

Module

INGÉNIERIE BIG DATA NoSQL

Présenté par:

Fatma Abdelhedi, PhD

 fatma.abdelhedi@trimane.fr

Bilel SDIRI, PhD

 bilel.sdiri@trimane.fr



TRIMANE

THE DATA INTELLIGENCE
COMPANY

A PART OF THE BLOCKCHAIN GROUP

102 TERRASSE BOIELDIEU , TOUR W
92 800 PUTEAUX

www.trimane.fr

Plan

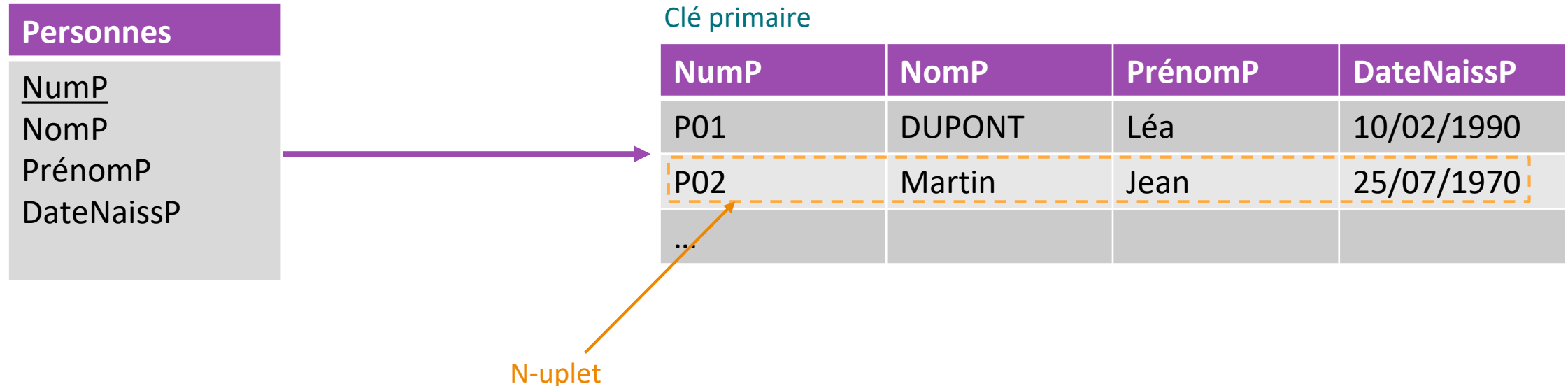
1. Introduction aux Big Data
2. Les bases de données NoSQL
3. Modélisation des données Big Data
4. La solution MongoDB
5. Manipulation des données avec MongoDB (CRUD)
6. Agrégation des données
7. Jointures et références
8. Recherche d'information

Plan

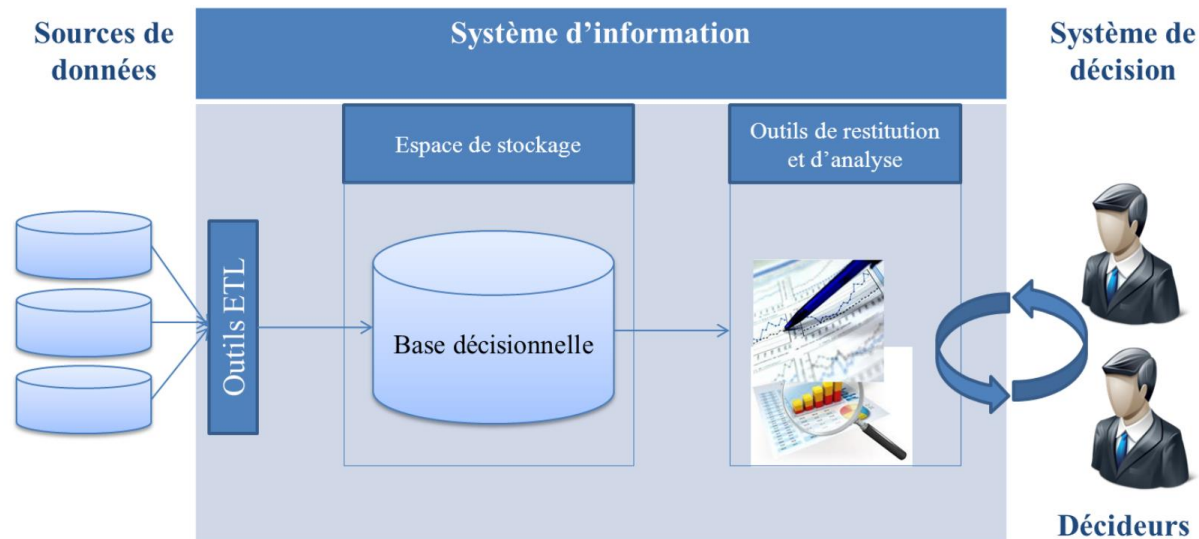
1. Introduction aux Big Data
2. Les bases de données NoSQL
3. Modélisation des données Big Data
4. La solution MongoDB
5. Manipulation des données avec MongoDB (CRUD)
6. Agrégation des données
7. Jointures et références
8. Recherche d'information

❑ BDR : Base de données relationnelle

- Depuis les années 70, les BDRs étaient considérées comme étant la référence pour la gestion des données des systèmes d'information.
- Le langage SQL était standardisé malgré les variantes.
- Repose sur des concepts mathématiques simples.
- Manipulation : Union, Restriction, Projection, Jointure.
- Description : Table, Attributs, Clés, CIR.

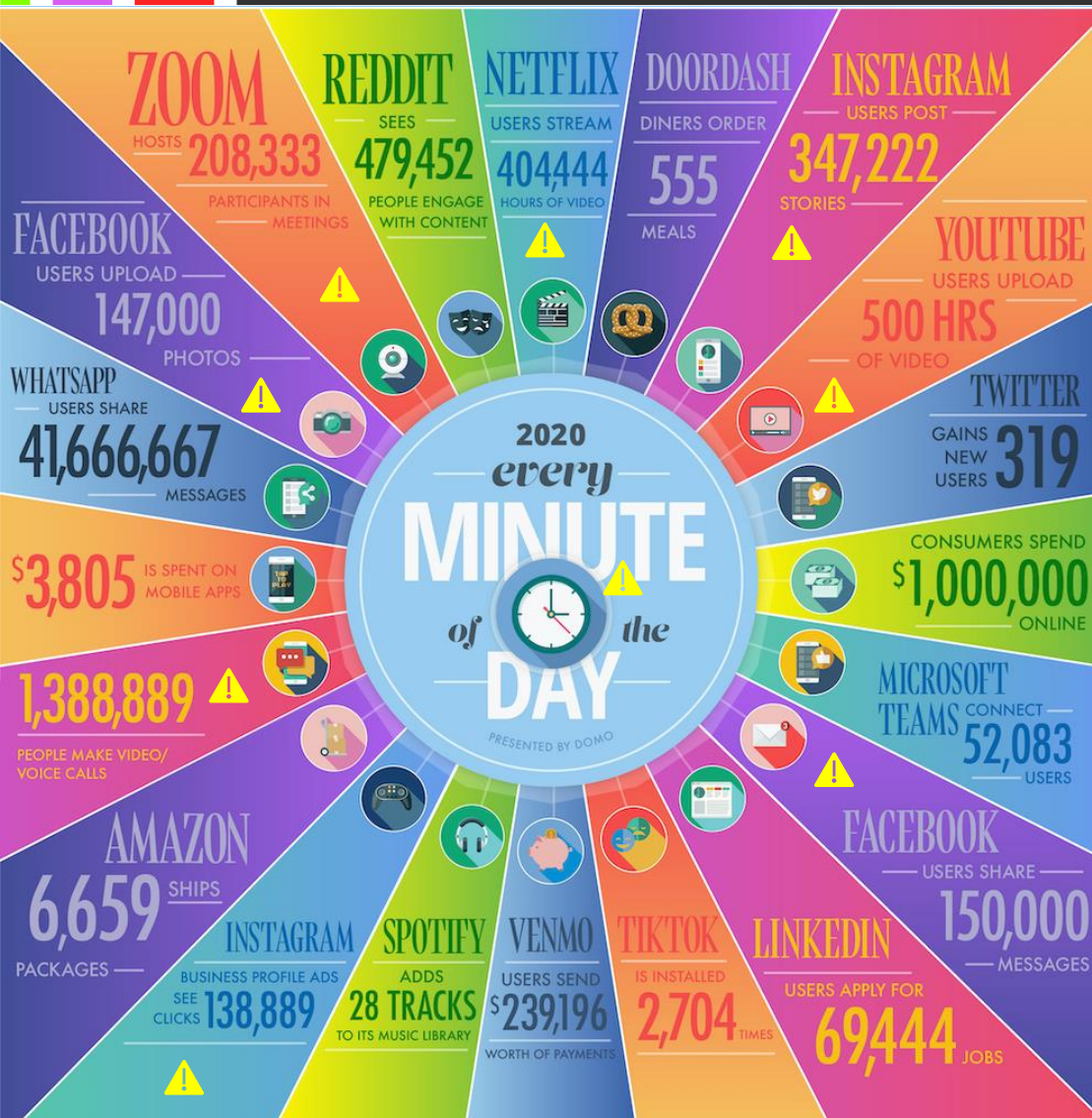


Définition : Une base décisionnelle est l'ensemble des outils informatiques (matériels et logiciels) permettant l'analyse des données. Ces données sont représentées dans la base décisionnelle en une vision orientée décideur puis analysées au moyen des outils de restitution et d'analyse.



