658290",
    "isActive": false,
    "picture": "http://placehold.it/32x32",
    "title": "do pariatur officia esse nulla",
    "price": 742407,
    "owner": {
        "firstName": "Parsons",
        "lastName": "Hubbard",
        "email": "parsons.hubbard@test.me",
        "phone": "+1 (898) 482-2191"
    },
    "address": "242 Monitor Street, Floriston, Northern Mariana Islands, 2615",
    "informations": "Laborum esse do deserunt eu ipsum laborum et veniam dolor elit et aute. Cupidatat ad elit do
}]
```


$ jq .[].id data1.json

Renvoi :

1.2 Accès aux données selon des sélecteurs


```
[{
    "id": "600ab7774fe33cf22a658290",
    "isActive": false,
    "picture": "http://placehold.it/32x32",
    "title": "do pariatur officia esse nulla",
    "price": 742407,
    "owner": {
        "firstName": "Parsons",
        "lastName": "Hubbard",
        "email": "parsons.hubbard@test.me",
        "phone": "+1 (898) 482-2191"
    },
    "address": "242 Monitor Street, Floriston, Northern Mariana Islands, 2615",
    "informations": "Laborum esse do deserunt eu ipsum laborum et veniam dolor elit et aute. Cupidatat ad elit do
}]
```


$ jq .[].title data1.json

Renvoi :

1.2 Accès aux données selon des sélecteurs


```
[{
    "id": "600ab7774fe33cf22a658290",
    "isActive": false,
    "picture": "http://placehold.it/32x32",
    "title": "do pariatur officia esse nulla",
    "price": 742407,
    "owner": {
        "firstName": "Parsons",
        "lastName": "Hubbard",
        "email": "parsons.hubbard@test.me",
        "phone": "+1 (898) 482-2191"
    },
    "address": "242 Monitor Street, Floriston, Northern Mariana Islands, 2615",
    "informations": "Laborum esse do deserunt eu ipsum laborum et veniam dolor elit et aute. Cupidatat ad elit do
}
]
```


$ jq .[].firstName data1.json

Renvoi :

- Voir image 01,,,,

1.2 Accès aux données selon des sélecteurs

- $ jq .[].owner immo.json
  - Demande la valeur de la propriété owner dans les documents de imm.json
  - Owner est lui même constitué de plusieurs propriétés
- $ jq -c .[].owner immo.json
  - La version compacté sur une ligne de la précédente commande
- $ jq .[].owner.firstName immo.json
  - Demande la valeur de la propriété owner.firstName dans les documents de imm.json

1.2 Accès aux données selon des sélecteurs

- jq utilise le pipe | pour connecter plusieurs opérations ensemble
- jq répétera le filtre pour chaque objet JSON fourni par l'étape précédente.
- jq <filter> [files…]
  - Le filtre doit être entouré de guillemets simples

1.2 Accès aux données selon des sélecteurs

- Avec jq, on peut enchaîner et imbriquer les filtres :
- $ jq '.[ ] | select(.owner.firstName == "Kirk")' immo.json
  - Revoie tout le contenu des documents dont owner.firstName est "Kirk"
- $ jq '.[ ] | select(.owner.firstName == "Kirk") | {id, price}' immo.json
  - Revoie la valeur des propriétés id et price des documents dont owner.firstName est "Kirk"

1.2 Accès aux données selon des sélecteurs

- Voir image 02,,,, :
  - $ jq '.[ ] | select(.owner.firstName == "Kirk")' immo.json
  - $ jq '.[ ] | select(.owner.firstName == "Kirk") | {id, price}' immo.json

1.2 Accès aux données selon des sélecteurs

- Avec jq, on peut enchaîner et imbriquer les filtres :
- $ jq '.[ ] | select(.owner.firstName | test("^Jo")) | {owner}' immo.json
  - Revoie tout le contenu de owner des documents dont owner.firstName commence par un "Jo"
- $ jq '.[1] | keys' immo.json
  - Renvoie l’ensemble des clés du document d’index 1

1.2 Accès aux données selon des sélecteurs

- Voir images 03 & 04 :
- $ jq '.[ ] | select(.owner.firstName | test("^Jo")) | {owner}' immo.json
- $ jq '.[1] | keys' immo.json

1.2 Accès aux données selon des sélecteurs

- Analysons la progression des instructions suivantes :

1) jq ' ' index.test

Équivalent à jq . index.test

Erreur si ce n’est pas du JSON

Attention dans les commandes suivantes le fichier manipulé est nécessairement NON JSON

2) jq --raw-input ' ' index.test

Équivalent à 1) l’option **--raw-input** considère que l'entrée est au format quelconque et pas nécessairement JSON. Chaque ligne de l’entrée est transmise au filtre sous forme de chaîne

Voire image 05,,,,,

1.2 Analysons la progression des instructions suivantes

3) jq --slurp --raw-input ' ' index.test

Équivalent à 2 l'option **--slurp** considère le flux d'entrée comme une unique chaîne de caractère et exécute le filtre une fois

4) jq --slurp --raw-input 'split("\n")' index.test

Améliore la 3) avec le filtre **split("\n")** : génère un tableau en découpant la chaîne de caractères selon \n

On s'approche du JSON

Voire image 06,,,,,

1.2 Analysons la progression des instructions suivantes

5) jq --slurp --raw-input 'split("\n") | map(split(" "))' index.test

Le filtre **map(x)** applique le filtre à chaque élément et renverra les sorties dans un nouveau tableau (dans cet exemple la sortie est du JSON)

6) jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1]})' index.test ***genere la sortie ok data0

7) jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:]})' index.test

8) jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:] | map(tonumber)})' index.test

Le filtre **tonumber** transforme une chaîne en nombre

1.2 Analysons la progression des instructions suivantes

9) jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:] | map(tonumber)}) | .[]' index.test

10) jq -c --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:] | map(tonumber)}) | .[ ]' index.test

2 Transformation des textes en JSON
2.2.1 Commencer par transformer les textes en JSON

- $ cat 96

  Join the "Graphemics in the 21st century" conference in June 2018!

