Bases de Données NoSQL

2021-2022

Support de Cours

ZOUARI Moez ...

ISET-Sousse

Description du Chapitre

Objectifs :

- Connaître les principes des bases de données NoSQL
- Choisir une solution NoSQL adaptée aux besoins
- Faire passer à l'échelle des bases de données NoSQL
- Gérer une énorme base de données et l'interroger efficacement
- Réaliser des requêtes pour interroger des SGBD NoSQL
- Rechercher et visualiser des données

Prérequis :

- Les Bases de données relationnelles
- Le langage SQL

Chapitre 1 : Introduction

- Historiquement, les bases de données relationnelles était la solution l'incontournable pour stocker les données d'un système d'information.
- Actuellement, Le données gérées pal l'outils informatique ne sont plus les mèmes. En effet, ils n'ont plus ni le même format, ni le même volume. On parle des données de type 3V (*Volume, Velocity, Variety*).
- Le relationnel est parfois n'est plus La meilleur solution pour certains types de données.
- Les nouvelles générations des SGBD doivent supporter les points suivants : La prise en charge des données non-structurées, La distribution, mise à l'échelle horizontale,
- Le NoSQL s'est naturellement imposé dans ce contexte en proposant une nouvelle façon de gérer les données, c'est "**Not Only SQL**".
- Le NoSQL propose une autre manière d'interroger les données, mais aussi de les stocker.
- NoSQL fait référence à une nouvelle classe de technologies de base de données créées pour résoudre des problèmes à l'ère de Big Data
- Les technologies NoSQL ne remplaceront pas les SGBDR
- le tableau suivant montre certaines des différences entre les bases de données SQL et les bases de données NoSQL.

	Bases de données relationnelles	**Bases de données NoSQL**
Charges de travail optimales	Les bases de données relationnelles sont conçues pour les applications de traitement transactionnel et de traitement des transactions en ligne.	Les bases de données NoSQL sont conçues pour un certain nombre de schémas d'accès aux données qui incluent des applications à faible latence. Aussi sont conçues pour l'analyse de données semi-structurées.
Modèle de données	Le modèle relationnel normalise les données dans des tables composées de lignes et de colonnes. Un schéma définit strictement les tables, rangées, colonnes, index, relations entre les tables et autres éléments de base de données. La base de données renforce l'intégrité référentielle dans les relations entre les tables.	Les bases de données NoSQL offrent une variété de modèles de données comme les clés-valeurs, les documents et les graphiques, qui sont optimisés pour les performances et l'évolutivité.
Propriétés ACID	Les bases de données relationnelles fournissent des propriétés d'atomicité, de cohérence, d'isolation et de durabilité	Les bases de données NoSQL font souvent des compromis en assouplissant certaines des propriétés

	(ACID) : • L'Atomicité exige qu'une transaction s'exécute complètement ou pas du tout. • La Cohérence signifie qu'une fois qu'une transaction a été engagée, les données doivent se conformer au schéma de base de données. • L'Isolation nécessite que les transactions simultanées s'exécutent séparément. • La Durabilité implique la capacité à revenir au dernier état connu après une panne système ou une coupure de courant inattendue.	ACID des bases de données relationnelles pour un modèle de données plus flexible qui peut évoluer horizontalement. Les bases de données NoSQL sont un excellent choix pour les cas d'utilisation à haut débit et à faible latence qui ont besoin d'évoluer horizontalement au-delà des limites d'une seule instance.
Performances	Les performances dépendent généralement de l'optimisation des requêtes, des index et de la structure des tables est nécessaire pour obtenir des performances élevées.	Les performances dépendent généralement de la taille du cluster matériel, de la latence réseau.
Évolutivité	Les bases de données relationnelles sont généralement mises à l'échelle en augmentant les capacités de calcul du matériel.	Les bases de données NoSQL sont typiquement partitionnables parce que les modèles d'accès sont capables d'évoluer en utilisant une architecture distribuée pour augmenter le débit qui fournit des performances constantes à une échelle quasi illimitée.
API	Les demandes de stockage et d'extraction de données sont communiquées à l'aide de requêtes conformes à un langage structuré (SQL). Ces requêtes sont analysées et exécutées par la base de données relationnelle.	Les API reposant sur des objets permettent aux développeurs d'applications de stocker et de récupérer très facilement des structures de données. Les clés de partition permettent aux applications de rechercher des paires clé-valeur, des jeux de colonnes ou des documents semi-structurés contenant des objets et attributs d'applications en série.

- Il existe différentes familles de bases NoSQL existent : *Clé/Valeur, colonnes, documents, graphes*.
- Le tableau suivant pris du site : https://db-engines.com/en/ranking , nous donne une idée sur le classement des SGBD sur le marché selon leurs popularités.

Rank			DBMS	Database Model	Score		
Apr 2021	Mar 2021	Apr 2020			Apr 2021	Mar 2021	Apr 2020
1.	1.	1.	Oracle	Relational, Multi-model	1274.92	-46.82	-70.51
2.	2.	2.	MySQL	Relational, Multi-model	1220.69	-34.14	-47.66
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1007.97	-7.33	-75.46
4.	4.	4.	PostgreSQL	Relational, Multi-model	553.52	+4.23	+43.66
5.	5.	5.	MongoDB	Document, Multi-model	469.97	+7.58	+31.54
6.	6.	6.	IBM Db2	Relational, Multi-model	157.78	+1.77	-7.85
7.	7.	↑8.	Redis	Key-value, Multi-model	155.89	+1.74	+11.08
8.	8.	↓7.	Elasticsearch	Search engine, Multi-model	152.18	-0.16	+3.27
9.	9.	9.	SQLite	Relational	125.06	+2.42	+2.87
10.	10.	10.	Microsoft Access	Relational	116.72	-1.41	-5.19
11.	11.	11.	Cassandra	Wide column	114.85	+1.22	-5.22
12.	12.	12.	MariaDB	Relational, Multi-model	96.37	+1.92	+6.47
13.	13.	13.	Splunk	Search engine	88.49	+1.56	+0.41
14.	14.	14.	Hive	Relational	78.50	+2.46	-5.56
15.	↑16.	↑23.	Microsoft Azure SQL Database	Relational, Multi-model	71.84	+0.96	+32.89
16.	↑17.	16.	Amazon DynamoDB	Multi-model	70.73	+1.84	+6.46
17.	↓15.	↓15.	Teradata	Relational, Multi-model	70.55	-0.88	-6.04
18.	↑20.	18.	SAP HANA	Relational, Multi-model	53.44	+2.45	+0.15
19.	19.	19.	SAP Adaptive Server	Relational, Multi-model	51.67	-0.51	-0.96
20.	↓18.	↑21.	Neo4j	Graph, Multi-model	51.04	-1.28	+0.24

- Un autre classement des TOP 11 des Meilleures bases de données NoSQL en 2021 selon le site : https://www.ambient-it.net/top-meilleures-db-nosql-2021/

MongoDB

Il s'agit d'une base de données NoSQL open source orientée document. MongoDB utilise des documents de type JSON pour stocker toutes les données. Il est écrit en C .

Cassandra

Il a été développé sur Facebook pour la recherche dans les boîtes de réception. Cassandra est un système de stockage de données distribué orienté colonnes pour le traitement de très grandes quantités de données structurées.

Redis

Redis est la plus célèbre base clé-valeur. Redis est composé en langage C. Il est autorisé sous BSD.

HBase

Il s'agit d'une base de données distribuée et non relationnelle qui est conçue pour la base de données BigTable par Google.

Neo4j

Neo4j est considéré comme une base de données de graphes native car il implémente efficacement le modèle de graphes de propriétés jusqu'au niveau du stockage.

Chapitre 1 :

Les Bases de données NoSQL de type Clé-Valeur

1. introduction

- Repose sur le couple Clé/Valeur.
- La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données. Une base de données clé-valeur stocke les données sous forme de paires clé-valeur dans lesquelles une clé sert d'identifiant unique
- Il n'y a ni schéma, ni structure pour le stockage.
- Les clés et les valeurs peuvent se présenter sous toutes les formes, des objets simples aux objets composés complexes
- Les bases de données clé-valeur sont hautement divisibles et permettent une mise à l'échelle horizontale à des échelles.

2. Examples:

Phone Directory	
Key	**Value**
Bob	(123) 456-7890
Jane	(234) 567-8901
Tara	(345) 678-9012
Tiara	(456) 789-0123

Artist Info	
Key	**Value**
artist:1:name	AC/DC
artist:1:genre	Hard Rock
artist:2:name	Slim Dusty
artist:2:genre	Country

Stock Trading

Cet exemple utiise une liste de valeurs. La lste contient le numéro de l'opéartion, le type (Buy/Sell), le nombre de pieces et le prix.