1. Introduction aux Big Data

1.3. Quelques chiffres

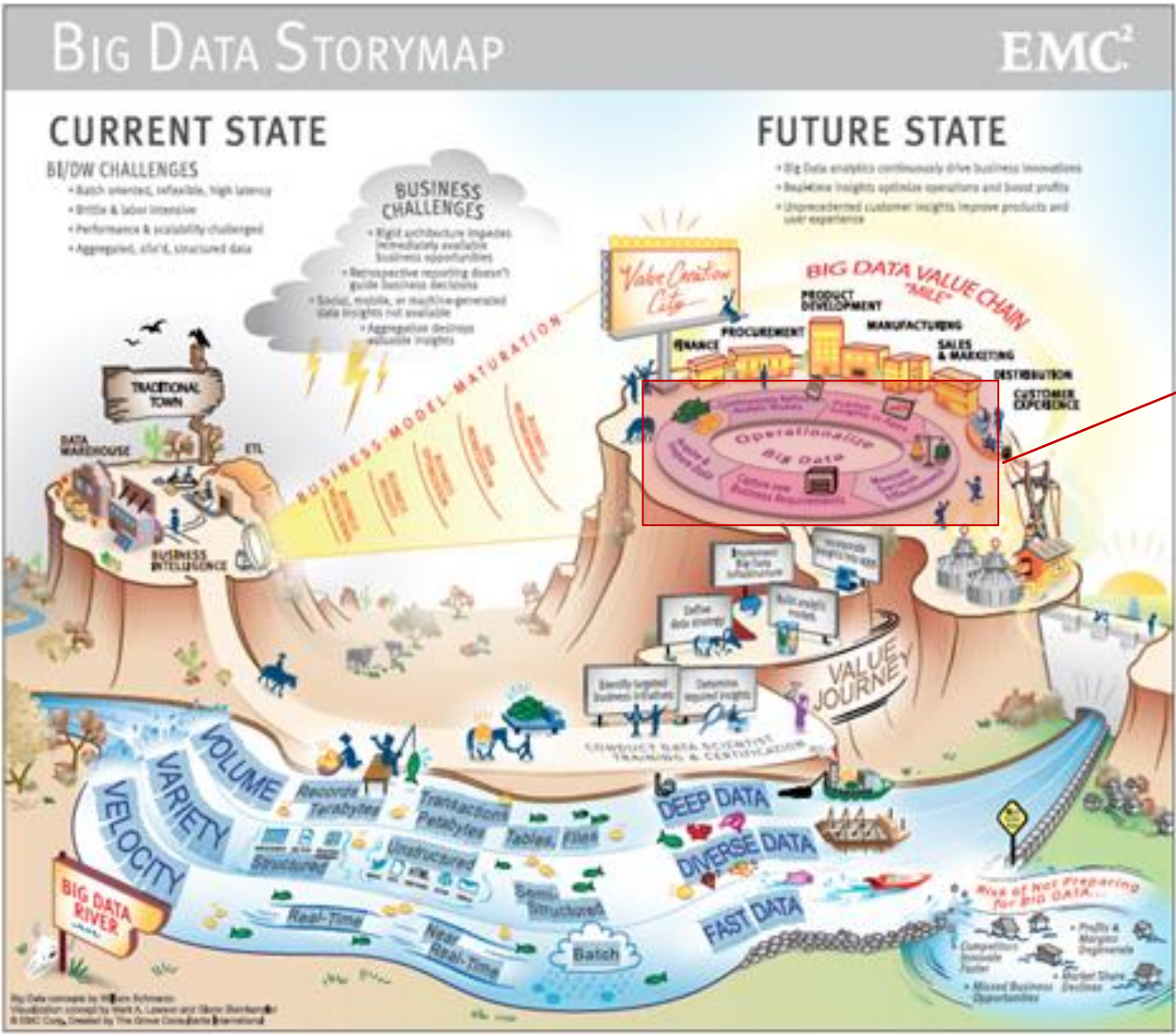


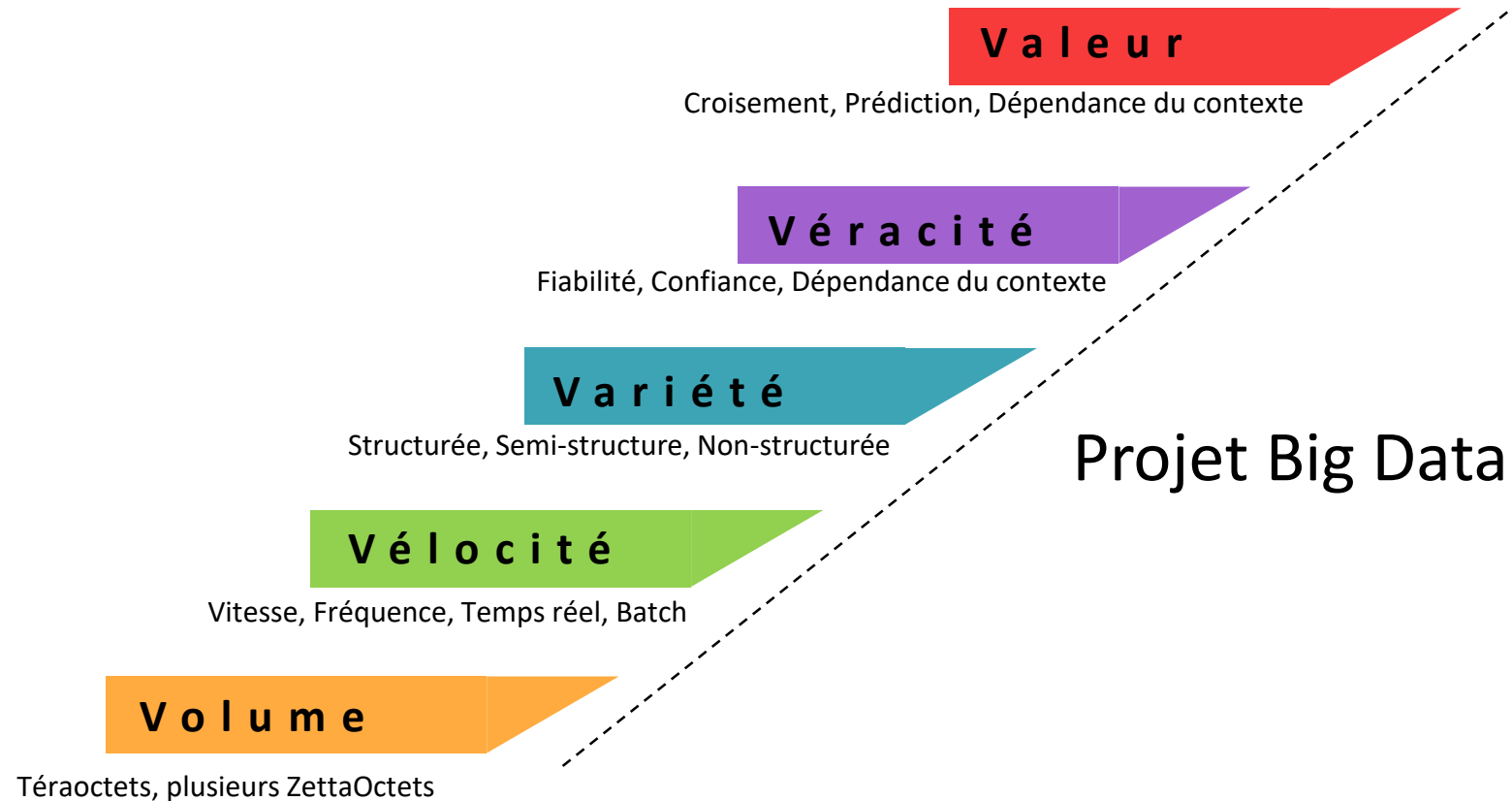
www.domo.com/learn/infographic/data-never-sleeps-8

- Progrès technologiques et la numérisation des business de différents domaines d'application (exemple : WCE, endoscopie 3D).
- Progrès des capacités des systèmes de stockage, de fouille et d'analyse de l'information numérique.
- Explosion du nombre d'applications (exemple : les réseaux sociaux, les vlogs)
- La forte croissance du nombre d'utilisateurs d'internet (internet population) : En avril 2020, l'internet a atteint 59% de la population mondiale et compte désormais 4,57 milliards de personnes (augmentation de 6% par rapport à janvier 2019).



GLOBAL INTERNET POPULATION GROWTH 2014-2020
(IN BILLIONS)



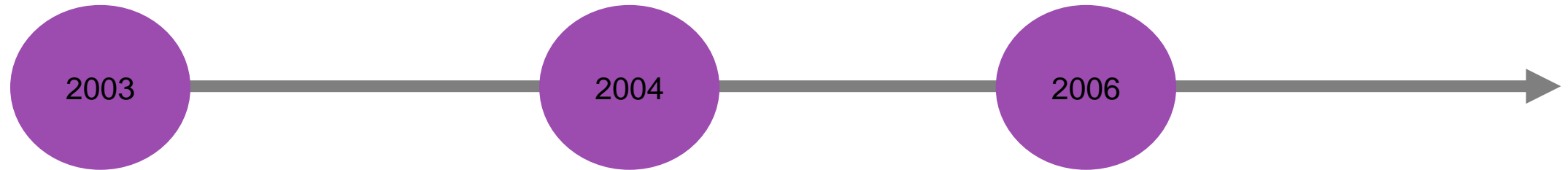


Google File System

Système de fichiers distribués, utilisé en production chez Google, redondant, fonctionnant sur des machines « jetables ».

Big Table

Gestion de base de données basée sur GFS et MapReduce.

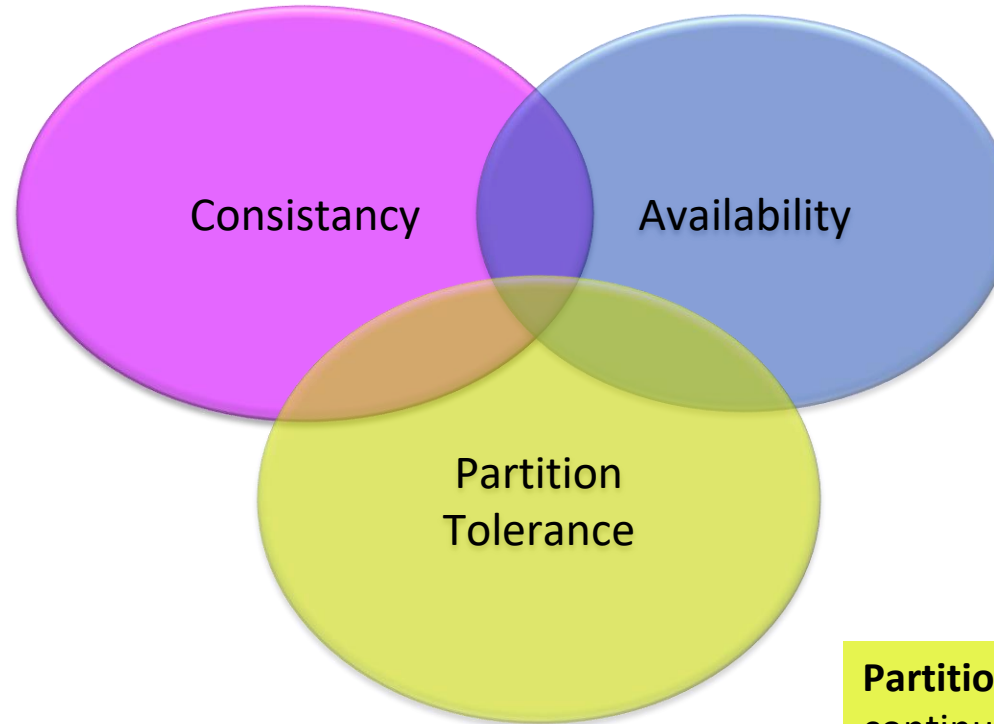


Map Reduce

Algorithme de calculs parallèles et distribués : Permet d'effectuer des traitements sur fichiers de manière distribuée.

❑ **Big Data : Théorème CAP** : Ne peut pas garantir en même temps les 3 propriétés.

Consistency (Cohérence) :
tous les nœuds du système contiennent la même version de données.

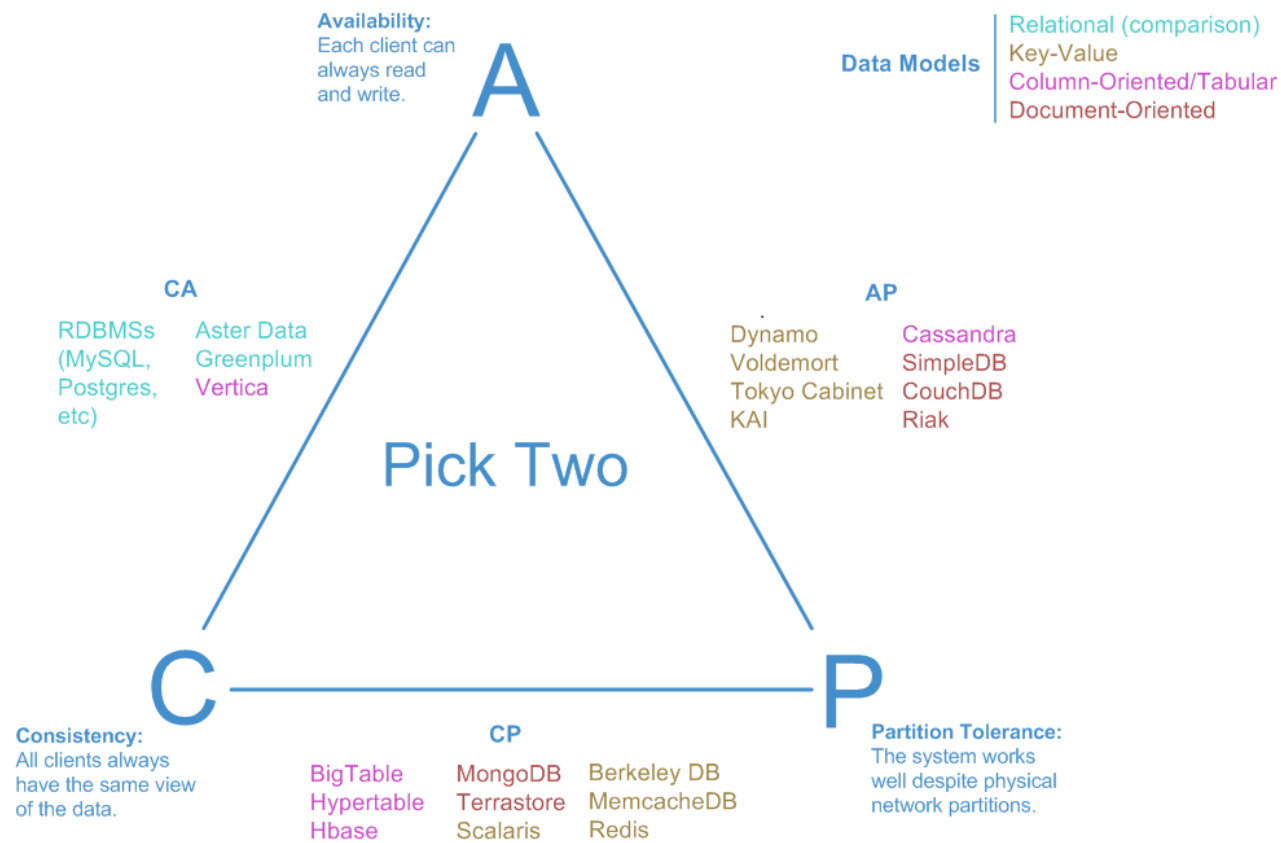


Availability (Disponibilité) :

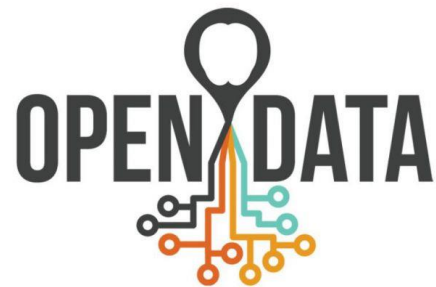
- L'application est accessible à tout instant.
- Toute requête reçoit une réponse.

