TP n°1, Exemples de systèmes NoSQL : clé-valeur (REDIS) et documentaires (MongoDB)

Partie 1 : Utilisation de Redis :

Mise en place de Redis dans docker et ouverture du logiciel :

Afin de mettre en place Redis sur une machine, il faut dans un premier temps installer docker. Pour se faire, il suffit d'aller sur le site, de télécharger l'installer adapté à votre système d'exploitation puis de l'exécuter.

Une fois docker installé, ouvrez un terminal et exécutez la commande suivante afin de lancer Redis dans le container name : `docker run –name name redis`
Dans le cas où vous n'avez pas d'image pour Redis sur votre machine, cette commande va télécharger une image, l'installer et l'exécuter.
Une fois cette commande exécutée, laissez Redis tourner et ouvrez un autre terminal.

Dans ce nouveau terminal, exécutez les commandes suivantes :

- Afin de voir les container présents dans docker : `docker ps`
- Afin d'exécuter une commande précise dans un container donné. Ici on souhaite ouvrir un terminal dans Redis : `docker exec -it loving_cori redis-cli`. *Docker exec* permet d'exécuter la commande, *-it* permet d'activer le mode itératif (-i), ce qui permet de garder le terminal actif et d'allouer un pseudo terminal (-t). *loving-cori* est le nom du container. Enfin, redis-cli ouvre un terminal dans Redis.

Une fois tout cela actif, nous pouvons commencer les manipulations dans Redis.

Manipulations des clés et valeurs :

Définir un couple clé valeur, ici clé = demo et valeur = Bonjour. Renvoie OK car bonne exécution :

```
127.0.0.1:6379> SET demo "Bonjour"
OK
```

Ensemble des opération existante sous Redis (ce ne sont pas les commandes) :CREATE, READ, UPDATE et DELETE (CRUD)

Création d'un autre couple :

```
127.0.0.1:6379> SET user:1234 "Arthur"
OK
```

Récupération de la valeur associée à une clé (ici la clé user:1234) :

```
127.0.0.1:6379> GET user:1234
"Arthur"
```

Suppression d'un couple via la clé, la commande renvoie 1 car la suppression fonctionne :

```
127.0.0.1:6379> del user:1234
```

```
(integer) 1
```

On tente de supprimer le même couple, 0 signifie que l'action échoue :

```
127.0.0.1:6379> del user:1234
(integer) 0
```

Il est possible d'incrémenter et de décrémenter des valeurs :

```
127.0.0.1:6379> SET 27novembre 0
OK
127.0.0.1:6379> incr 27novembre
(integer) 1
127.0.0.1:6379> incr 27novembre
(integer) 2
127.0.0.1:6379> decr 27novembre
(integer) 1
```

Il est possible de vérifier le temps de vie d'une clé, ici -1 signifie que la durée de vie est infinie :

```
127.0.0.1:6379> SET macle mavaleur
OK
127.0.0.1:6379> ttl macle
(integer) -1
```

On peut bien sûr définir ce temps de vie :

```
127.0.0.1:6379> expire macle 120
(integer) 1
127.0.0.1:6379> ttl macle
(integer) 118
```

Manipulations des structures de données :

Exemple avec liste :

Création d'une liste :

```
127.0.0.1:6379> RPUSH mesCours "BDA"
(integer) 1
```

Ajouter un élément (2 indique le nombre d'élément de la liste) :

```
127.0.0.1:6379> RPUSH mesCours "Service Web"
(integer) 2
```

On ne peut pas afficher les élémenrs avec GET :

```
127.0.0.1:6379> GET mesCours
(error) WRONGTYPE Operation against a key holding the wrong kind of value
```

Il faut passer par LRANGE. Ici, 0 indique que l'on affiche les éléments à partir du premier élément et -1 que l'on veut afficher TOUS les éléments :

```
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "BDA"
2) "Service Web"
```

On veut afficher uniquement le premier élément :

```
127.0.0.1:6379> LRANGE mesCours 0 0
1) "BDA"
```

On veut afficher le second élement :

```
127.0.0.1:6379> LRANGE mesCours 1 1
1) "Service Web"
```

Suppression d'un élément :

```
127.0.0.1:6379> LPOP mesCours
"BDA"
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "Service Web"
```