Key	Value
123456789	APPL, Buy, 100, 84.47
234567890	CERN, Sell, 50, 52.78
345678901	JAZZ, Buy, 235, 145.06
456789012	AVGO, Buy, 300, 124.50

IP Forwarding Table

Ceci est un exemple de table de transfert IP. Il transmet une adresse IP à une adresse MAC d'un ordinateur physique.

Key	Value
202.45.12.34	01:23:36:0f:a2:33
202.45.123.4	00:25:33:da:4c:01
245.12.33.45	02:03:33:10:e2:b1
101.234.55.1	b8:67:a3:11:23:b1

Les données de twitter

Exemple Twitter

Utilisateurs

Clés primaires	Valeurs
user:guillaumeharry:nom	HARRY
user:guillaumeharry:prenom	guillaume

Messages

Clés primaires	Valeurs
message:post32	"user_id":"guillaumeharry", "time":"26/04/2016", "body":"En pleine présentation"
message:post33	"user_id":"bernardchetrit", "time":"26/04/2016", "body":"En pleine concentration"

Mots clés

Clés primaires	Valeurs
pleine	post32, post33
concentration	post33

3. Cas d'utilisation des bases de Données clés-Valeurs

Les bases de données peuvent être utilisées dans plusieurs cas. On peut citer par exemple :

General Web/Computers

- User profile
- Session information
- Article/blog comments
- Emails
- Status messages

E-commerce

- Shopping cart contents (Gestion des paniers)
- Product catégories (Gestion des catégories)
- Product détails (Détail produit)

Redis : Un exemple de SGBD NoSQL de type clé/Valeurs

4. Tutoriels en ligne de REDIS :

Les sites suivants accessibles sur les URL ci-dessous, contiennent des manipulations pour se familiariser avec redis.

https://redis.io/documentation

https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/redis-tutoriel/

- Il est possible d'utiliser RedIs avec python pour sauvegarder les données. Voir le site suivant :

https://realpython.com/python-redis/

5. Présentation de Redis (REmote DIctionary Server)

Redis :

- Est un Système de Gestion de données open source.
- Conserve les données « en mémoire ».
- Dispose d'un certain nombre de structures de données internes simples comme les chaînes, les hashes, les listes ainsi que certaines autres structures plus complexes optimisées pour la performance d'écriture, d'accès et de calcul.
- Est un système très complet qui sait gérer en natif la réplication, la reprise sur panne, le partitionnement, la persistance des données par sauvegardes complètes ou par lignes de log, ainsi que les transactions atomiques.
    - Il propose également une base de données en mémoire qui permettent au SGBD d'enregistrer toutes les données directement dans la mémoire vive.
        - Des délais d'accès très courts
    - Un système clé-valeur
        - Performance élevée et possibilité de mise à l'échelle grâce à une structure simple.

6. Les types de données Redis

Redis supporte différents types de données :

- strings : une suite de caractères d'une taille maximale de 512 Mo
- hashes : une entrée avec plusieurs champs
- lists : un ensemble de strings triés par ordre de saisie
- sets : un ensemble de strings non triés
- sorted sets : un ensemble de strings triés par l'utilisateur
- bitmaps : un ensemble d'opérations au niveau des bits
- ……

7. Les clients Redis

7.1. Redis-cli

Redis propose une console permettant de se familiariser très vite avec les différentes commandes du client en les testant directement dans une console connectée à un serveur redis.

```
$redis-cli
```

7.2. Les interfaces graphiques

Plusieurs clients basés sur une interface graphique existent pour administrer Redis. Ils sont répartis en trois catégories :

- Clients lourds gratuits pour une utilisation non commerciale de type desktop comme « *Redis Desktop* » : http://redisdesktop.com/
- Clients sous forme de serveur web gratuit et open source comme « *Redis Commander écrit en nodejs : https://www.npmjs.org/package/redis-commander* »

7.3. L'interface Client lourd

Une fois installé, il faut de lancer :

```
A lancer à partir du menu en choisissant « RDM »
```

7.4. L'interface Web

Une fois installé, il faut de lancer :

```
> redis-commander
```

L'interface graphique sera alors disponible à l'adresse http://localhost:8081.

8. Manipulations pratique avec REDIS :

8.1. Sources :

http://b3d.bdpedia.fr/redis.html

8.2. Manipulation des structures de Données

a. Les valeurs simples

La première commande présentée est SET qui permet de stocker très simplement une valeur au moyen d'une clé :

```
> SET ma:clé "ma valeur"
OK
```

Il existe également SETNX qui permet de stocker la valeur d'une clé uniquement si cette clé n'existe pas.

Pour récupérer la valeur associée à notre clé fraîchement stockée il suffit d'utiliser GET suivi du nom de la clé :

```
> GET ma:clé
"ma valeur"
```

Il est également possible de supprimer la clé et sa valeur au moyen de la commande DEL suivi du nom de la clé :

```
> DEL ma:clé
(integer) 1
```

La valeur de retour nous indique que la clé a bien été effacée (1 dans ce cas). Si la clé n'existe pas, la valeur retournée est alors 0.

Il est également possible de stocker des entiers sur lesquels il sera ensuite possible de faire des opérations arithmétiques simples :

```
> SET mon:entier 1
OK
> INCR mon:entier
(integer) 2
```

INCR permet donc d'incrémenter un entier de manière atomique. Ceci nous garantit que plusieurs appels simultanés à la fonction seront exécutés de manière séquentielle et

nous ne risquons pas d'incrémenter la valeur une seule fois alors que nous souhaitions l'incrémenter successivement. Utiliser INCR avec une clé inexistante stockera la valeur 1 pour cette clé. Enfin, INCRBY permet de spécifier de combien l'on souhaite incrémenter la valeur :

```
> INCRBY mon:entier 10
(integer) 12
```

b. Les listes

Redis permet également de stocker des structures de données plus complexes telles que des listes. Afin de créer une liste il suffit d'ajouter un élément à la fin de cette nouvelle liste, l'opération se charge de créer la liste :

```
> RPUSH ma:liste  "Citron"
(integer) 1
> RPUSH ma:liste  "Pomme"
(integer) 2
```

Ici nous remarquons que la valeur de retour correspond au nombre d'éléments présents dans la liste. Il est aussi possible d'ajouter des éléments au début de la liste grâce à LPUSH :

```
> LPUSH ma:liste  "Fraise"
(integer) 3
```

Afin d'accéder aux valeurs de la liste, on utilise LRANGE suivi de la clé et des indices de début et de fin représentés par des entiers :

```
> LRANGE ma:liste  0 -1
1) "Fraise"
2) "Citron"
3) "Pomme"
```

Nous souhaitons ici récupérer les éléments en commençant à l'indice 0 et en parcourant la liste jusqu'à la fin (-1).

Il est possible de récupérer la taille de la liste avec LLEN :

```
> LLEN ma:liste
(integer) 3
```

LPOP permet de supprimer le premier élément de la liste :

```
> LPOP ma:liste
"Fraise"
```

RPOP supprime le dernier élément de la liste :

```
> RPOP ma:liste
```

```
"Pomme"
```

c. *Les sets*

Une autre structure de données proposée est le set. Ce dernier est similaire aux listes mais il ne conserve pas l'ordre des données insérées et une donnée ne peut apparaître qu'une seule fois dans un set. Les commandes principales pour ajouter et supprimer des valeurs dans un set sont SADD et SREM :

```
> SADD mon:set "Assiette"
> SADD mon:set "Fourchette"
> SADD mon:set "Couteau"
> SREM mon:set "Fourchette"
```

La commande SISMEMBER permet de savoir si un élément est contenu dans un set:

```
> SISMEMBER mon:set "Assiette"
(integer) 1
> SISMEMBER mon:set "Fourchette"
(integer) 0
```

SMEMBERS retourne tous les membres d'un set :

```
> SIMEMBERS mon:set
1) "Assiette"
2) "Couteau"
```

Enfin, SUNION retourne les valeurs des différents sets passés en paramètres :

```
> SADD mon:autre-set "Casserole"
> SADD mon:autre-set "Cocotte"
> SUNION mon:set mon:autre-set
1) "Assiette"
2) "Couteau"
3) "Casserole"
4) "Cocotte"
```

Redis propose également des sets ordonnés depuis sa version 1.2 grâce à la commande ZADD. Cette commande permet d'ajouter un score à chaque valeur d'un set.

```
> ZADD mes:amis 1981 "Thomas"
> ZADD mes:amis 1979 "Jérémie"
> ZADD mes:amis 1986 "Céline"
> ZADD mes:amis 1982 "Mano"
> ZADD mes:amis 1988 "Julie"
```

Il est ensuite possible de récupérer les valeurs dans l'ordre grâce à ZRANGE suivi d'un indice de début et d'un indice de fin:

```
> ZRANGE 2 4
1) "Mano"
2) "Céline"
3) "Julie"
```

d. Les hashes

Enfin, Redis permet de stocker des hashes. Ce sont des structures plus complexes permettant de stocker sous une même clé des noms de champs associés à des valeurs, ils sont donc le candidat idéal pour stocker des objets simples :

```
> HSET utilisateur:1 nom "Thomas Bourgier"
> HSET utilisateur:1 email "t.bourgier@exemple.com"
> HSET utilisateur:1 mot_de_passe "1234"
```