Partition Tolérance (Résistance au morcellement) :
continuer à fonctionner en cas de morcellement du réseau (sauf panne totale).

Visual Guide to NoSQL Systems



Big Data -Théorème CAP : Ne peut pas garantir en même temps les 3 propriétés.



Données internes	Objets connectés	Open Data	Réseaux sociaux
Seules 10% à 30% des données stockées par les entreprises sont exploitées.	200 milliards d'objets connectés d'ici 2020.	Plus de 19 500 ensembles de données sont publiés.	4,2 milliards d'internautes : 55%, 3,4 milliards d'utilisateurs des réseaux sociaux : 44% 3,2 milliards d'utilisateurs des réseaux sociaux sur mobile : 42%

❑ HADOOP

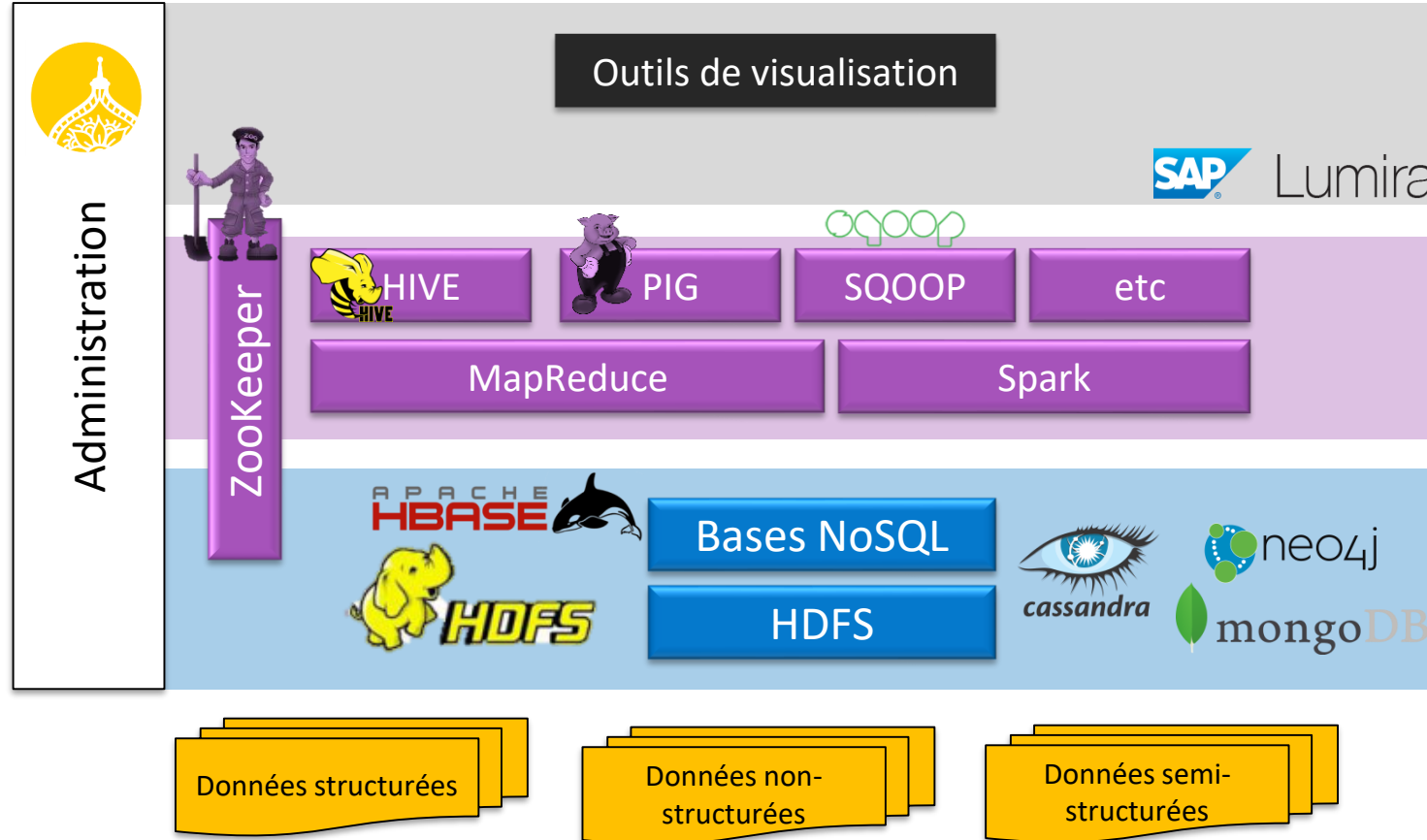
- Projet prioritaire de la fondation Apache.
- Framework de traitements parallèles et distribués des données.
- Permet le passage d'un seul serveur à des milliers de machines.
- Version actuelle 3.3.0: Hadoop 3
- Hadoop Common: gestionnaire des modules Hadoop
- HDFS: un système de fichiers distribués.
- Hadoop HBase: Gestion de la base de données
- Hadoop MapReduce: Traitements parallèles de grands volumes de données.

❑ Distributions commerciales de Hadoop



1. Introduction aux Big Data

1.9. Présentation des technologies au cœur du Big Data



Visualisation des données

Traitement des données

Stockage des données

Sources des données

HADOOP

Mon Fichier : 300Mo

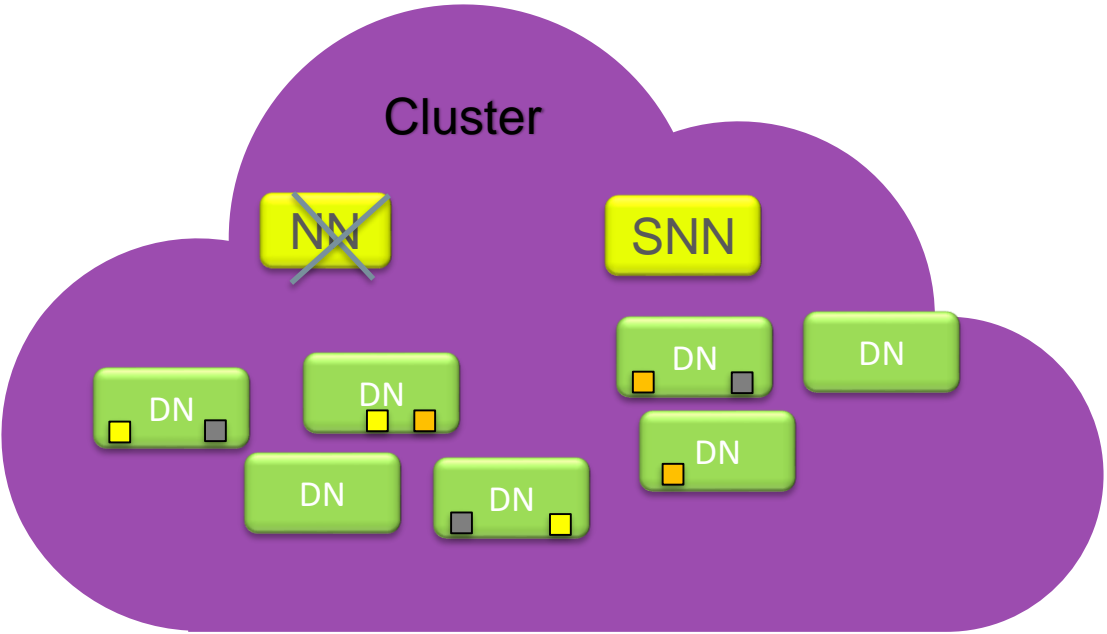
Nombre de blocs ?

128 Mo

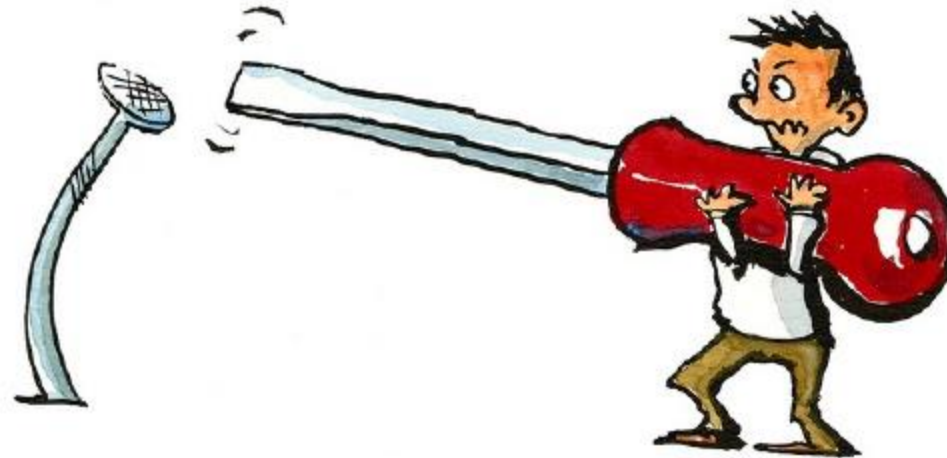
128 Mo

44Mo

Name Node est défaillant



The Law of the Hammer



If the only tool you have is a hammer,
everything looks like a nail.

Abraham Maslow - The Psychology of Science - 1966

The Law of the Relational Database



If the only tool you have is a relational database,
everything looks like a table.

A Walk in Graph Databases - 2012

❑ SGBDR : Propriétés ACID des transactions des bases de données relationnelles

- Atomicité : une transaction est erronée → toutes les transactions sont arrêtées.
- Consistance : contraintes d'intégrités, clés étrangères.
- Isolation : Isoler les transactions.
- Durabilité : Transactions persistantes (commit)



Dans un contexte distribué, difficultés de maintenir les contraintes ACID en maintenant des performances correctes.

❑ BDR : Limites (suite)

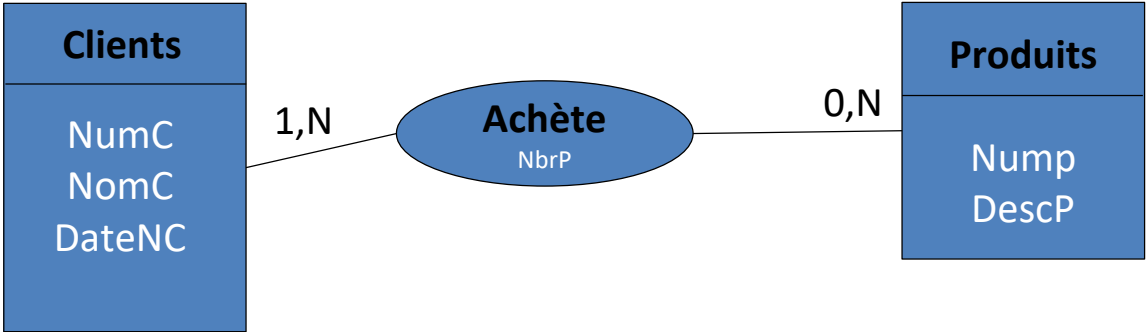
- Peu adaptés aux données non et semi structurées,
- Performance,
- Coût.

➡ Besoin d'un nouveau système de gestion des données qui ne repose pas sur le paradigme relationnel : d'où le "Not Only SQL".



Relâcher certaines contraintes lourdes du paradigme relationnel dans le but de favoriser la distribution : structure des données, langage d'interrogation ou la cohérence des données.

➡ NoSQL est une autre façon de stockage et d'interrogation des données.