On peut ajouter plusieurs fois le même élément :

```
127.0.0.1:6379> RPUSH mesCours "Service Web"
(integer) 2
127.0.0.1:6379> RPUSH mesCours "Service Web"
(integer) 3
127.0.0.1:6379> RPUSH mesCours "Service Web"
(integer) 4
127.0.0.1:6379> RPUSH mesCours "Service Web"
(integer) 5
```

Exemple avec sets (pas de doublons) :

Création d'un set :

```
127.0.0.1:6379> SADD utilisateurs "FZ"
(integer) 1
```

Ajout d'éléments :

```
127.0.0.1:6379> SADD utilisateurs "Catherine"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Ethan"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Dima"
(integer) 1
```

On ne peut pas ajouter plusieurs fois le même élément :

```
127.0.0.1:6379> SADD utilisateurs "Catherine"
(integer) 0
```

Affichages des éléments de la liste :

```
127.0.0.1:6379> SMEMBERS utilisateurs
1) "FZ"
2) "Catherine"
3) "Ethan"
4) "Dima"
```

Suppression (1 car trouvé et supprimé, 0 car déja supprimé) :

```
127.0.0.1:6379> SREM utilisateurs "Dima"
(integer) 1
127.0.0.1:6379> SREM utilisateurs "Dima"
(integer) 0
```

Fusion de deux SET :

```
127.0.0.1:6379> SADD autresutilisateurs "Gerard"
(integer) 1
127.0.0.1:6379> SADD autresutilisateurs "Gilbert"
(integer) 1
127.0.0.1:6379> SUNION utilisateurs autresutilisateurs
1) "Gerard"
2) "Catherine"
3) "FZ"
4) "Gilbert"
5) "Ethan"
```

Afin d'avoir l'ensemble des commandes REDIS existantes, il existe une documentation Redis.

Exemple avec sets ordonnés :

Création :

```
127.0.0.1:6379> ZADD score4 19 "FZ"
(integer) 1
```

Ajout d'élément :

```
127.0.0.1:6379> ZADD score4 89 "Cath"
(integer) 1
127.0.0.1:6379> ZADD score4 2 "Dima"
(integer) 1
```

Affichage des éléments dans l'ordre croissant des valeurs :

```
127.0.0.1:6379> ZRANGE score4 0 -1
1) "Dima"
2) "FZ"
3) "Cath"
```

Affichage des éléments dans l'ordre décroissant des valeurs :

```
127.0.0.1:6379> ZREVRANGE score4 0 -1
1) "Cath"
2) "FZ"
3) "Dima"
```

Connaître l'indice d'un élément :

```
127.0.0.1:6379> ZRANK score4 "FZ"
(integer) 1
127.0.0.1:6379> ZRANK score4 "Dima"
(integer) 0
127.0.0.1:6379> ZRANK score4 "Catherine"
(nil)
127.0.0.1:6379> ZRANK score4 "Cath"
(integer) 2
```

Exemple avec les hash :

Création d'un hset et remplissage :

```
127.0.0.1:6379>  HSET user:11 username "abassette"
(integer) 1
127.0.0.1:6379>  HSET user:11 age 23
(integer) 1
127.0.0.1:6379>  HSET user:11 mail a.bassette@uspn.fr
(integer) 1
```

Affichage de tout le hset :

```
127.0.0.1:6379> HGETALL user:11
1) "username"
2) "abassette"
3) "age"
4) "23"
5) "mail"
6) "a.bassette@uspn.fr"
```

Création d'un hset en une seule ligne :

```
127.0.0.1:6379> HMSET suer:21 username "FZ" age 21 email fz.elyounoussi@uspn.fr
OK
127.0.0.1:6379> HGETALL suer:21
1) "username"
2) "FZ"
3) "age"
4) "21"
5) "email"
6) "fz.elyounoussi@uspn.fr"
```

Incrémentation de l'âge d'un utilisateur :

```
127.0.0.1:6379> HINCRBY suer:21 age 2
```

```
(integer) 23
```

Connaître les valeurs des clés dans le hset :

```
127.0.0.1:6379> HVALS suer:21
1) "FZ"
2) "23"
3) "fz.elyounoussi@uspn.fr"
```

Autres commandes :

Echange de messages via abonnement :

S’abonner à un canal et recevoir des messages :

```
127.0.0.1:6379(subscribed mode)> SUBSCRIBE mesCours user:1
1) "message"
2) "mesCours"
3) "nouveau cours"
1) "subscribe"
2) "mesCours"
3) (integer) 2
1) "subscribe"
2) "user:1"
3) (integer) 2
1) "message"
2) "mesCours"
3) "nouveau cours"
1) "message"
2) "user:1"
3) "Hello"
```