Il est également possible de stocker plusieurs champs à la fois :

```
> HSET utilisateur:2 nom "Myriam Fois" email "m.fois@exemple.com"
```

Afin de récupérer les données asso

ciées à une clé, on utilise HGETALL :

```
> HGETALL utilisateur:1
1) "nom"
2) "Thomas Bourgier"
3) "email"
4) "t.bourgier@exemple.com"
5) "mot_de_passe"
6) "1234"
```

Il est aussi possible de récupérer les données d'un seul des champs avec HGET :

```
> HGET utilisateur:1 nom
"Thomas Bourgier"
```

Il est possible d'effectuer les même opérations sur les entiers contenus dans un hash que sur les autres entiers de la base, en utilisant par exemple HINCRBY suivi du nom de la clé, du nom du champ à incrémenter et d'un entier, ou encore HDEL :

```
> HSET utilisateur:1 visites 10
> HINCRBY utilisateur:1 visites 1
(integer) 11
> HDEL utilisateur:1 visites
```

e. Expire

EXPIRE n'est pas un type de donnée mais c'est une fonctionnalité intéressante de Redis puisqu'elle permet de donner une durée de vie à une clé. Quand cette durée est expirée, la clé est simplement supprimée de la base. Il est possible de connaître la durée de vie restante avec la commande TTL, ainsi que de prolonger ou diminuer cette durée de vie en appelant EXPIRE à nouveau :

```
> SET ma:ressource "Je suis éphémère"
> EXPIRE ma:ressource 120
> TTL ma:ressource
(integer) 118
```

Il existe d'autres commandes que nous ne détaillerons pas ici, en voici toutefois la liste complète de manière à vous faire une idée des possibilités du client Redis, d'autant plus que l'on peut déduire l'utilité de nombre d'entre elles en lisant simplement leur nom :

NB : Toutes les commandes sont bien expliquées dans la documentation officielle :

https://redis.io/commands

DECR, DECRBY, DEL, EXISTS, EXPIRE,
GET, GETSET, HDEL, HEXISTS, HGET, HGETALL,
HINCRBY, HKEYS, HLEN, HMGET, HMSET, HSET,
HVALS, INCR, INCRBY, KEYS, LINDEX, LLEN,
LPOP, LPUSH, LRANGE, LREM, LSET, LTRIM,
MGET, MSET, MSETNX, MULTI, PEXPIRE, RENAME,
RENAMENX, RPOP, RPOPLPUSH, RPUSH, SADD,
SCARD, SDIFF, SDIFFSTORE, SET, SETEX, SETNX,
SINTER, SINTERSTORE, SISMEMBER, SMEMBERS,
SMOVE, SORT, SPOP, SRANDMEMBER, SREM, SUNION,
SUNIONSTORE, TTL, TYPE, ZADD, ZCARD, ZCOUNT,
ZINCRBY, ZRANGE, ZRANGEBYSCORE, ZRANK, ZREM,

ZREMRANGEBYSCORE, ZREVRANGE, ZSCORE

8.3. Programmation avec Python

https://docs.redis.com/latest/rs/references/client_references/client_python/

- Connexion :

Ajouter les lignes suivantes dans votre programme python pour lui permettre de se connecter à Redis.

```
import redis

r = redis.Redis(
    host='127.0.0.1',
    port=6379,
    password='')
```

- Tester « get et set » :

```
r.set('pruduct1:name','stylo')
value = r.get('pruduct1:name')
print(value.decode("utf-8") )
```

9. Exercices pratiques avec REDIS :

9.1. Exercice 1 :

Soit le document JSON suivant

```
{"id" :1,
 "nom": {
         "prénom": "Philippe",
         "famille": "Rigaux"
        },
 "tél": 2157786,
 "email": "philippe@cnam.fr"
}
```

1. Ajouter ses valeurs dans la base de données Redis selon format clé valeur. Il faut utiliser le type adéquat.
2. Ajouter les données d'une autre personne fournie dans ce qui suit :

```
{"id" :2,
 "diplomes": ["baccalauréat","Licence","Master"]
 "tél": 2859986,
```

```
}
```

3. Ecrire une programme python pour ajouter lire des informations d'une personne :Id, Nom, age et adresse et les ajouter à la base de données Redis.

9.2. *Exercice 2 :*

1. Ecrire un script python permettant d'insérer les données d'un fichier JSON contenant des données des Films Fourni avec l'énoncé.
2. Afficher le film le plus long
3. Afficher le nombre de films

9.3. *Exercice :*

Soit le script Python suivant :

```
import json
import redis

def get_activities_from_json():
    with open('./prod.activity.json') as file:
        data = json.loads(file.read())
    return data

def import_stringified_json():
    r = redis.StrictRedis(host='localhost', port=6379, db=0)
    pipeline = r.pipeline()
    data = get_activities_from_json()
    for document in data:
        pipeline.set(document['_id']['$oid'],
json.dumps(document))
    pipeline.execute()
```

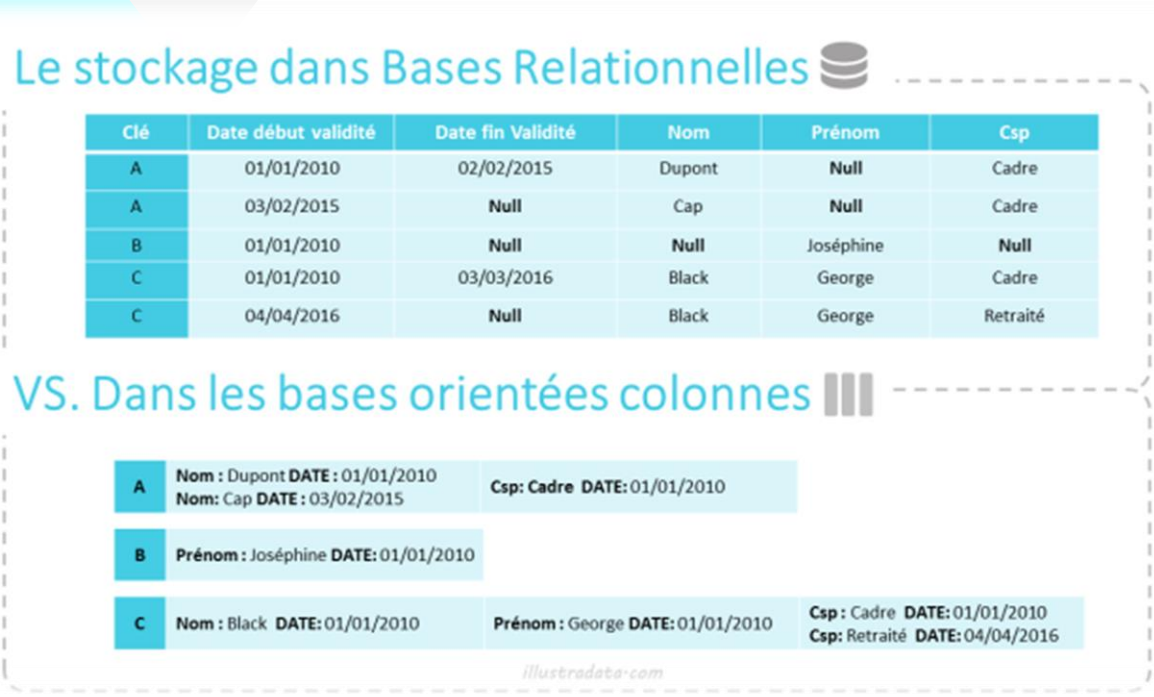
4. Que fait l'instruction pipeline.
   - Consulter : https://redis.io/topics/pipelining
5. Expliquer le principe de fonctionnement du code suivant.
6. Tester ce code après avoir créer le fichier JSON adéquat.

Chapitre 3 :

Les Bases de données NoSQL Orientées Colonne

1. introduction

- Elles sont les plus proches des bases de données classiques (SGBDR) : On y retrouve le principe de “table” avec des lignes et des colonnes.
- **Les colonnes sont dynamiques.** Au sein d’une même table deux individus peuvent ne pas avoir le même nombre de colonnes car **les valeurs nulles ne sont pas stockées** (ce qui est le cas dans les SGBDR relationnels).
    - Cette propriété permet de libérer de la place de stockage et d’améliorer les performances de traitement car la volumétrie de données à traiter est plus faible.
    - Avec cette propriété, on a plus tendance également à ne créer qu’une seule table contenant toutes les données (et donc colonnes) dont on a besoin et non plus une multitude de tables comme c’est le cas dans les modèles relationnels. L’absence de ‘jointure’ entre les tables améliore également les performances.
- **L’historisation des données se fait à la valeur et non pas à la ligne comme dans les SGBDR** cela empêche le stockage d’informations en doublon et de ce fait allège considérablement la base de données et les temps de calcul.
- La figure suivante montre la différence entre un SGBD Relationnel et une base de données NoSQL Orientée Colonne.