Clients

<u>NumC</u>	NomC	DateNC
C01	MARTIN	16/02/1992

NumC → NomC
NumC → DateNC
NumC → NbrP

Achète

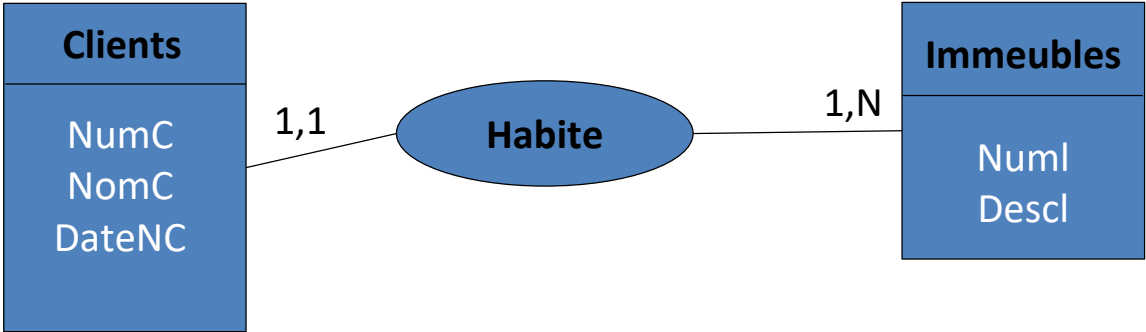
<u>NumC</u>	<u>Nump</u>	NbrP
C01	P01	3

Nump, NumC → NbrP

Produits

<u>Nump</u>	DescP
P01	Téléphone

Nump → DescP



Clients

NumC	NomC	Date	NumI
01	Martin	16/02/1992	I0043

NumC → NomC
NumC → DateNC
~~NumC → NbrP~~

Nump, NumC → NbrP

Produits

<u>Nump</u>	DescP
P01	Téléphone

NumI → DescI

Plan

1. Introduction aux Big Data
2. Les bases de données NoSQL
3. Modélisation des données Big Data
4. La solution MongoDB
5. Manipulation des données avec MongoDB (CRUD)
6. Agrégation des données
7. Jointures et références
8. Recherche d'information

❑ **NOSQL : Not Only SQL, et non pas No SQL,**

❑ **Caractéristiques des systèmes NOSQL :**

- Clustérisable et montée en charge approximativement linéaire.
- Sans schéma ou « schemaless ».
- Non relationnels et distribués.

❑ **Ces systèmes répondent aux priorités suivantes :**

- Evolutivité.
- Distribution des données et des traitements.
- Performance et disponibilité sur la totalité des données.
- Traitement des données non structurées.

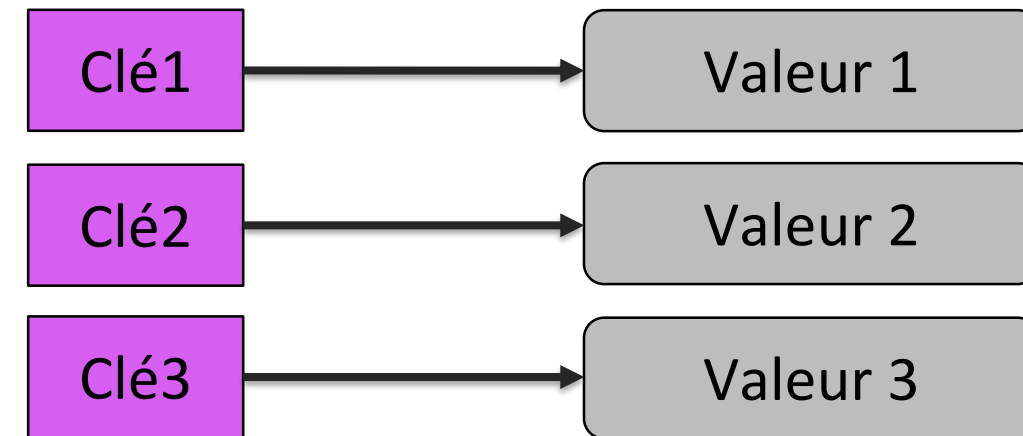
□ 4 Types de base NOSQL

- BD orientée clé-valeur
- BD orientée colonne
- BD orientée document
- BD orienté graphe



❑ BD clé/valeur (1/3)

- Type le plus basique une clé, valeur
 - ✓ Clé : permet l'accès à la ligne.
 - ✓ Valeur : chaîne de caractères, structure complexe.
- Modèle « simpliste » permettant un simple CRUD
 - ✓ Create (key,value)
 - ✓ Read(key)
 - ✓ Update(key,value)
 - ✓ Delete(key)



❑ BD clé/valeur (2/3)

- **Avantage** : une structure de données libre permettant une gestion des objets hétérogènes.
- Exemples d'acteurs : Redis, Dynamo , Riak.



❑ BD clé/valeur (3/3)

Exemple :

User
<u>IdU</u>
CountryU

Base de données relationnelle

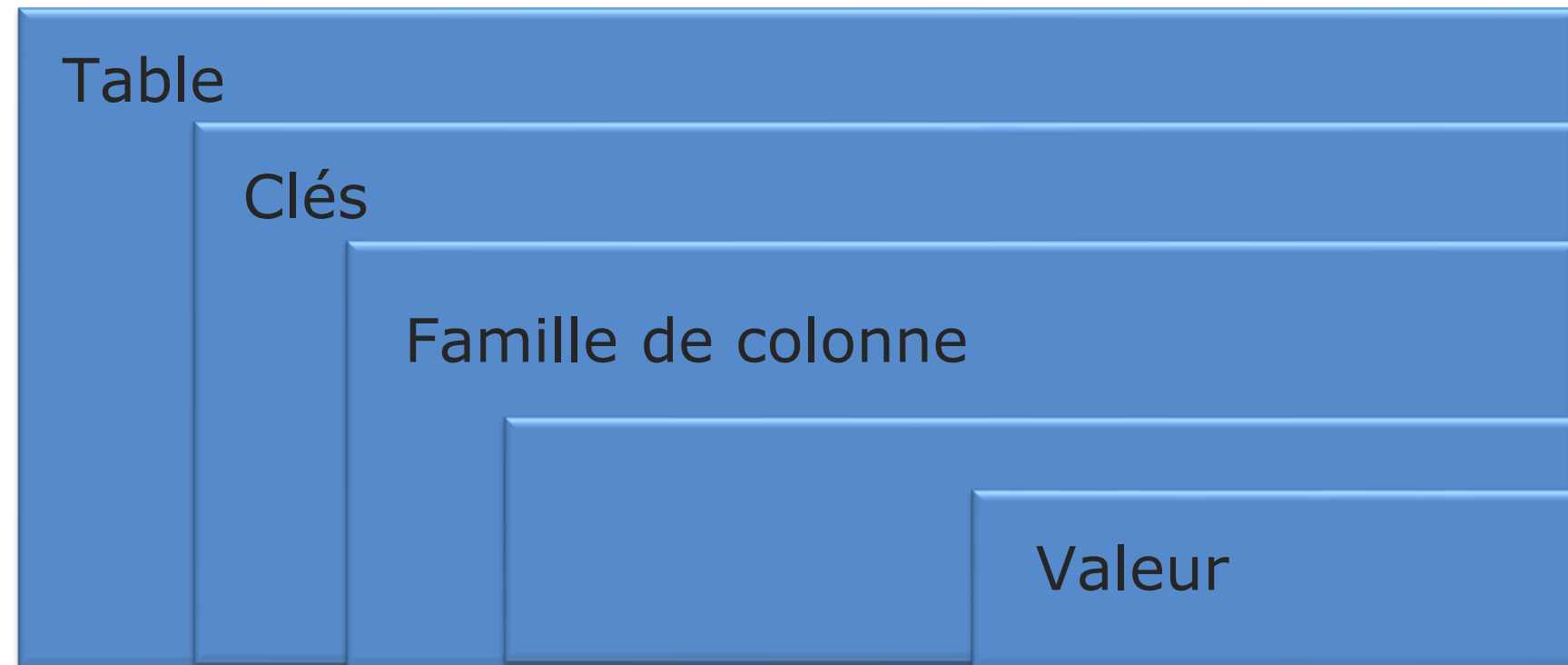
IdU	nomU	PaysU
U1	Martin	France
U2	Dupont	null
...	...	

Base de données Clé / Valeur valeur

U1	U2	...
idU : U1 nameU: Martin countryU: France	idU : U2 nameU: Dupont	...

❑ BD orientée colonne (1/3)

- Basé sur Google et les BigTable
- Le principe est :



❑ BD orientée colonne (2/3)

- On gagne une flexibilité : Il est possible d'ajouter une colonne à n'importe quelle ligne d'une famille de colonnes.
- Gestion des NULL
- Très utilisé dans les traitements d'analyse de données et dans les traitements massifs (lors d'un besoin de MapReduce)
- Exemples d'acteurs: Google Big Table, HBase, Cassandra.



❑ BD orientée colonne (3/3)

Utilisateurs
<u>IdU</u>
nomU
PrénomU

Base de données relationnelle

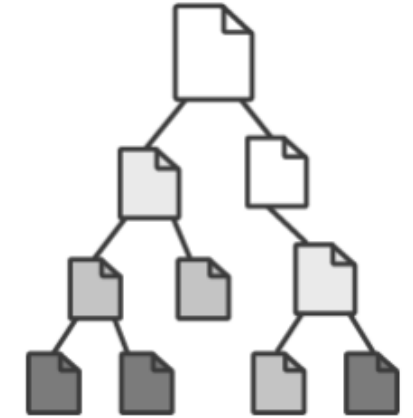
IdU	nomU	PaysU
U1	Martin	France
U2	Dupont	null
...	...	

Base de données orientée colonne

1	IdU : U1	nomU : Martin	PaysU : France
2	IdU : U2	nomU : DUPONT	
...			

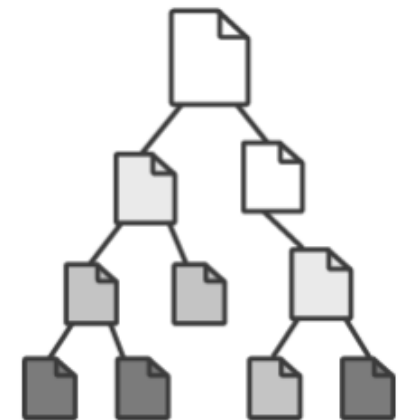
❑ BD orientée document (1/4)

- Type le plus basique une clé = un objet.
- Ce type de base est une évolution de la simple clé/valeur.
- Elle est composée de clé/document.
- Avec un document qui contient un ensemble de couple champ/valeur.
- Les valeurs peuvent être des entiers, chaînes de caractères, date, etc, mais aussi des compositions de couple.
- Lorsque l'on parle de document, il est important qu'il ne s'agît pas ici de document binaire, mais bien des documents lisibles par le moteur.
- Le type de document généralement utilisé est du JSON.



❑ BD orientée document (2/4)

- Deux différences majeures avec la simple clé/valeur:
- Possibilité d'effectuer des requêtes plus complètes qu'une simple lecture de la valeur : capacité à effectuer des requêtes sur le contenu des objets.
- Documents structurés MAIS il n'est pas nécessaire de définir au préalable les champs des documents. On garde donc la flexibilité du modèle.
- L'accès se fait aussi à l'aide de Web Service mais les requêtes sont plus complètes qu'un simple CRUD.
- Ce type de base de données est populaire.



❑ BD orientée document (3/4)

- [Exemples](#) d'acteurs : MongoDB, Amazon SimpleDB, CouchDB, Riak, Lotus Notes.



☐ **BD orientée document (4/4)**

Base de données relationnelle

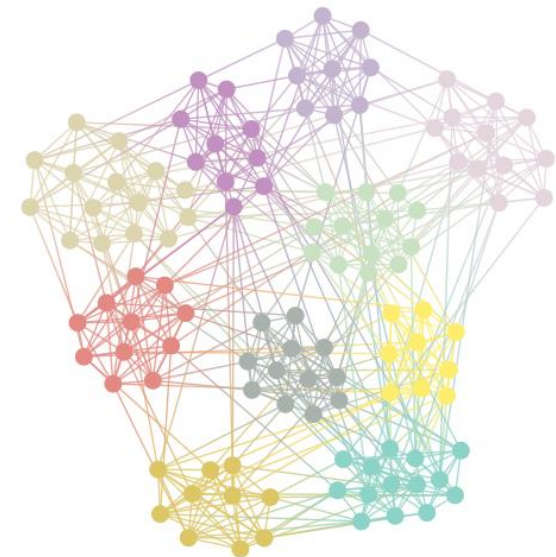
IdU	nomU	PaysU
U1	Martin	France
U2	Dupont	null
...	...	

Base de données orientée document

U1	U2	...
{ "_id": "db000900", "idU" : "U1", "nameU": "Martin", "countryU": "France", }	{ "_id": "db000910", "idU" : "U2", "nameU": "DUPONT", }	{ "_id": "db000911", "idU" : "U2", "adresse" : { "street" : "Bd Starsbourg", "zip" : 31000 "city": "Toulouse" } }

❑ BD orientée graphe (1/4)

- Existait déjà avant l'invention du modèle relationnel.
- Les implémentations modernes et l'utilisation d'un système totalement distribué a permis à ces bases de données de ressortir du lot.
- Basée sur les algorithmes très performants de la théorie des graphes: Dijkstra, problème du voyageur de commerce, etc.
- Composée de nœuds, d'arcs, et de propriétés portées.
- Aucune structure des nœuds ou des arcs.
- La notion même de jointure « n'existe pas ».