  http://conferences.telecom-bretagne.eu/grafematik/

  ICBM address: 48°21'31.57"N 4°34'16.76"W
- $ pageId=96

2 Transformation des textes en JSON
2.2.1 Commencer par transformer les textes en JSON

- $ jq --raw-input --slurp '' $pageId

  "Join the “Graphemics in the 21st century” conference in June 2018!\nhttp://conferences.telecom-bretagne.eu/grafematik/\nICBM address: 48°21'31.57\"N 4°34'16.76\"W\n\n"
- $ jq --raw-input --slurp '{"_id":1, "text":.}' $pageId

  {

  "_id": 1,

  "text": "Join the “Graphemics in the 21st century” conference in June 2018!\nhttp://conferences.telecom-bretagne.eu/grafematik/\nICBM address: 48°21'31.57\"N 4°34'16.76\"W\n\n"

  }
- $ jq --raw-input --slurp --arg pageId $pageId '{"_id": $pageId, "text":.}' $pageId

  {

  "_id": "96",

  "text": "Join the “Graphemics in the 21st century” conference in June 2018!\nhttp://conferences.telecom-bretagne.eu/grafematik/\nICBM address: 48°21'31.57\"N 4°34'16.76\"W\n\n"

  }
- Voir image 07,,,,

2 Transformation des textes en JSON
2.2.2 Transformons les tableaux de (mot, tfidf) en JSON

- head $pageId.tfidf | jq --raw-input --slurp ''
- head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n")}'
- head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) }'
- head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) | map({"word": .[0], "tfidf": .[1]})}'
- head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) | map({"word": .[0], "tfidf": .[1]})}'
- head $pageId.tfidf | head -c -1 | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) | map({"word": .[0], "tfidf": .[1]})}'
- Voir fichier texte 01,,,,

2.3 Importation des données
TP6.1

- Syntaxe : $ mongoimport -d database_name -c collection_name --file collection_name.json

  $ jq -c .[ ] immo.json | mongoimport -d immo -c biens

  Renvoie :

  connected to: mongodb://localhost/

  11 document(s) imported successfully. 0 document(s) failed to import.
- Se connecter avec

  $ mongo

  >

2.3 Importation des données
TP6.1

- Lister les DB disponibles avec

  > show databases

  admin 0.000GB

  immo 0.000GB
- Avant de requêter il faut accéder à la DB avec

  > use immo

  > show collections
- Requêtes :

  > db.immo.find()

  > db.immo.findOne()

  > ...

2.3 Importation des données
TP 6.2

- Syntaxe : $ mongoimport -d database_name -c collection_name --file collection_name.json
- $ cat mongo/pages.bson | mongoimport -d engine -c pages
  Renvoie :
  connected to: mongodb://localhost/
  3655 document(s) imported successfully. 0 document(s) failed to import.
- $ cat mongo/terms.bson | mongoimport -d engine -c terms
- Se connecter avec
  $ mongo
  >

2.3 Importation des données

- Lister les DB disponibles avec
  > show databases
  admin 0.000GB
  cfp 0.000GB
  config 0.000GB
  engine 0.028GB
  local 0.000GB
- Avant de requêter il faut accéder à la DB avec
  > use engine

2.3 Requêter
avec la DB engine

> db.terms.find({_id:"abcd"})
renvoie { "_id" : "abcd", "index" : [3127, 3136, 3578] }

> db.pages.find({"words.word":"abcd"} , {_id:1})
renvoie { "_id" : "3127" }
{ "_id" : "3136" }
{ "_id" : "3578" }

3 Déconnexion

- Quitter mongo
  > exit ou ^C
  bye
  $