2. Les points forts:

- Flexibilité
- Temps de traitement
- Non-stockage des valeurs null
- Historisation à la valeur

3. Les limites :

- Non-adaptée aux données interconnectées
- Non-adaptée pour les données non-structurées

4. Les cas d'utilisation :

- Ces bases sont particulièrement bien adaptées lorsque l'on doit stocker de très nombreux évènements qui doivent être mis à jour très régulièrement. Comme par exemple :
    - Le suivi de colis (de nombreux évènements dont le statut change : En préparation, en cours de livraison, livré..)
    - La récupération et l'analyse de données en temps réel issues de capteurs, IOT etc.....

5. Les principales solutions :

6. Tutoriaux

- Hbase : http://hbase.apache.org/
- Cassandra : http://cassandra.apache.org/
- Google Big Table : https://cloud.google.com/bigtable/
- Hypertable: http://www.hypertable.org/
- Apache Parquet : https://parquet.apache.org/

Pour notre cours nos choisissons Hbase.

7. Manipulations pratique avec REDIS :

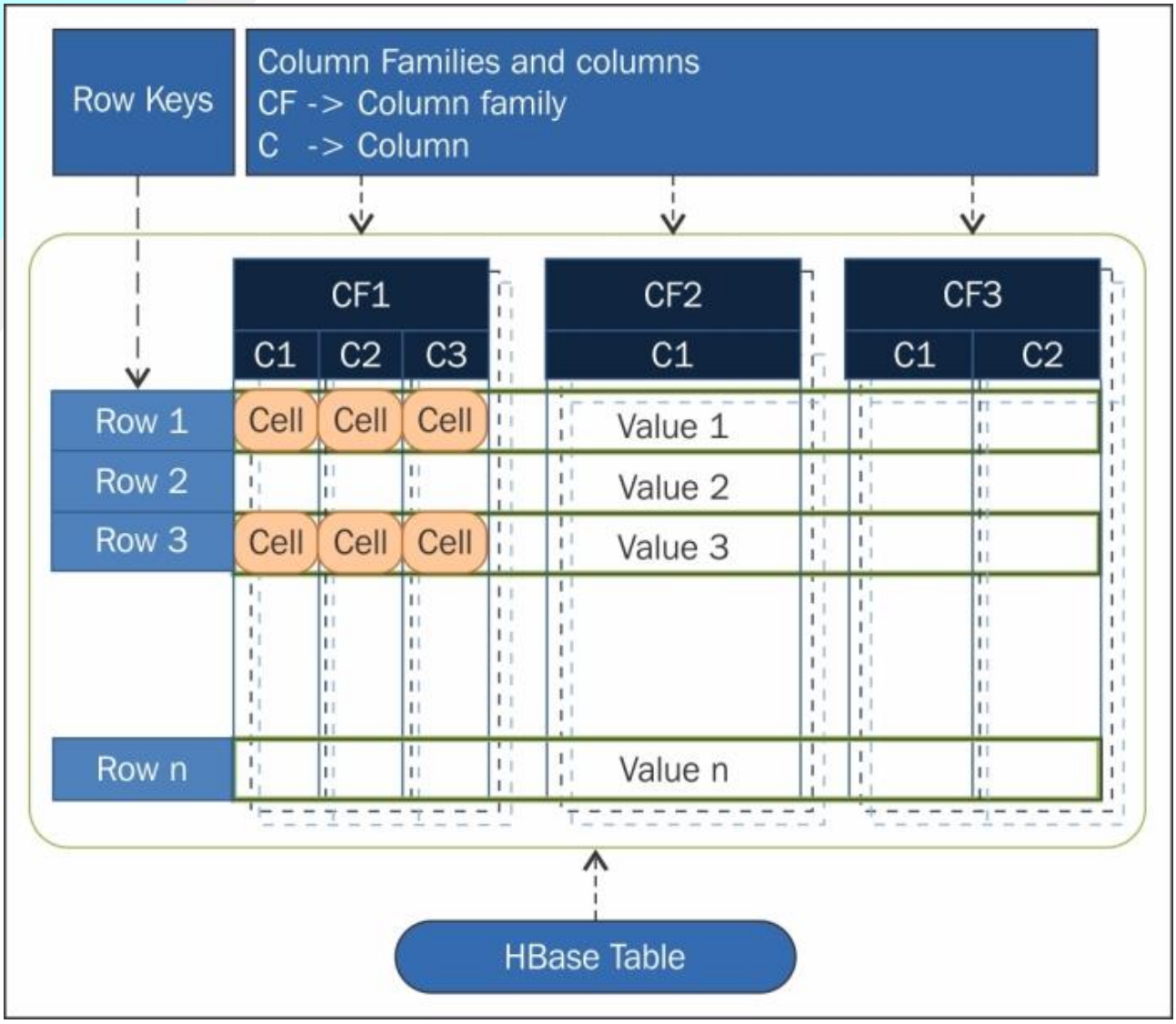
7.1. Présentation :

HBase est un système de gestion de bases de données distribué, non-relationnel et orienté colonnes, développé au-dessus du système de fichier HDFS.
Il permet un accès aléatoire en écriture/lecture en temps réel à un très grand ensemble de données.

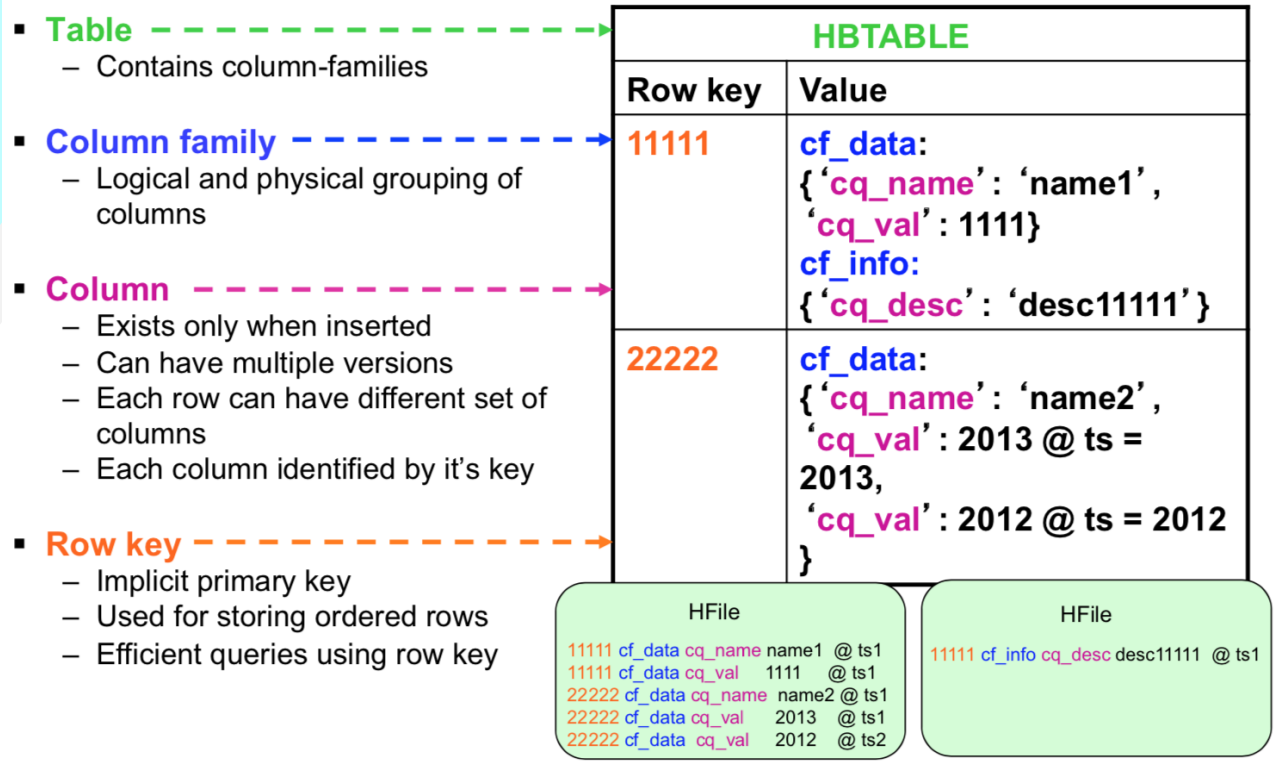
7.2. Modèle de données

Le modèle se base sur six concepts, qui sont :

- **Table** : dans HBase les données sont organisées dans des tables. Les noms des tables sont des chaînes de caractères.
- **Row** : dans chaque table, les données sont organisées dans des lignes. Une ligne est identifiée par une clé unique (RowKey). La Rowkey n'a pas de type, elle est traitée comme un tableau d'octets.
- **Column Family** : Les données au sein d'une ligne sont regroupées par column family. Chaque ligne de la table a les mêmes column families, qui peuvent être peuplées ou pas. Les column families sont définies à la création de la table dans HBase. Les noms des column families sont des chaînes de caractères.
- **Column qualifier** : L'accès aux données au sein d'une column family se fait via le column qualifier ou column. Ce dernier n'est pas spécifié à la création de la table mais plutôt à l'insertion de la donnée. Comme les rowkeys, le column qualifier n'est pas typé, il est traité comme un tableau d'octets.
- **Cell** : La combinaison du RowKey, de la Column Family ainsi que la Column qualifier identifie d'une manière unique une cellule. Les données stockées dans une cellule sont appelées les valeurs de cette cellule. Les valeurs n'ont pas de type, ils sont toujours considérés comme tableaux d'octets.
- **Version** : Les valeurs au sein d'une cellule sont versionnés. Les versions sont identifiés par leur timestamp (de type long). Le nombre de versions est configuré via la Column Family. Par défaut, ce nombre est égal à trois.