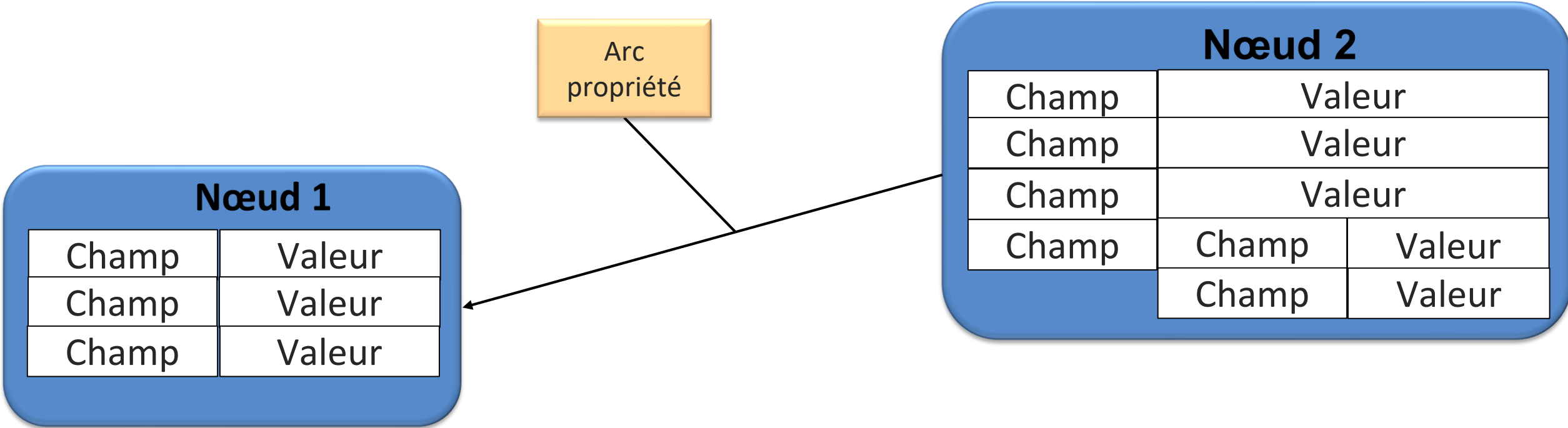
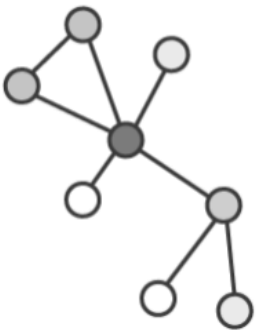
❑ BD orientée graphe (2/4)

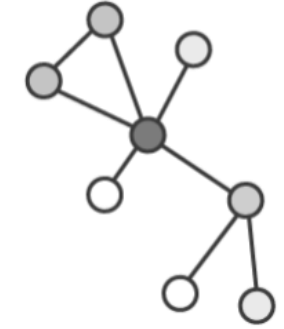
- Existait déjà avant l'invention du modèle relationnel.
- Les implémentations modernes et l'utilisation d'un système totalement distribué a permis à ces bases de données de ressortir du lot.
- Basée sur les algorithmes très performants de la théorie des graphes: Dijkstra, problème du voyageur de commerce, etc.
- Composée de nœuds, d'arcs, et de propriétés portées.
- Aucune structure des nœuds ou des arcs.
- La notion même de jointure « n'existe pas ».



❑ **BD orientée graphe (3/4)**

- Nœud : Même principe qu'un document dans une base de données documentaire.
- Arc : Permet de décrire une relation orientée et porteuse de propriétés entre les objets.





❑ BD orientée graphe (4/4)

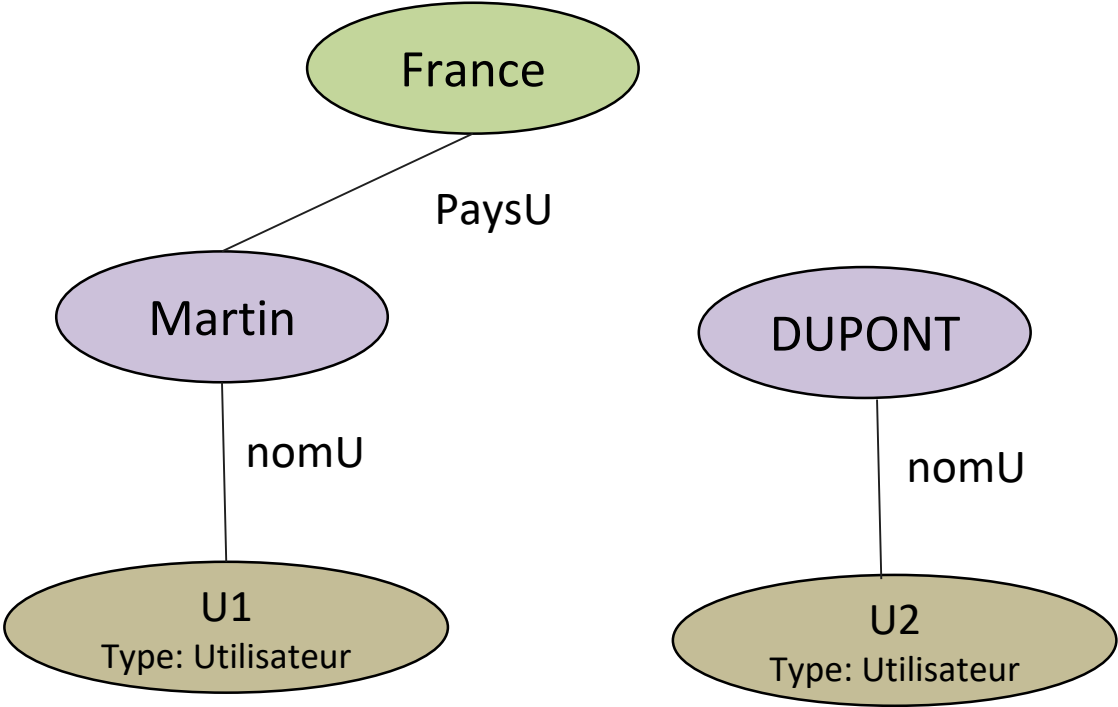
- Peu utilisées : nécessitent de connaissances spécifiques
- Mais très utile pour des besoins :
 - ✓ Découvrir les amis des amis d'un utilisateur qui habitent dans la même ville.
 - ✓ Calculer un trajet d'un point vers un autre selon des critères de temps ou de distance.
 - ✓ Représenter la traçabilité d'un produit.
 - ✓ Etc.
- **Acteur notoire** : Neo4J, OrientDB.



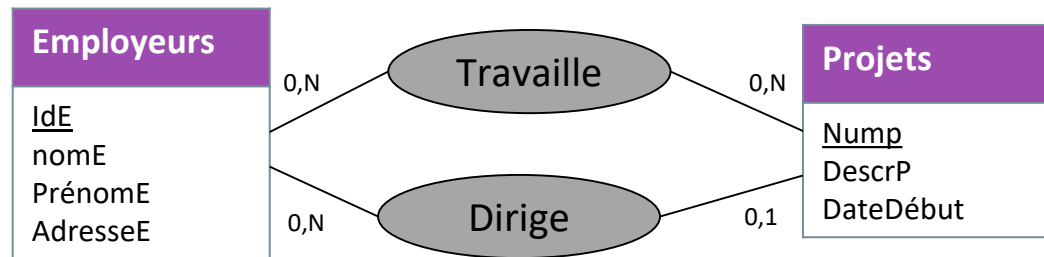
Base de données relationnelle

IdU	nomU	PaysU
U1	Martin	France
U2	Dupont	null
...	...	

Base de données orientée graphe



❑ Exercice



- Base de données relationnelle ?
- Base de données orientée clé/valeur ?
- Base de données orientée colonne ?
- Base de données orientée document ?
- Base de données orienté graphe ?

Plan

1. Introduction aux Big Data
2. Les bases de données NoSQL
3. Modélisation des données Big Data
4. La solution MongoDB
5. Manipulation des données avec MongoDB (CRUD)
6. Agrégation des données
7. Jointures et références
8. Recherche d'information

3. Modélisation des données Big Data

3.1. Les données semi-structurées

```
{ "id":18, "first_name": "Heather", "last_name": "Ramirez", "email": "hramirezh@instagram.com", "country": "Indonesia", "modified": "2015-07-13" },  
{ "id":19, "first_name": "Jason", "last_name": "Sanders", "email": "jsandersi@earthlink.net", "country": "Canada", "modified": "2015-02-25" },  
{ "id":20, "first_name": "Juan", "last_name": "Evans", "email": "jevansj@google.de", "country": "Philippines", "modified": "2015-07-09" },  
{ "id":21, "first_name": "Billy", "last_name": "Tucker", "email": "btuckerk@businessweek.com", "country": "Indonesia", "modified": "2015-02-08" },  
{ "id":22, "first_name": "Fred", "last_name": "Duncan", "email": "fduncanl@smugmug.com", "country": "Brazil", "modified": "2015-03-05" },  
{ "id":23, "first_name": "Daniel", "last_name": "Peterson", "email": "dpetersonm@deliciousdays.com", "country": "Nigeria", "modified": "2014-10-08" },
```



Quelles sont les requêtes que nous pouvons appliquer sur ce document ?


```
[{"NumP":1,"DescP":"Cardify","dateDeb":"11/2/2020","DureeP":55},
{"NumP":2,"DescP":"Zamit","dateDeb":"2/20/2018","DureeP":22},
{"NumP":3,"DescP":"Greenlam","dateDeb":"6/2/2020","DureeP":78},
{"NumP":4,"DescP":"Flowdesk","dateDeb":"10/9/2019","DureeP":28},
{"NumP":5,"DescP":"Zamit","dateDeb":"3/18/2020","DureeP":90},
{"NumP":6,"DescP":"Cardguard","dateDeb":"3/20/2018","DureeP":34},
{"NumP":7,"DescP":"Duobam","dateDeb":"9/28/2018","DureeP":11},
{"NumP":8,"DescP":"Mat Lam Tam","dateDeb":"11/22/2020","DureeP":75},
{"NumP":9,"DescP":"Andalax","dateDeb":"10/21/2018","DureeP":86},
{"NumP":10,"DescP":"Transcof","dateDeb":"3/20/2018","DureeP":77}]
```



Quelles sont les requêtes que nous pouvons appliquer sur ce document ?

```
[{"NumP":57,"NumEmp":61,"dateDeb":"6/14/2019","DureeP":92},  
{"NumP":14,"NumEmp":97,"dateDeb":"2/21/2020","DureeP":22},  
{"NumP":84,"NumEmp":85,"dateDeb":"7/10/2019","DureeP":21},  
{"NumP":64,"NumEmp":93,"dateDeb":"11/6/2018","DureeP":95},  
{"NumP":92,"NumEmp":90,"dateDeb":"11/17/2020","DureeP":75},  
{"NumP":45,"NumEmp":99,"dateDeb":"6/26/2020","DureeP":57},  
{"NumP":82,"NumEmp":53,"dateDeb":"5/24/2019","DureeP":4},  
{"NumP":48,"NumEmp":93,"dateDeb":"8/11/2019","DureeP":34},  
{"NumP":81,"NumEmp":98,"dateDeb":"10/17/2019","DureeP":70},  
{"NumP":1,"NumEmp":51,"dateDeb":"12/12/2020","DureeP":33}]
```



Quelles sont les requêtes que nous pouvons appliquer sur ce document ?



Quelles sont les requêtes que nous pouvons appliquer sur ce document ?

Quelle est la signification de dateD et dateC ? Quelle est la différence entre ces deux champs ?

```
[{"NumP": 57, "NumEmp": 61, "dateD": "6/14/2019", "dateC": "5/08/2021"},  
{"NumP": 14, "NumEmp": 97, "dateD": "2/21/2020", "dateC": "10/11/2021"},  
{"NumP": 84, "NumEmp": 85, "dateD": "7/10/2019", "dateC": "11/12/2020"},  
{"NumP": 64, "NumEmp": 93, "dateD": "11/6/2018", "dateC": "13/04/2019"}]
```

- ❑ Les bases de données NoSQL sont **schemaLess**

Projet.json

```
[  
  {"NumP":1,"DescP":"Cardify"},  
  {"NumP":2,"dateDeb":"2/20/2018"},  
  {"NumP":3,"DescP":"Greenlam","dateDeb":"6/2/2020"},  
  {"NumP":4,"DescP":"Flowdesk","dateDeb":"10/9/2019","DureeP":28},  
]
```

Projets

NumP
DescP
dateDeb
DureeP

Une structure existe mais peut évoluer dans le temps sans aucune prédéfinition de cette structure.