S’abonner à des canaux en suivant un pattern :

```
127.0.0.1:6379(subscribed mode)> PSUBSCRIBE mes*
1) "psubscribe"
2) "mes*"
3) (integer) 3
1) "message"
2) "mesCours"
3) "nouveau cours"
1) "pmessage"
2) "mes*"
3) "mesCours"
4) "nouveau cours"
1) "pmessage"
2) "mes*"
3) "mesNotes"
4) "Note disponible"
```

Envoie de message dans un canal :

```
127.0.0.1:6379> PUBLISH mesCours "nouveau cours"
(integer) 1
127.0.0.1:6379> PUBLISH mesCours "nouveau cours"
(integer) 1
```

Envoie de message à un utilisateur :

```
127.0.0.1:6379> PUBLISH user:1 "Hello"
(integer) 1
```

Messages envoyés dans les canaux :

```
127.0.0.1:6379> PUBLISH mesCours "nouveau cours"
(integer) 2
127.0.0.1:6379> PUBLISH mesNotes "Note disponible"
(integer) 1
```

Changement de base de données et vérification des clés :

Accéder à la base de données numéro 1 :

```
127.0.0.1:6379> SELECT 1
OK
```

Les clés définies dans cette base (aucune car jamais utilisée)

```
127.0.0.1:6379[1]> KEYS *
(empty array)
```

Retour sur la base de donnée 0 :

```
127.0.0.1:6379[1]> SELECT 0
OK
```

Vérification des clés de la base de donnée 0 (ce sont les clés définies ci-dessus) :

```
127.0.0.1:6379> KEYS *
1) "suer:21"
2) "27novembre"
3) "score4"
4) "utilisateurs"
5) "user:11"
6) "mesCours"
7) "demo"
8) "autresutilisateurs"
```

Partie 2 : Utilisation de MongoDB :

Mise en place de MongoDB dans docker, ouverture du logiciel, accès à l'archive et utilisation du bon sample :

Installation de téléchargement et installation de MongoDB dans un container :
```
docker run --name MongoMongo mongo
```

Copie du sample dans le container :
```
docker cp sampledata.archive MongoMongo:/
```

Connexion au terminal de MongoDB :
```
docker exec -it MongoMongo bash
```

Restauration de l'archive :
```
mongorestore --archive=sampledata.archive
```

Connexion à MongoDB :
```
mongosh
```

Affichage des bases de données disponibles :
```
show bds
```

Utilisation de la base de données sample_mflix :
```
use sample_mflix
```

Voir les collections de la base :
```
sample_mflix> show collections
```

Filtrer et projeter les données :

Affichage des 5 films sortis depuis 2015
```
sample_mflix> db.movies.find({ year: { $gte: 2015 } }).limit(5)
```

```
[
  {
    _id: ObjectId('573a13adf29313caabd2b765'),
    plot: "A new theme park is built on the original site of Jurassic Park. Everything is going well until the park's newest attraction--a genetically modified giant stealth killing machine--escapes containment and goes on a killing spree.",
    genres: [ 'Action', 'Adventure', 'Sci-Fi' ],
    runtime: 124,
    metacritic: 59,
    rated: 'PG-13',
    cast: [
      'Chris Pratt',
      'Bryce Dallas Howard',
```

```
    'Irrfan Khan',
    "Vincent D'Onofrio"
  ],
  num_mflix_comments: 0,
  poster:
'https://m.media-amazon.com/images/M/MV5BNzQ3OTY4NjAtNzM5OS00N2ZhLWJlOWUtY
zYwZjNmOWRiMzcyXkEyXkFqcGdeQXVyMTMxODk2OTU@._V1_SY1000_SX677_AL_.jpg
',
  title: 'Jurassic World',
  fullplot: '22 years after the original Jurassic Park failed, the new park (also known as
Jurassic World) is open for business. After years of studying genetics the scientists on the
park genetically engineer a new breed of dinosaur. When everything goes horribly wrong,
will our heroes make it off the island?',
  languages: [ 'English' ],
  released: ISODate('2015-06-12T00:00:00.000Z'),
  directors: [ 'Colin Trevorrow' ],
  writers: [
    'Rick Jaffa (screenplay)',
    'Amanda Silver (screenplay)',
    'Colin Trevorrow (screenplay)',
    'Derek Connolly (screenplay)',
    'Rick Jaffa (story)',
    'Amanda Silver (story)',
    'Michael Crichton (characters)'
  ],
  awards: { wins: 0, nominations: 5, text: '5 nominations.' },
  lastupdated: '2015-09-11 00:21:41.630000000',
  year: 2015,
  imdb: { rating: 7.3, votes: 252556, id: 369610 },
  countries: [ 'USA', 'China' ],
  type: 'movie'
 },
```