Les données dans HBase sont stockées sous forme de HFiles, par colonnes, dans HDFS. Chaque HFile se charge de stocker des données correspondantes à une column family particulière.



Autres caractéristiques de HBase:

- HBase n'a pas de schéma prédéfini, sauf qu'il faut définir les familles de colonnes à la création des tables, car elles représentent l'organisation physique des données
- Les données HBase sont décrite sous forme de clef/valeur, où la clef est la combinaison (row-column family-column-timestamp) représente la clef, et la cell représente la valeur.

Dans le cas de distribution, les tables HBase sont divisées horizontalement, par row en plusieurs **Regions**. Une region contient toutes les lignes de la table comprises entre deux clefs données. Les regions sont affectées à des noeuds dans le cluster, appelés Region Servers, qui permettent de servir les données pour la lecture et l'écriture. Un region server peut servir jusqu'à 1000 régions.

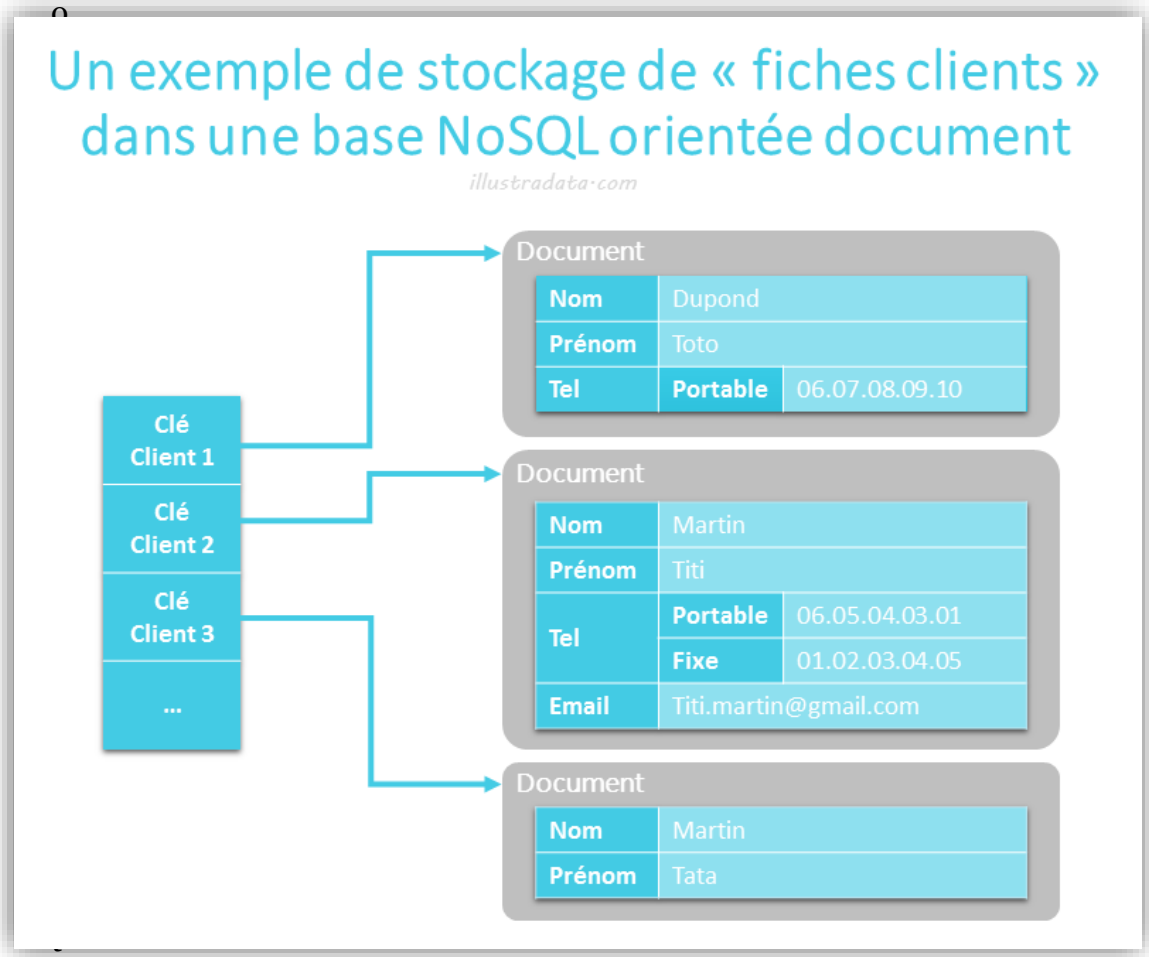
Chapitre 4 :

Les Bases de données NoSQL Orientées Document

1. Introduction

- Les bases de données orientées documents créent une paire simple : une **clé** est associée à un document spécifique.
- Elles reposent sur le paradigme [clé, valeur] : la valeur, dans ce cas, est un document.
- Un document a une structure arborescente : il contient une liste de champs, un champ est associé à une valeur qui peut-elle même être une liste. Ces documents s

o



de type **JSON** ou **XML**.

- Elles sont basées sur une structure de tableau simple et des documents **permettant d'enregistrer les informations.**

- Elles sont Dédiées pour les données qui n'ont pas de structure claire, et qui ne ne sont pas adaptées à des bases de données relationnelles
- Elles utilisent généralement un format de fichier pour les documents et l'on intègre les informations dans une structure fixe. Ce qui permet de traiter plus efficacement les requêtes de recherche dans la base de données.
- Chaque document devrait avoir un **identifiant unique**

2. Les avantages des bases de données orientées documents:

- La base de données peut héberger de grands volumes de données non structurés.
- Les informations ne sont pas partagées entre plusieurs tableaux reliés les uns aux autres. Tout se trouve à un même endroit ce qui peut entraîner une meilleure performance. En effet, Les données orientées documents se caractérise par l'unité d'information et l'adaptation à la distribution.
- Les documents sont autonomes, on peut les déplacer facilement, ils sont indépendants les uns des autres.
- Les documents sont structurés mais aucune définition de structure préalable n'est nécessaire.

3. Les bases de données orientées documents les plus connues

Pour le développement d'applications Web en particulier, les bases de données jouent un rôle considérable pour les documents. En raison de la forte demande résultant du développement Web, de nombreux **systèmes de gestion de bases de données** (SGBD) sont maintenant disponibles sur le marché. La liste suivante présente les plus connus :

- **BaseX** : ce projet open source utilise Java et XML. BaseX est fourni avec une interface graphique utilisateur.

- **CouchDB** : l'Apache Software Foundation a publié le logiciel open source CouchDB. Ce système de gestion de base de données est codé en Erlang, utilise JavaScript et est notamment utilisé dans Ubuntu et dans les applications de Facebook.
- **Elasticsearch** : ce moteur de recherche fonctionne sur la base d'une base de données orientée documents. Pour ce faire, des documents JSON sont utilisés.
- **eXist** : le système de gestion de base de données open source eXist fonctionne via une machine virtuelle Java et peut donc être utilisé indépendamment d'un système d'exploitation. Les documents utilisés sont principalement au format XML.
- **MongoDB** : MongoDB est la base de données NoSQL la plus répandue. Le logiciel est codé en C++ et utilise des documents similaires à JSON.
- **SimpleDB** : avec SimpleDB (codé en Erlang), Amazon a développé son propre système de gestion de base de données pour les services Cloud de l'entreprise. Le fournisseur facture des frais pour son utilisation.