➔ Flexibilité

3. Modélisation des données Big Data

3.1. Les données semi-structurées



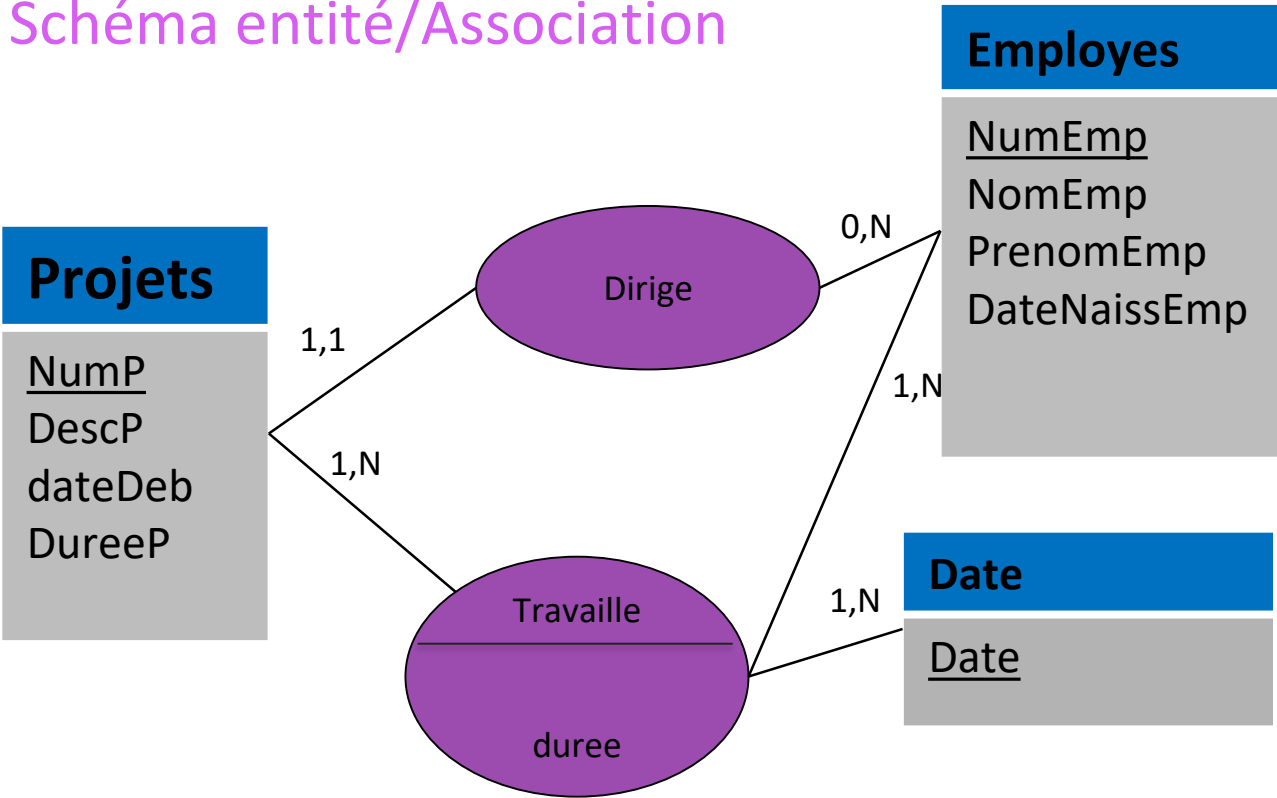
```
[{"NumP":174,"DescP":"Floncon","dateDeb":"1/2/2019","Employes":{"NumEmp":62,"NomEmp":"Martin","PrenomEmp":"Eve","DateNaissEmp":"7/27/1998","travailleH":91,"date":"11/10/1993","duree":18},"Dirige":{"NumEmp":11,"NomEmp":"Marc","PrenomEmp":"Dion","DateNaissEmp":"5/21/1977"}}, {"NumP":116,"DescP":"Nuage","dateDeb":"6/8/2018","Employes":{"NumEmp":53,"NomEmp":"Sarah","PrenomEmp":"« DUPONT","DateNaissEmp":"4/17/1997","travailleH":41,"date":"8/18/1972","duree":70},"Dirige":{"NumEmp":62,"NomEmp":"Martin","PrenomEmp":"Eve","DateNaissEmp":"9/3/1978"}}, {"NumP":187,"DescP":"Curabit","dateDeb":"4/18/2019","Employes":{"NumEmp":1,"NomEmp":"Ferris","PrenomEmp":"DafnÃ©e","DateNaissEmp":"6/22/1973","travailleH":27,"date":"5/3/1992","duree":58},"Dirige":{"NumEmp":91,"NomEmp":"Margarita","PrenomEmp":"Leodora","DateNaissEmp":"1/1/1980"}}, {"NumP":112,"DescP":"« Galaxie","dateDeb":"5/3/2018","Employes":{"NumEmp":11,"NomEmp":"Hulda","PrenomEmp":"VÃ©nus","DateNaissEmp":"1/3/1996","travailleH":52,"date":"2/13/2000","duree":17},"Dirige":{"NumEmp":74,"NomEmp":"Kort","PrenomEmp":"Nevil","DateNaissEmp":"1/2/1993"}}, {"NumP":197,"DescP":"imperdiet","dateDeb":"1/29/2019","Employes":{"NumEmp":57,"NomEmp":"Bellanca","PrenomEmp":"RÃ©jane","DateNaissEmp":"12/1/1990","travailleH":81,"date":"4/8/1995","duree":13},"Dirige":{"NumEmp":94,"NomEmp":"Johnathon","PrenomEmp":"Lyndell","DateNaissEmp":"12/28/1990"}}, {"NumP":105,"DescP":"amet"}]
```

```
[{"NumP":174,"DescP":"Floncon","dateDeb":"1/2/2019","Employes":{"NumEmp":62,"NomEmp":"Martin","PrenomEmp":"Eve","DateNaissEmp":"7/27/1998","travailleH":91,"date":"11/10/1993","duree":18},"Dirige":{"NumEmp":11,"NomEmp":"Marc","PrenomEmp":"Dion","DateNaissEmp":"5/21/1977"}}, {"NumP":116,"DescP":"Nuage","dateDeb":"6/8/2018","Employes":{"NumEmp":53,"NomEmp":"Sarah","PrenomEmp":"< DUPONT","DateNaissEmp":"4/17/1997","travailleH":41,"date":"8/18/1972","duree":70},"Dirige":{"NumEmp":62,"NomEmp":"Martin","PrenomEmp":"Eve","DateNaissEmp":"9/3/1978"}}, {"NumP":187,"DescP":"Curabit","dateDeb":"4/18/2019","Employes":{"NumEmp":1,"NomEmp":"Ferris","PrenomEmp":"DafnÃ©e","DateNaissEmp":"6/22/1973","travailleH":27,"date":"5/3/1992","duree":58},"Dirige":{"NumEmp":91,"NomEmp":"Margarita","PrenomEmp":"Leodora","DateNaissEmp":"1/1/1980"}}, {"NumP":112,"DescP":"< Galaxie","dateDeb":"5/3/2018","Employes":{"NumEmp":11,"NomEmp":"Hulda","PrenomEmp":"VÃ©nus","DateNaissEmp":"1/3/1996","travailleH":52,"date":"2/13/2000","duree":17},"Dirige":{"NumEmp":74,"NomEmp":"Kort","PrenomEmp":"Nevil","DateNaissEmp":"1/2/1993"}}, {"NumP":197,"DescP":"imperdiet","dateDeb":"1/29/2019","Employes":{"NumEmp":57,"NomEmp":"Bellanca","PrenomEmp":"RÃ©jane","DateNaissEmp":"12/1/1990","travailleH":81,"date":"4/8/1995","duree":13},"Dirige":{"NumEmp":94,"NomEmp":"Johnathon","PrenomEmp":"Lyndell","DateNaissEmp":"12/28/1990"}}, {"NumP":105,"DescP":"ametnunc","dateDeb":"5/31/2019","Employes":{"NumEmp":24,"NomEmp":"Ester","PrenomEmp":"LÃ©on","NumEmp":51,"NomEmp":"Godfrey","PrenomEmp":"Kristofer","DateNaissEmp":"5/7/1996"}}]
```

❑ Modélisation d'une collection JSON

```
[
  { "NumP":174, "DescP": "Floncon", "dateDeb": "1/2/201[
    ", "Employes": { "NumEmp":62, "NomEmp": "Martin", "PrenomEmp": "Eve", "DateNaissEmp": "7/27/1998", "travailleurH":91, "date": "11/10/1993", "duree":18}, "Dirige": {
      "NumEmp":11, "NomEmp": "Marc", "PrenomEmp": "Dion", "DateNaissEmp": "5/21/1977" } },
  { "NumP":116, "DescP": "Nuage", "dateDeb": "6/8/2018",
    "Employes": { "NumEmp":53, "NomEmp": "Sarah", "PrenomEmp": "DUPONT", "DateNaissEmp": "4/17/1997", "travailleurH":41, "date": "8/18/1972", "duree":70}, "Dirige": {
      "NumEmp":62, "NomEmp": "Martin", "PrenomEmp": "Eve", "DateNaissEmp": "9/3/1978" } },
  ...
]
```

Schéma entité/Association



❑ Traduction du schéma E/A au **niveau logique**

Schéma entité/Association

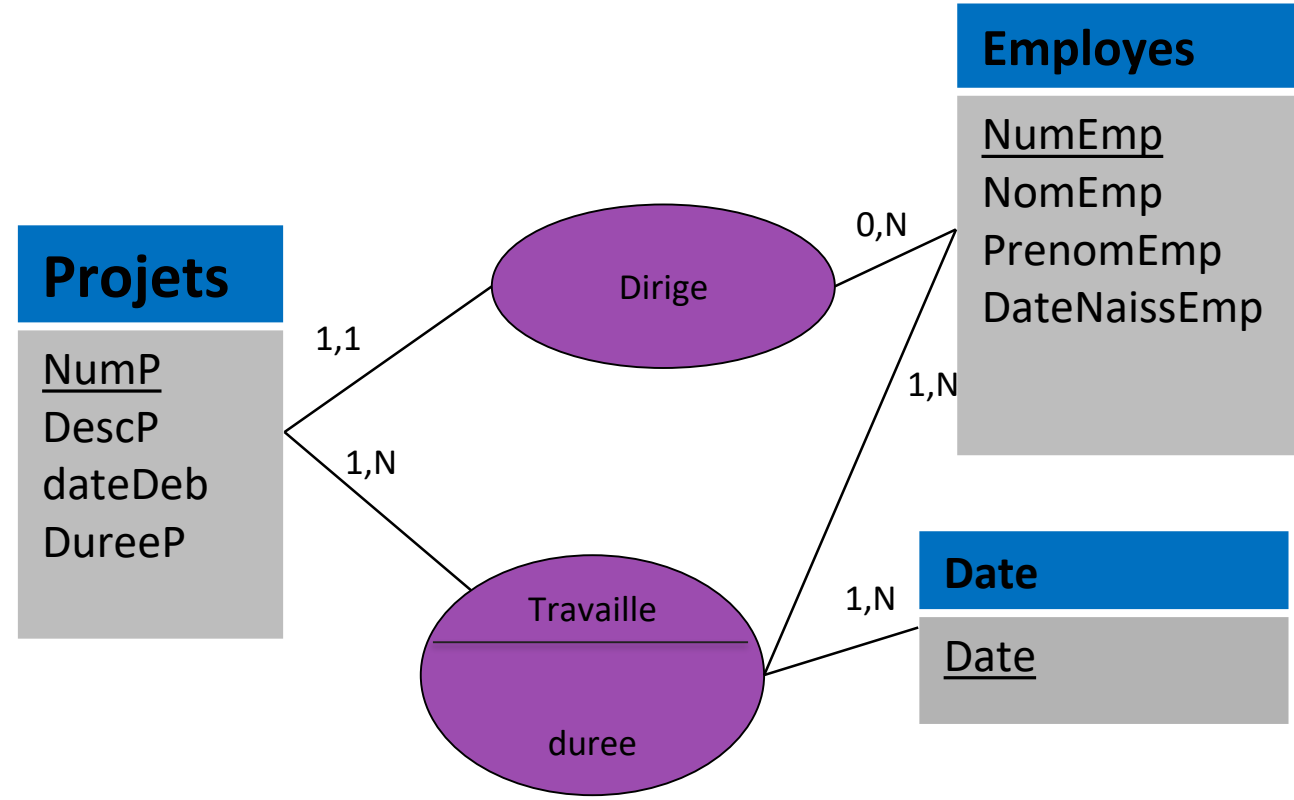


Schéma Relationnel

Projets (NumP, DescP, dateDeb, DureeP, NumEmp*)

Employes (NumEmp, NomEmp, PrenomEmp, DateNaissEmp)

Travaille (NumEmp, NumP, Date, travailleh, duree)

Date (Date)



☐ Les bases de données relationnelles

- La base de données est normalisée : donc pas de redondance.
- Les relations entre les tables se font par des jointures.
- Intégrité des données : cohérence de données.