Affichage des films donc le genre est comédie :

```
db.movies.find({genres: "Comedy"})
```

Affichage des films sortis entre 2000 et 2005 :

```
db.movies.find({year: {$gte: 2000, $lte: 2005}}, {title: 1, year: 1}).pretty()
{
 _id: ObjectId('573a139af29313caabcf0f58'),
 year: 2000,
 title: "The Emperor's New Groove"
}
```

Affichage des films de genres drama et romance :

```
db.movies.find({genres: {$all: ["Drama", "Romance"]}}, {title: 1, genres: 1})
{
```

```
  _id: ObjectId('573a1392f29313caabcd9c5d'),
  genres: [
    'Drama',
    'Romance'
  ],
  title: 'Red Dust'
}
```

Affichage des films sans champs rate :

```
db.movies.find({rated: {$exists: false}}, {title: 1})
{
  _id: ObjectId('573a1391f29313caabcd7472'),
  title: 'Foolish Wives'
}
```

En résumé, pour afficher les données une requête est construire de la manière suivant :

db.nom_collection.find({nom_champs : {condition}} , {champ_a_afficher : 1})

Agrégation :

Affichage du nombre de films par année :

```
b.movies.aggregate([ {$group: {_id: "$year", total: {$sum: 1}}}, {$sort: {_id: 1}} ])
```

Affichage de la moyenne des notes IMDb par genre :

```
db.movies.aggregate([ {$unwind: "$genres"}, {$group: {_id: "$genres", moyenne: {$avg: "$imdb.rating"}}}, {$sort: {moyenne: -1}} ])
```

Affichage du nombre de films par pays :

```
db.movies.aggregate([ {$unwind: "$countries"}, {$group: {_id: "$countries", total: {$sum: 1}}}, {$sort: {total: -1}} ])
```

Affichage du top Top 5 réalisateurs :

```
db.movies.aggregate([ {$unwind: "$directors"}, {$group: {_id: "$directors", total: {$sum: 1}}}, {$sort: {total: -1}}, {$limit: 5} ])
```

Affichage des films triés par note :

```
db.movies.aggregate([ {$sort: {"imdb.rating": -1}}, {$project: {title: 1, "imdb.rating": 1}} ])
```

En résumé : sort permet de définir le tri sur un champ donné. La valeur indiquée est de choisir si le tri est croissant ou décroissant (1 pour le tri croissant, -1 pour le tri décroissant.

Group permet de grouper les champs définis qui ont la même valeur puis d'appliquer une fonction d'agrégation (max, min, sum).

Unwind permet de séparer les éléments d'une table en plusieurs objets distincts.

Project permet de modifier les champs à la convenance et d'en construire de nouveaux.

Mises à jour :

Ajout d'un champ état :
```
db.movies.updateOne({title: "Jaws"}, {$set: {etat: "culte"}})
```

Incrémentation des votes de 100 :
```
db.movies.updateOne({title: "Inception"}, {$inc: {"imdb.votes": 100}})
```

Supprimer le champ poster :
```
db.movies.updateMany({}, {$unset: {poster: ""}})
```

Modifier le réalisateur :
```
db.movies.updateOne({title: "Titanic"}, {$set: {directors: ["James Cameron"]}})
```

En résumé :updateOne est l'instruction principale de mise à jour. Set permet de créer un champ ou de modifier une valeur, Unset d'en supprimer un incr permet d'incrémenter une valeur.