4. Tutoriel MongoDB

Présentation MongoDB:

- *MongoDB* est une base de données NoSQL distribuée de type *Document Store dont le p*rincipe de base est : les données sont des documents stockés en Binary JSON (BSON)
- Les documents similaires rassemblés dans des collections
- Pas de schéma des documents définis à l'avance.
- Les documents peuvent n'avoir aucun point commun entre eux.
- Un document contient (généralement) l'ensemble des informations pas (ou très peu) de jointure à faire

Quelques Idées sur JSON :

- JSON : JavaScript Object Notation
- Format d'échange de données structurées léger
- Schéma des données non connu : contenu dans les données
- Basé sur deux notions :
    - collection de couples clé/valeur

- liste de valeurs ordonnées

- Structures possibles :
  - objet (couples clé/valeur) :
    - {}
    - { "nom": "jollois", "prenom": "fx" }
  - tableau (collection de valeurs) :
    - []
    - [1, 5, 10]

- Deux types atomiques (string et number) et trois constantes (true, false, null)
- Validation possible du JSON sur jsonlint.com/

```
{
	"tubd": {
		"formation": "DU Analyste Big Data",
		"responsable": { "nom": "Poggi", "prenom": "JM" },
		"etudiants" : [
			{ "id": 1, "nom": "jollois", "prenom": "fx" },
			{  "id":  2,  "nom":  "aristote",  "details":
"délégué" },
			{ "id": 5, "nom": "platon" }
		],
		"ouverte": true
	},
	"tudv": {
		"formation": "DU Data Visualisation",
		"ouverte": false,
		"todo": [
			"Creation de la maquette",
			"Validation par le conseil"
			],
		"responsable": { "nom": "Métivier" }
	}
}
```

Quelques Idées concernant BSON :

- BSON : extension de JSON
  - Quelques types supplémentaires (identifiant spécifique, binaire, date, ...)
  - Distinction entier et réel
- Schéma dynamique

- Documents variant très fortement entre eux, même dans une même collection
- On parle de self-describing documents
- Ajout très facile d'un nouvel élément pour un document, même si cet élément est inexistant pour les autres
- Pas de ALTER TABLE ou de redesign de la base
- Pas de jointures entre les collections

10. Manipulations pratique avec MongoD :

Les principales opérations avec MongoDB

- **filtarge :** find() findOne()
- **la projection :** db.getCollection('collection').find({filtre}{clé:1})
- **Filtrage avec des opérations :** clé:{$opérateur:valeur}
- **liste de valeurs distincts :** db.rcollectiondistinct("clé")
- **update :** db.collection.update({filtre}{$set:{clé:valeur})
- **supprimer clé :** db.collection.update({filtre}{$unset:{clé:1})
- **supprimer des documents** : db.collection.remove(filtre)
- **insérer :** insertOne/insertMany

La commande help affiche la liste de toutes les commandes :

```
mongo> db.help()
```

1. Gestion de collection

Exercice 1 : démarrage de MongoDB et importation d'une Base de données

1. Lancement du serveur : ilfaut créer un répertoire où seront stockées les données de la base MongoDB. Utiliser ~/mongoDB
2. Lancez le serveur en entrant : mongod --dbpath ~/mongoDB
3. Exécuter les commandes suivantes :
   - Se connecter à la base dbFilms :

```
use dbFilms;
```

- Créer une collection «cfilms» : `db.createCollection('cFilms');`

Créer le document suivant (Utiliser pour cela la commande insertOne):

{

"Title": "Avatar",

"Year": "2009",

"Rated": "PG-13",

"Released": "18 Dec 2009",

"Runtime": "162 min",

"Genre": "Action, Adventure, Fantasy",

"Director": "James Cameron",

"Writer": "James Cameron",

"Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang"

}

4. Consulter le contenu de la collection : db.cFilms.find()
5. Importer le fichier JSON films.json fournie avec l'énoncé.

```
$ mongoimport --host localhost:27017 --db dbFilms --collection cFilms ~/films.json
```

6. Afficher le contenu de toute la collection

```
db.cFilms.find()
```

Remarque : pour un affichage plus agréable, utiliser la commande :

```
dbcFilms.find().pretty()
```

7. Exprimez des requêtes simples pour les recherches suivantes :
   a. Liste de tous les films (Type «series») ;

   →find

   b. Liste des films parus après 2015 ;

c. Liste des series depuis 2015 ;

d. Liste des films du réalisateur « James Cameron » ;

e. Liste des films de genre Thriller

→{"Genre" : /thriller/i }

f. Le nombre total des films

→ .count()

g. Afficher uniquement les titres des films joués par « Gerard Butler »

h. Afficher les titres et les écrivains des films de « comedy » joué par « Leonardo DiCaprio »

i. afficher la liste des titres films triés :

→{$sort

j. Mettre à jour l'année d'édition du film avatar en 2019

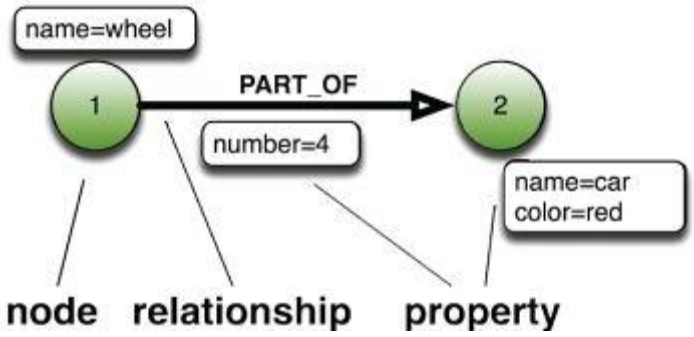
k. Supprimer les clés images de tous les documents

Chapitre 1 :

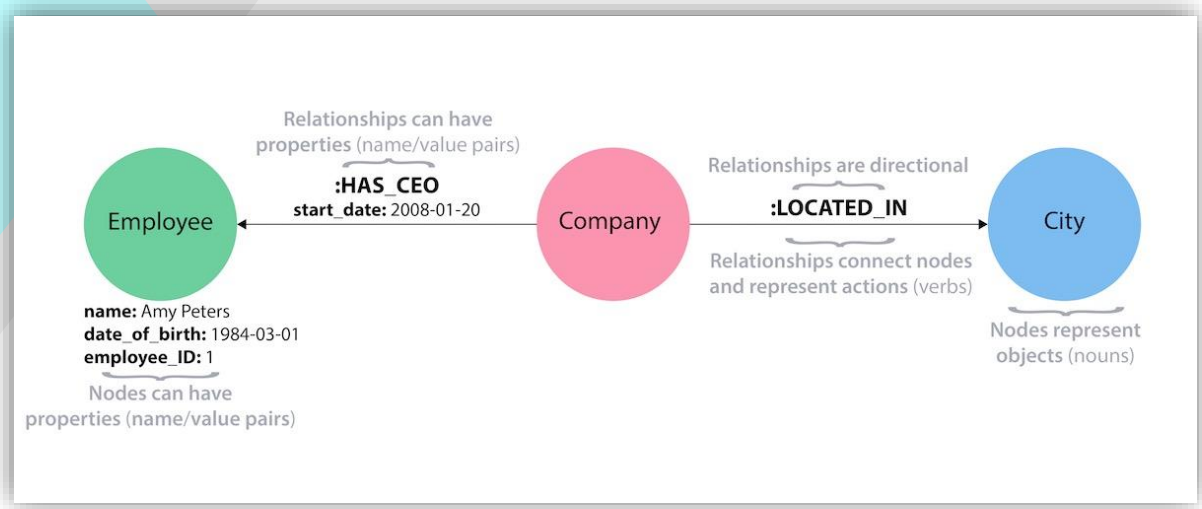
Les Bases de données Orientées Graphes

1. Presentation

- Conçues pour stocker et rechercher des relations entre des entités
- Les bases de données orientées graphe utilisent des nœuds, des propriétés et des arêtes pour représenter les données :
    - Les nœuds ou sommet (*node, vertex*)représentent des entités ou des instances telles que des personnes, des entreprises, des comptes ou tout autre élément à suivre. Ils peuvent être comparés aux documents du modèle orientée document. Un nœud est défini par un ou plusieurs labels, qui sont des propriétés sous forme de paires clé / valeur.
    - Les bords, également appelés graphiques ou relations ou arêtes (*relationship, edge*), avec une orientation et un type (orienté et marqué) sont les lignes qui relient les nœuds à d'autres nœuds ; ils représentent la relation entre eux. Ce sont le concept clé des bases de données de graphes.
    - Les propriétés ou attribut (*property, attribute*) sont des informations associées aux nœuds ou une relation.



- Il s'agit d'un graphe attribué, marqué et orienté :
    - Marqué par une étiquette à chaque arête.
    - Les arêtes ont une direction fixée, depuis le nœud source vers le nœud destination
- Exemple de modélisation de données :



2. Types de graphes

Il existe plusieurs types de graphiques qui peuvent être classés par catégories.

Graphe social : il s'agit des liens entre les personnes ; par exemple les données de Facebook, Twitter, …

- Graphe d'intention : il traite du raisonnement et de la motivation. Expliquer des faits, des causes conséquences
- Graphe de consommation : également appelé "graphique de paiement", le graphique de consommation est très utilisé dans le secteur du commerce de détail pour suivre la consommation des clients individuels.
- Graphe des intérêts : ce graphique représente les intérêts d'une personne. (Regarder, écouter, apprécier, ...)
- Graphe mobile : il est construit à partir de données mobiles.