☐ Les bases de données documentaires

- La base de données n'a pas de contraintes de normalisation : Donc pas de soucis redondance.
- Les relations entre les documents n'est pas un aspect très développé.
- Les dépendances fonctionnelles ne sont pas respectées.
- N'assure pas l'intégrité référentielle.

☐ Les bases de données relationnelles

- Les données sont fortement structurée : Définition du schéma puis insertion des données.
- Les données sont atomiques.

☐ Les bases de données documentaires

- Définition du schéma se fait au même temps que l'insertion des données.
- Les données peuvent être simples ou complexes (un tableau, une liste, un sous-document, etc.).



Dans les deux types de bases de données : la définition de schéma est importante

❑ Ecrire la requête permettant d'afficher le nombre d'employés par projets où les directeurs ont plus de 50 ans.

```
{ "NumP":174, "DescP":"Floncon", "dateDeb":"1/2/2019", "Employes":{ "NumEmp":62, "NomEmp":"Martin", "PrenomEmp":"Eve", "DateNaissEmp":"7/27/1998", "travailleH":91, "date":"11/10/1993", "duree":18}, "Dirige":{ "NumEmp":11, "NomEmp":"Marc", "PrenomEmp":"Dion", "DateNaissEmp":"5/21/1977"}},  
  
{ "NumP":116, "DescP":"Nuage", "dateDeb":"6/8/2018", "Employes":{ "NumEmp":53, "NomEmp":"Sarah", "PrenomEmp":"« DUPONT", "DateNaissEmp":"4/17/1997", "travailleH":41, "date":"8/18/1972", "duree":70}, "Dirige":{ "NumEmp":62, "NomEmp":"Martin", "PrenomEmp":"Eve", "DateNaissEmp":"9/3/1978"}},  
  
{ "NumP":187, "DescP":"Curabit", "dateDeb":"4/18/2019", "Employes":{ "NumEmp":1, "NomEmp":"Ferris", "PrenomEmp":"DafnÃ©e", "DateNaissEmp":"6/22/1973", "travailleH":27, "date":"5/3/1992", "duree":58}, "Dirige":{ "NumEmp":91, "NomEmp":"Margarita", "PrenomEmp":"Leodora", "DateNaissEmp":"1/1/1980"}},  
  
{ "NumP":112, "DescP":"« Galaxie", "dateDeb":"5/3/2018", "Employes":{ "NumEmp":11, "NomEmp":"Hulda", ""
```

➡ Nombre de jointures importants !

❑ Les avantages des bases de données NoSQL

- Schéma flexible.
- Données variées : différents types de données.
- Stockage distribué et scalable.
- Traitement rapide.
- Moins de jointures : plus simple à traiter.

❑ Les inconvénients des bases de données NoSQL

- Schéma risque d'être compliqué à interpréter : on ne peut pas contrôler le contenu de la base.
- Redondances des données : plus de risque d'erreurs
- Jointure manuelle.
- Absence de l'intégrité référentielle.

- Données semi-structurée et non-structurée ?
- Pas de problème de stockage ?
- Données mises à jour régulièrement ?
- Base de données cohérente ?
- Traiter des gros volumes de données en temps réel ?
- Structure de données claire ?



☐ Bases de données NoSQL

- Données semi-structurée et non-structurée.
- Pas de problème de stockage.
- Traiter des gros volumes de données en temps réel.

☐ Base de données relationnelle

- Structure de données claire.
- Base de données cohérente.
- Données mises à jour régulièrement.



NoSQL = Not Only SQL

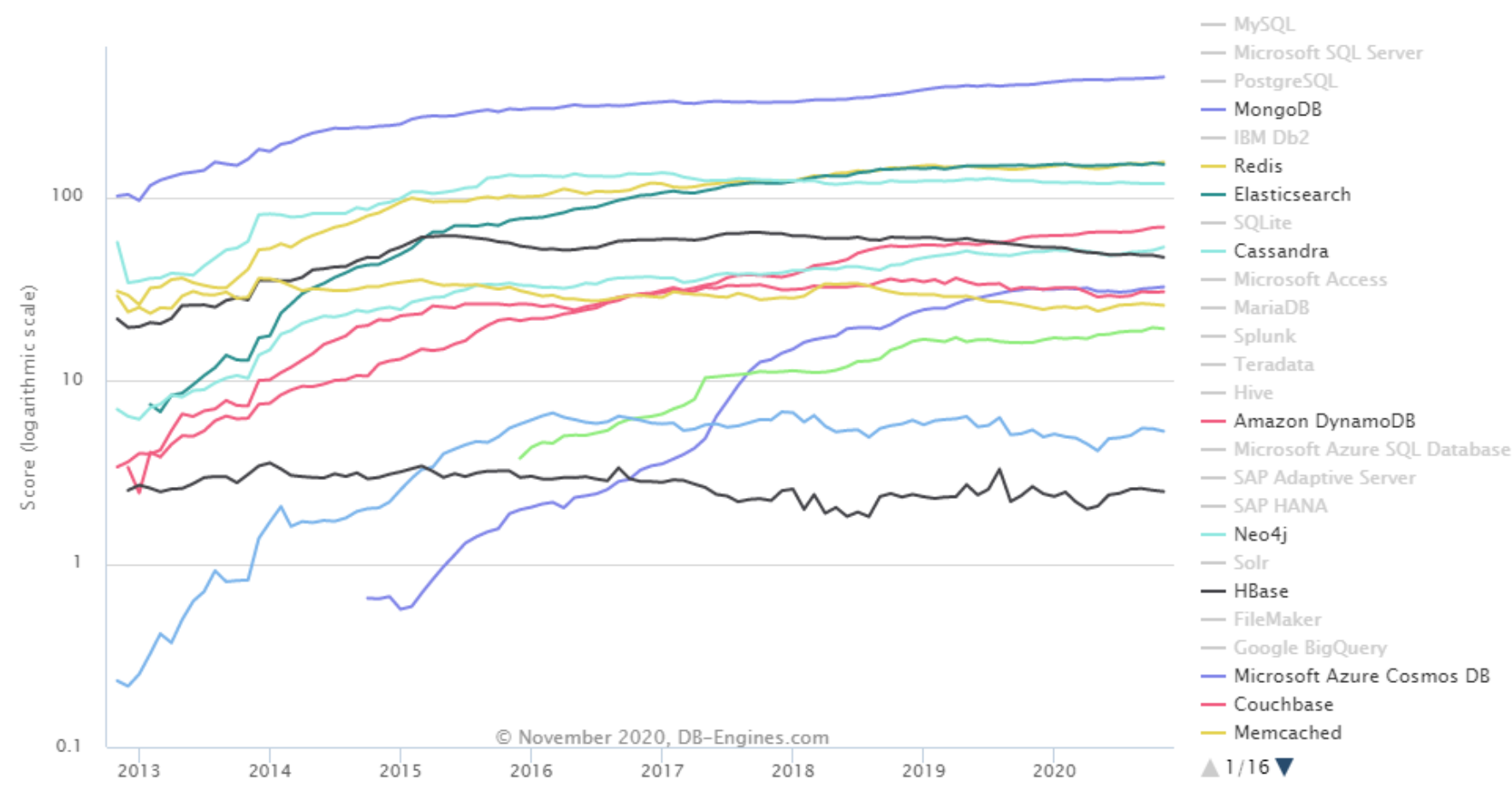
➡ Ces deux types de bases de données sont complémentaires.

Plan

1. Introduction aux Big Data
2. Les bases de données NoSQL
3. Modélisation des données Big Data
4. La solution MongoDB
5. Manipulation des données avec MongoDB (CRUD)
6. Agrégation des données
7. Jointures et références
8. Recherche d'information

4. La solution MongoDB

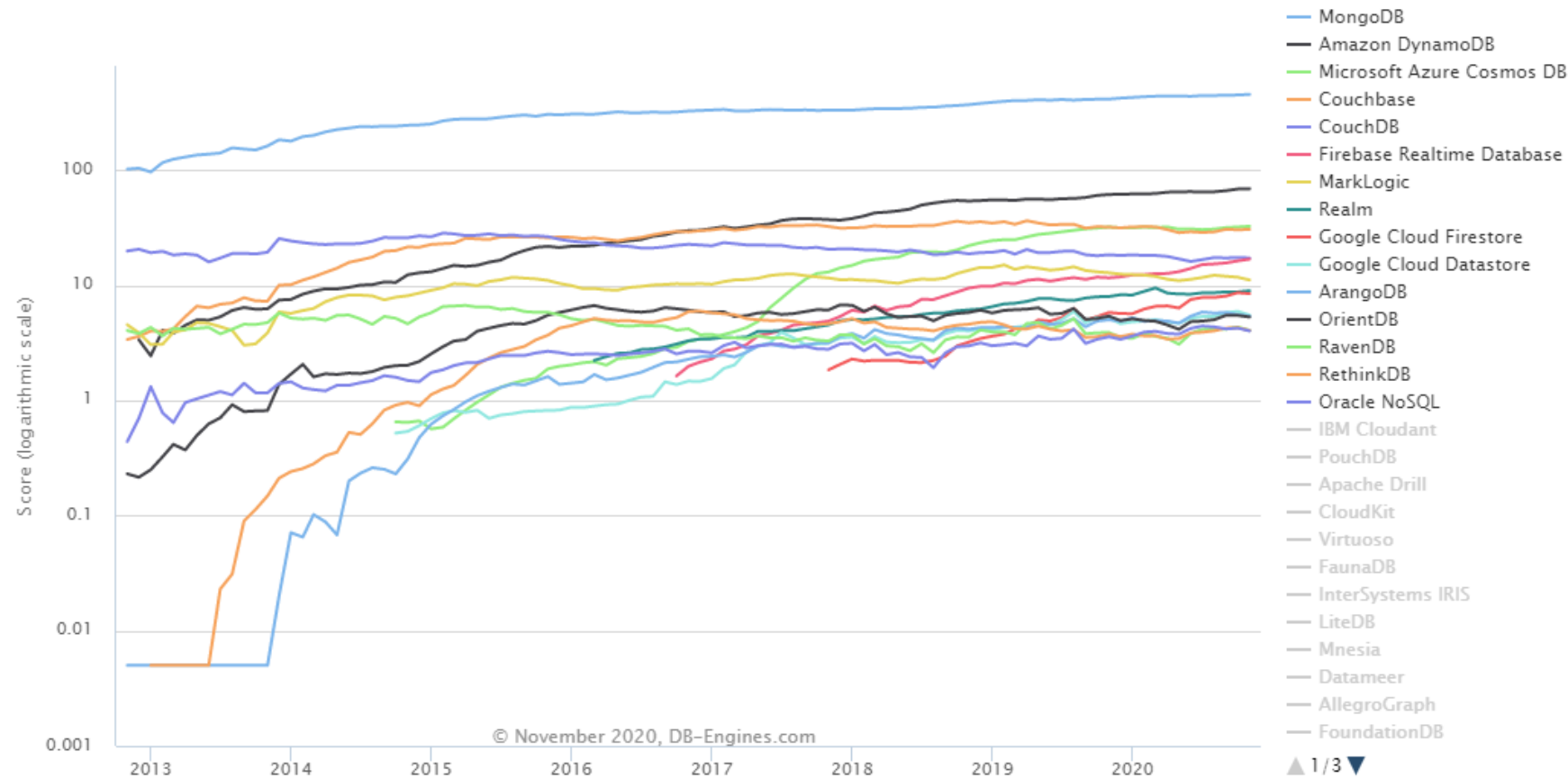
4.1. Popularité des SGBD NoSQL : toutes les familles NoSQL



Classement de DB-Engines (db-engines.com)