Requêtes complexes :

Affichage des films les mieux notés par décennie :
```
db.movies.aggregate([ {$match: {"imdb.rating": {$exists: true}}}, {$project: { title: 1, decade: {$subtract: ["$year", {$mod: ["$year", 10]}]}, "imdb.rating": 1 }}, {$group: {_id: "$decade", maxRating: {$max: "$imdb.rating"}}}, {$sort: {_id: 1}} ])
```

Affichage des films dont le titre commence par "Star" :
```
db.movies.find({title: /^Star/}, {title: 1})
```

Affichage des films qui ont plus de 2 genres :
```
db.movies.find({$where: "this.genres.length > 2"}, {title: 1, genres: 1})
```

Affichage des films de Christopher Nolan :
```
db.movies.find({directors: "Christopher Nolan"}, {title: 1, year: 1, "imdb.rating": 1})
```

Ces requêtes réutilisent tous les éléments vus précédemment afin de récupérer des données précises et complexes.

Indexation :

Création d'un index sur année :
```
db.movies.createIndex({year: 1})
```

Vérification des index existants :
```
db.movies.getIndexes()
```

Comparaison des requêtes avec et sans l'index :
```
db.movies.find({year: 1995}).explain("executionStats")
```
Sans l'index sur année :

```
executionTimeMillis: 30,
totalDocsExamined: 21349,
```
Avec l'index sur année :
```
executionTimeMillis: 5,
 totalDocsExamined: 372,
```

On a donc une nette amélioration du temps d'exécution et beaucoup moins de documents parcourus.

Suppression de l'index sur année :
```
db.movies.dropIndex({year: 1})
```

Création d'un index composé sur année et rote imbd :
```
db.movies.createIndex({year: 1, "imdb.rating": -1})
```

Partie 3 : Prise en main de MongoDB. Création de requêtes :

Afficher les films d'action :
```
db.films.find({ genre: "Action" })
```

Afficher le nombre de films d'action :
```
db.films.aggregate([
  { $match: { genre: "Action" } },
  { $count: "nombre_de_films_d_action" }
])
```

Afficher les films d'action produits en France:
```
db.films.find({
  genre: "Action",
  country: "FR"
})
```

Afficher les films d'action produits en France en 1963 :
```
db.films.find({
  genre: "Action",
  country: "FR",
  year: 1963
})
```

Afficher les films d'action produits en France sans afficher tous les attributs :
```
db.films.find(
  { genre: "Action", country: "FR" },
  { title: 1, year: 1}
)
```

Afficher les films d'action produits en France sans afficher tous les attributs et sans afficher l'id :

```
db.films.find(
  { genre: "Action", country: "FR" },
  { _id: 0 }
)
```

Afficher les films d'action réalisés en france avec le grade mais sans id :

```
db.films.find(
  { genre: "Action", country: "FR" },
  { _id: 0, title: 1, grades: 1 }
)
```

Afficher les films et les notes de films d'action en france avec une note supérieure à 10, mais affiche tous les films quand même :

```
db.films.find(
  {
    genre: "Action",
    country: "FR",
    "grades.note": { $gt: 10 }
  },
  {
    _id: 0,
    title: 1,
    "grades.note": 1
  }
)
```

Même requête mais en affichant UNIQUEMENT les films avec une note supérieure à 10 :

```
db.films.find(
  {
    genre: "Action",
    country: "FR",
    grades: {
      $not: {
        $elemMatch: { note: { $lte: 10 } }
      }
    }
  }
)
```

Afficher les différents genre de films :

```
db.lesfilms.distinct("genre")
```

Afficher les différents grade de la base :

```
db.lesfilms.distinct("grades.grade")
```

Afficher les films dans lesquels joue un moins un des artistes suivant (s ["artist:4","artist:18","artist:11"]) (requêtes produites même si pas d'id, requête théorique et non testée) :

```
db.lesfilms.find({
  "actors._id": { $in: ["artist:4", "artist:18", "artist:11"] }
})
```

Afficher les films qui n'ont pas de résumé :

```
db.lesfilms.find({
  $or: [
    { summary: { $exists: false } },
    { summary: null },
    { summary: "" }
  ]
})
```

Afficher les films tournés avec Léonardo DiCaprio en 1997 :

```
db.lesfilms.find({
  year: 1997,
  actors: {
    $elemMatch: {
      first_name: "Leonardo",
      last_name: "DiCaprio"
    }
  }
})
```

Afficher tourné avec Leonardo DiCaprio OU tournés en 1997 :

```
db.lesfilms.find({
  $or: [
    { year: 1997 },
    {
      actors: {
        $elemMatch: {
          first_name: "Leonardo",
          last_name: "DiCaprio"
        }
      }
    }
  ]
})
```