3. SGBDS Oriéntés graphes

- Il existe plusieurs SGDB orienté graphes :
  - Neo4j - Open Source
  - AllegroGraph - Source fermée
  - Sones - Source fermée
  - Virtuoso - Source fermée
  - HypergraphDB - Open Source
  - ...

Neo4j : Un exemple de SGBD NoSQL Orienté Graphe

1. Tutoriels en ligne de Neo4j :

Les sites suivants accessibles sur les URL ci-dessous, contiennent des manipulations pour se familiariser avec redis.

- https://www.tutorialspoint.com/neo4j/index.htm
- https://neo4j.com/developer/cypher/guide-cypher-basics/

2. Présentation de Neo4j)

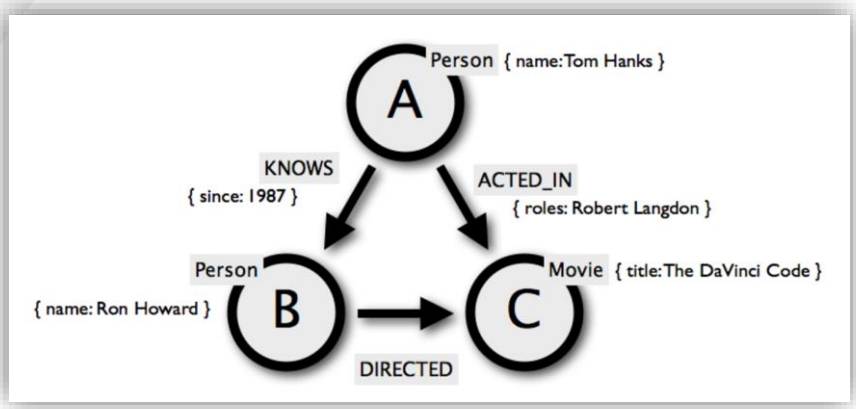
Neo4j :

- Est une base de données de graphes native open source, NoSQL, qui fournit un backend transactionnel compatible ACID.
- Est devenu public depuis 2007.
- Est open source, écrit en Java et Scala.
- Conserve les données « en mémoire ».
- Est une base de données de graphes native car il implémente efficacement le modèle de graphe de propriétés jusqu'au niveau de stockage.
- Fournit des caractéristiques de base de données complètes : la conformité des transactions ACID, la prise en charge des clusters et le basculement de l'exécution.
- Offre :
    - Un langage de requête déclaratif similaire à SQL appelé Cypher qui est optimisé pour les graphiques.
    - Des Pilotes pour les langages de programmation populaires, notamment Java, JavaScript, .NET, Python et bien d'autres

3. Manipulation Pratique

Les données que nous traitons représentent :

- Les **films** avec un label *: Movie* et les propriétés *title, released* et *tagline*
- Les **personnes** avec un label : Person et des propriétés *name* et *born*
- Une relation *ACTED_IN* allant d'un acteur à un film, avec une propriété *roles* contenant la liste des noms des caractères
- Une relation *DIRECTED* allant d'un réalisateur à un film.



Activité 1. Commencer par installer Neo4J. Lancez-le sur votre navigateur.

Créer la base de données Movies donnée comme exemple et récupérer le la bases de données movies

1. Installation de Neo4J

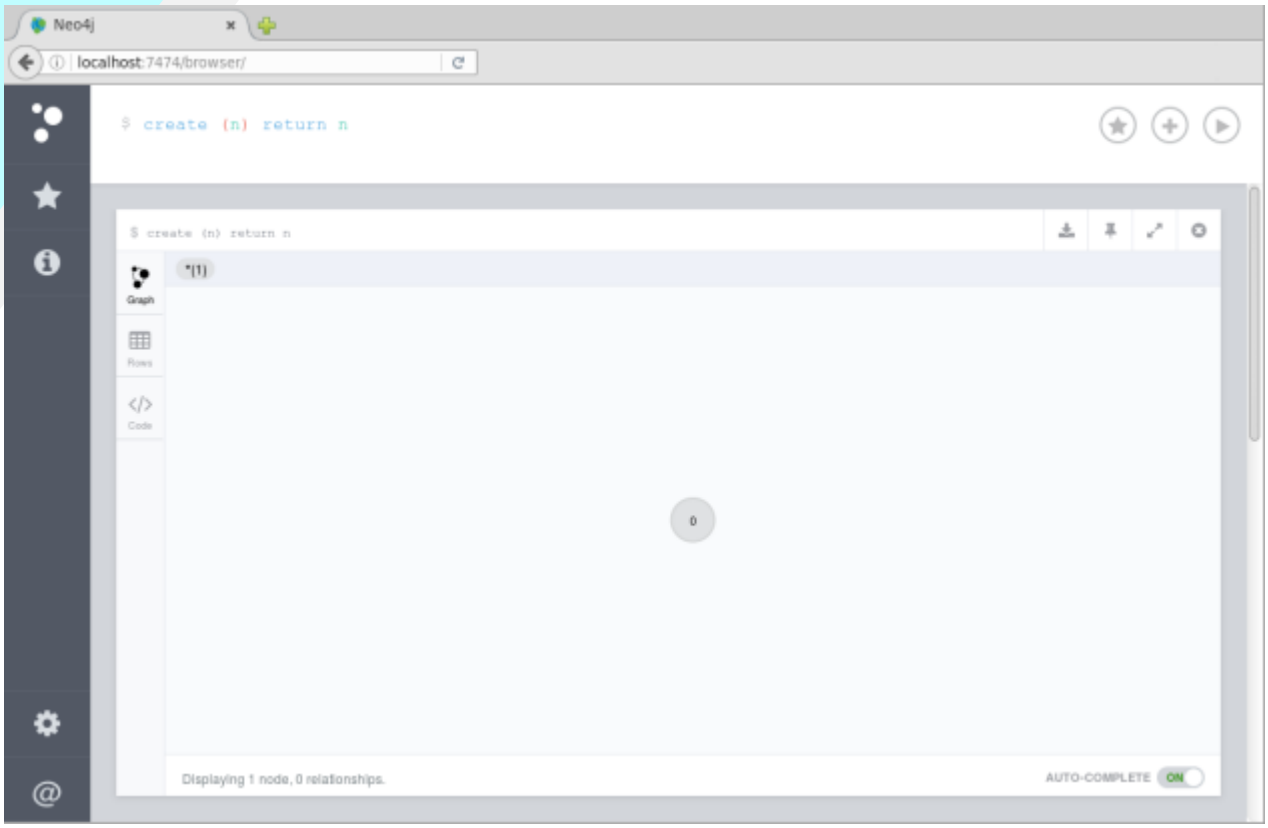
1. Télécharger depuis : http://neo4j.com/download/
2. Choisir la version `adéquate`
3. Suivez la procédure suivant votre système d'exploitation (Mac OSX, Linux/UNIX, Windows)

2. Lancer le serveur

Le démarrage du serveur dépend du système d'exploitation.

- Sous Windows et Mac OSX lancer l'application et cliquer sur le bouton `Start`

3.Test de Neo4J



Interface de Neo4j (création d'un nœud)

- Les requêtes s'écrivent dans la partie haute.
- Les résultats se visualisent dans la partie basse.
- ExempleCréer un nœud

```
create (n) return n
```

- ExempleVoir tous les nœuds

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

Créer un nœud avec une propriété

```
create (n {name:'Sun'}) return n
```

*Exemple*Créer un nœud avec un label

```
create (n:Astre {name:'Sun'}) return n
```

*Syntaxe*Créer des relations

le formalisme pour les relations est :

- ()<-[]-()
- ()-[]->()
- ()<-[]->()
- ()-[]-()

où :

- on mettra nos nœuds à l'intérieur des parenthèses ()
- on mettra nos relations à l'intérieur des crochets []
- enfin on désignera le sens de la relation avec la flèche

*Exemple*Créer deux nœuds et une relation entre ces deux noeuds

```
create (e:Astre {name:'Earth'}) <-[:Satellite {distance:384000}]- (m:Astre {name:'Moon'}) return e, m
```

*Exemple*Créer deux nœuds et une relation entre ces deux noeuds en plusieurs instructions

```
create (e:Astre {name:'Earth'})

create (m:Astre {name:'Moon'})

create (e)<-[:Satellite {distance:384000}]-(m)

return e, m
```

4.Le Langage Cypher

Nœuds

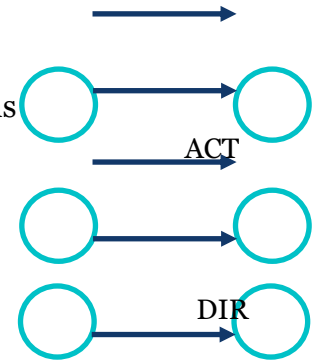
(a) Acteurs

(m) Films

( ) Noeud Anonyme

Relations

-[r]->	relation appelée r
(a) –[r]-> (m)	acteurs ayant une relation r avec des films
-[:ACTED_IN]->	La relation est de type *ACTED_IN*
(a)-[:ACTED_IN]->(m)	Les acteurs qui jouent dans un film
(d)-[:DIRECTED]->(m)	Les réalisateurs d'un film



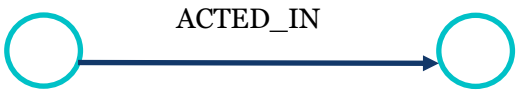
Propriétés des Nœuds

(m {title : "The Matrix"}) Film avec une propriété *title*

(a {name: "Keanu Reeves", born : 1964}) Acteur avec des propriétés *name* et *born*

Propriétés des Relations

(a)-[:ACTED_IN {roles :["Neo"]}]->(m) Relation *ACTED_IN* avec la propriété *roles*, qui est un tableau de noms



Labels : permettent de reconnaître différents types de noeuds

(a:Person)-[:ACTED_IN]->(m:Movie) Une personne qui a joué dans un film

5.Requêtes Cipher

Requêtes de lecture simples

Voici des exemples de requêtes Cipher:

```
MATCH (node) RETURN node.property

MATCH (node1)-->(node2)
RETURN node2.propertyA, node2.propertyB
```

La première instruction permet de retourner la valeur de la propriété *property* pour tous les nœuds. La seconde, retourne les valeurs de *propertyA* et *propertyB* de chaque couple de nœuds *node2* et *node1* qui sont connectés par une relation (quelle que soit sa nature).

Pour retourner seulement les nœuds connectés par une relation de type *REL_TYPE* :

```
MATCH (node1)-[:REL_TYPE]->(node2)
```

Si la relation en question comporte des propriétés qu'on désire lire, lui donner un nom (ici *rel*) :

```
MATCH (node1)-[rel:TYPE]->(node2)
RETURN rel.property
```

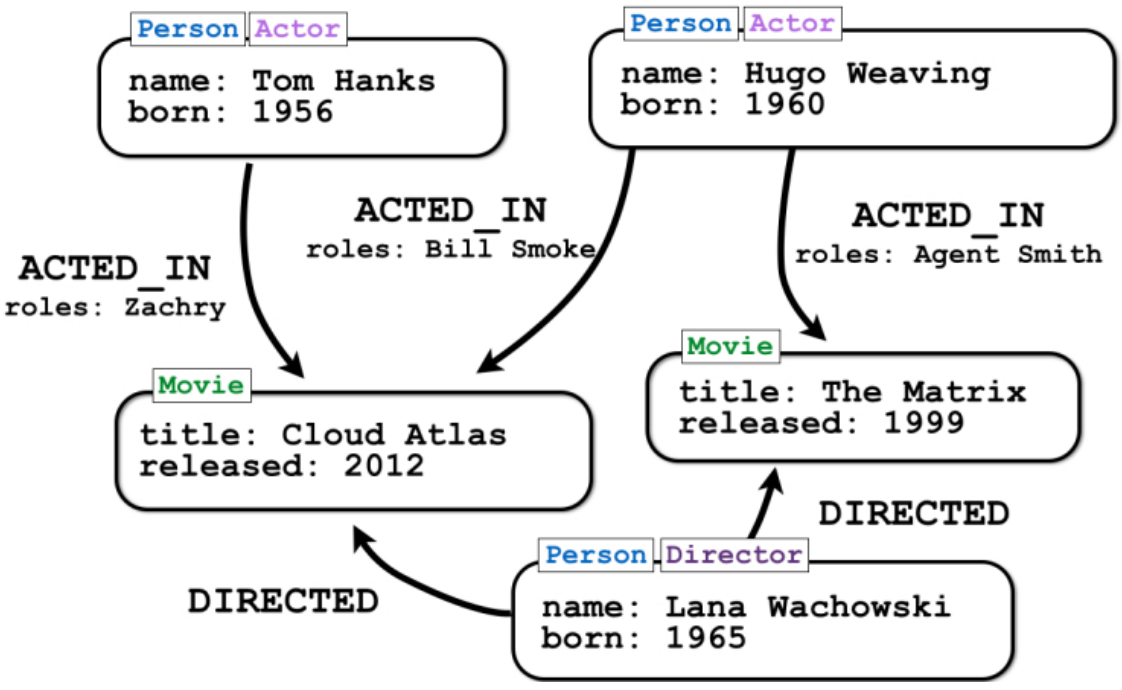
Pour indiquer un type particulier de nœuds, utiliser les labels :

```
MATCH (node:Label) RETURN node

MATCH (node1:Label1)-[:REL_TYPE]->(node2:Label2)
RETURN node1, node2
```

Il est possible d'affecter un identifiant à une relation, puis retourner le type de la relation, ou bien ses propriétés :

```
MATCH (node)-[rel]-> ()
RETURN node, rel.property;

MATCH (node)-[rel]-> ()
RETURN node, type(rel);
```

Pour étayer par un exemple, soit le graphe suivant :



La requête permettant de trouver tous les caractères du film *The Matrix* est la suivante :

```
MATCH    (movie :Movie)<-[role :ACTED_IN]-(actor :Person)
WHERE    movie.title= « The Matrix »
RETURN   role.roles, actor.name
```

Un autre moyen est également possible pour filtrer les données (à la place du traditionnel WHERE), et ce remplaçant la requête précédente par :

```
MATCH    (movie :Movie {title : "The Matrix"})<-[role :ACTED_IN]-(actor :Person)
RETURN   role.roles, actor.name
```

Activité 2. Exécuter les requêtes suivantes dans votre éditeur:

- Donner tous les films où a joué "Tom Hanks"
- Donner la liste de tous les personnages de "The Matrix"
- Donner pour chaque film sa date de sortie

Les Chemins (*Paths*)

Un chemin est un ensemble de nœuds connectés et de relations, pouvant correspondre à un patron. Par exemple :

(a) --> (b) --> (c)

(a) --> (b) <-- (c)

Exemple : Pour afficher tous les acteurs et réalisateurs de tous les films :

```
MATCH    (a)-[ :ACTED_IN]->(m)<-[ :DIRECTED]-(d)
RETURN   a.name, m.title, d.name
```

Il est possible également de diviser la requête en plusieurs chemins, comme suit :

```
MATCH    (a)-[ :ACTED_IN]->(m),
         (m) <-[ :DIRECTED]-(d)
RETURN   a.name, m.title, d.name
```

Ou bien :

```
MATCH    (a)-[ :ACTED_IN]->(m),
         (d)-[ :DIRECTED]->(m)
RETURN   a.name, m.title, d.name
```

Les résultats des requêtes précédentes sont des tableaux.

Il est possible de retourner le chemin entier, avec toutes les propriétés de tous les nœuds et relations concernés :

```
MATCH   p = (a)-[ :ACTED_IN]->(m)<-[ :DIRECTED]-(d)
RETURN  p
```

Si cela représente trop d'informations, vous pouvez juste sélectionner les nœuds du chemin :

```
MATCH   p = (a)-[ :ACTED_IN]->(m)<-[ :DIRECTED]-(d)
RETURN  nodes(p)
```

... ou bien seulement les relations :

```
MATCH   p = (a)-[ :ACTED_IN]->(m)<-[ :DIRECTED]-(d)
RETURN  rels(p)
```

Activité 3. Exécuter les requêtes suivantes dans votre éditeur:

- Quels réalisateurs ont joué dans leurs propres films ?
- Donner les films où Keanu Reeves a joué le rôle de Neo.
- Quels acteurs ont déjà joué dans un même film avec Tom Hanks, et dans quel film ?
- Quels acteurs ont joué dans un même film avec Tom Hanks, et avec Keanu Reeves ?

Éditer un graphe avec Cipher

Pour ajouter un nœud au graphe :

```
CREATE (me:Person {name: "My Name"}) return me;
```

Pour ajouter (ou modifier) des propriétés à un nœud existant :

```
MATCH (movie:Movie)
WHERE movie.title="Mystic River"
SET movie.tagline = "We bury our sins here, Dave. We wash them clean."
RETURN movie;
```

Pour ajouter une relation entre deux nœuds existants :

```
MATCH (me:Person), (movie:Movie)
WHERE me.name="My Name" AND
      movie.title="Mystic River"
CREATE (me)-[:REVIEWED {rating:80, summary:"tragic character movie"}]->(m
ovie);
```

Activité 4. Exécuter les requêtes suivantes dans votre éditeur:

- Créer un film appelé "Mystic River"
- Ajouter Kevin Bacon comme acteur dans le film « Mystic River » avec le rôle « Sean »
- Modifier le rôle pour devenir « Sean Devine »