4. La solution MongoDB

4.2. Popularité des SGBD NoSQL : bases orientées document

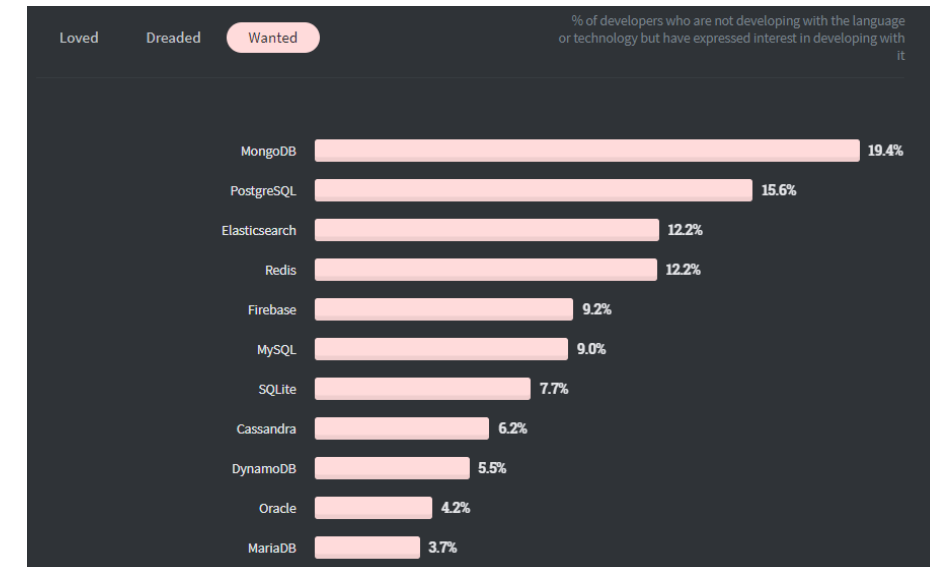
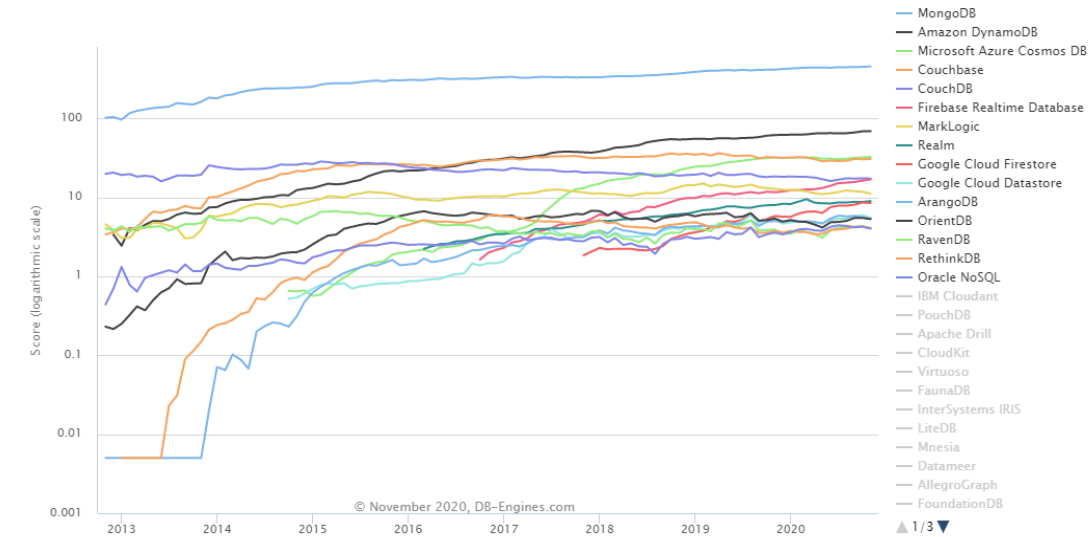


Classement de DB-Engines (db-engines.com)

4. La solution MongoDB

4.3. Popularité de MongoDB

- La BD orientée document la plus populaire d'après le classement de *DB-Engines*
- Nommée pour la 4^{ème} année consécutive comme étant la BD la plus populaire par les développeurs d'après l'enquête *Stack Overflow* (catégorie : [Most wanted](#)).
- Nommée Leader par les rapports *Forrester Wave™: Big Data NoSQL, Q1 2019* et par *Forrester Wave: Database-As-A-Service, Q2 2019*





4. La solution MongoDB

4.4. Avantages et fonctionnalités clés de MongoDB

❑ Richesse du langage d'interrogation

- Agrégation de données
- Filtrage et recherche du contenu textuel
- Possibilité d'exécuter des requêtes spatiales sur une collection contenant des formes et des points géospatiaux.
Exemple d'application : localiser les restaurants dans un périmètre spatiale bien déterminé.

❑ Disponibilité des données grâce à un processus de réplication appelé **Replica Set** permettant :

- Une bascule (*failover*) automatique 
- Une redondance de données 

❑ Sécurité et performances

(sous réserve de ne pas utiliser MongoDB comme système relationnel)

4. La solution MongoDB

4.4. Avantages et fonctionnalités clés de MongoDB

- ❑ Support de **plusieurs moteurs de stockage (MS)**
 - WiredTiger MS (par défaut)
 - In-Memory MS (disponible uniquement pour la version MongoDB Enterprise)

- ❑ **Scalabilité et flexibilité**
 - Supporter 29 langages de programmation
 - Prise en compte de l'homogénéité des données
 - Modélisation flexible et dynamique des données
 - Possibilité d'opérer sur une infrastructure multicloud
 - Adaptée à plusieurs types d'applications (transactionnel, opérationnel et analytique)
 - Méthode de *sharding*

- ❑ **Support** de bonne qualité avec des réponses concises et précises et des propositions adaptées aux problèmes exposés.

SGBDR	MongoDB
Entité	Entité
Relation	Relation
Base de données	Base de données
Table	Collection
Attribut / Colonne	Champ
Ligne	Document
Jointure	Imbrication / Liens

Plan

1. Introduction aux Big Data
2. Les bases de données NoSQL
3. Modélisation des données Big Data
4. La solution MongoDB
5. Manipulation des données avec MongoDB (CRUD)
6. Agrégation des données
7. Jointures et références
8. Recherche d'information

□ JSON

Abréviation: *JavaScript Standard Object Notation*

❖ Format JSON

- ✓ Encapsulé dans des accolades `{ }`
- ✓ Séparer la clé (champ) et sa valeur par deux points disposés verticalement `:`
- ✓ Séparer chaque pair *clé : valeur* par une virgule `,`
- ✓ Les clés doivent être placés entre deux guillemets : `"clé"`

- ✓ Valeurs possibles:
 - Un type scalaire (*String, Boolean, Number, Array*).
 - Une liste de valeurs [...].
 - Des documents imbriqués.

□ JSON

```
{
  "nickname" : "Eseo",
  "followers" : 5865,
  "following" : 81,
  "gender" : "F",
  "profil_Photo" : {
    "type" : "jpeg",
    "link" : "www.instagram/profil?87357286",
    "size" : 6379
  },
  "tags" : [
    "food",
    "street art"
  ]
}
```

Sous-document :
un document imbriqué

Liste

Exemple de fichier JSON valide.

□ JSON

- + User friendly
- + Facile à lire (machines, humains)
- + Connue par beaucoup de développeurs

- Basé sur du texte (analyse syntaxique difficile).
 - Encombrant en termes de stockage (espace mémoire).
 - Limité en termes de types de données supportés
- ➡ Le format **BSON** a été inventé pour adresser ces limites du format JSON.

❑ BSON

Abréviation: **Binary JSON**

But : Une représentation binaire pour stocker les données en format JSON.

+ Optimisé pour :

- ✓ Vitesse.
- ✓ Charge de stockage.
- ✓ Flexibilité.
- ✓ Haute Performance.
- ✓ Faciliter la communication des données entre les bases MongoDB et les différentes applications (technologies et langages différents).

```
_id[0a2...>E0<00  
saleDate"0uHLitems00mnameprinter  
papertags%0office1stationaryprice0  
<0quantity1rnamenotepadtags00office  
1writing2schoolprice0  
<0quantity20namepenstagsB0writing1o  
ffice2school3stationaryprice0<0qua  
ntity3pname  
backpacktags-0school1travel2kidspri  
ce[<0quantity4rnamenotepadtags00off  
ice1writing2schoolprice7<0quantity5  
xname  
envelopestags40  
stationary1office2generalprice0<0qu  
antity6xname
```

Exemple de fichier en format BSON illustrant comment les données sont stockées en mémoire.

	JSON	BSON
Encodage	UFT-8 String	Binaire
Types de données supportés	String, Boolean, Nombre, Array	String, Boolean, Nombre(Integer, Long, Float,...), Array, Date, Timestamp, Object, Null, DBPointer, etc.
Lisibilité	Humaine Machine	Machine uniquement

L'opération d'exportation ou l'importation dépend du format des données (JSON, BSON) et du besoin (analyse, ou communiquer les données à une autre application).

❑ JSON

- *mongoimport* : importation des fichiers JSON ↵
- *mongoexport* : exportation des fichiers JSON ↗

❑ BSON

- *mongorestore* : importation des fichiers BSON ↵
- *mongodump* : exportation des fichiers en format BSON ↗

- Afficher la liste toutes les bases de données

```
# afficher toutes les bases de données  
> show dbs
```

✓ A ce stade, la commande doit retourner "test", qui est la base de données par défaut.

- Pour changer de base de données, utilisez la commande **use <db>**

NB: Pas besoin de créer la BD avant d'y accéder. *MongoDB* crée automatiquement la BD lorsque vous demandez de l'utiliser via la commande "use".

```
> use exemples  
# vérifier si la BD actuelle est bien exemples  
> db exemples
```

- Afficher toutes les collections d'une base de données (après commande use)

```
> show collections
```

- Afficher **tous** les documents d'une collection

```
> db.nomcollection.find() // ou find({})
```



Besoin d'indiquer le nom de la base lors de l'interrogation d'une collection

- Au cas où le shell MongoDB n'accepte pas le nom d'une collection*, utiliser la commande suivante:

```
> db.getCollection("nom_collection").find()
```

* Exemples: le nom de la collection contient un espace, un trait d'union (-), commence par un nombre, ou similaire au nom d'une fonction intégrée (e.g. stats)

- Afin de formater le résultat, utilisez la méthode **pretty()**

```
> db.nomcollection.find().pretty()
```

- Pour parcourir le curseur, qui pointe (i.e. c'est un pointeur) sur le résultat retourné par une requête, utiliser la commande suivante :

```
> it // pour dire "iterate"
```

NB: cette commande est pratique lorsqu'une requête retourne un nombre important de lignes (centaine par exemple) et que le shell MongoDB ne peut pas les afficher toutes en une seule fenêtre.

- Compter le nombre de documents dans une collection

```
> db.nomcollection.count()
```

- Effectuer une requête sur une collection

```
> db.nomcollection.find( { "cle1": "valeur1", "cle2": "valeur2", .. } )
```

- Compter le nombre de documents d'un curseur (retournés par une requête)

```
> db.nomcollection.find( { "cle1": "valeur1", "cle2": "valeur2", .. } ).count()
```


Si nous voulons spécifier les attributs à retourner, il faut utiliser **un document de projection** que l'on passe comme paramètre **optionnel** à la méthode *find()*, comme suit:

```
> db.collection.find(<requête de filtrage des documents>, <document de projection>)
```

- <attribut> : 1 ➡ inclure l'attribut dans les documents retournés
- <attribut> : 0 ➡ exclure l'attribut des documents retournés



On ne peut pas combiner l'utilisation des 0 et 1. Il faut donc utiliser soit les 0 pour exclure des champs ou les 1 pour les inclure.



La seule exception permettant de combiner des 0 et 1, est l'exclusion de l'identifiant en incluant d'autres champs. En effet l'ID est inclus par défaut.

❑ Insertion d'un seul document à la fois

```
> db.nomcollection.insertOne(<document>)
```

- Exemple: insertion d'une nouvelle personne

```
> db.nomcollection.insertOne({"fonction": "Toto", "salaire": 55})
```

❑ Insertion de plusieurs documents à la fois sous forme de liste de documents

```
> db.nomcollection.insertMany( [ {document1}, {document2}, .. , {documentN} ] )
```

- ❑ Insertion d'un seul ou plusieurs documents à la fois en utilisant la même méthode.

```
> db.nomcollection.insert(<document ou liste de documents>)
```

- ❑ L'ordre d'insertion :

```
> db.nomcollection.insert( [ {doc1}, {doc2},{doc3} ] )
```

- **Par défaut**, l'ordre d'insertion est celui présenté dans la commande d'insertion: *doc1*, puis *doc2*, puis *doc3*.
- Sinon on souhaite désactiver la prise en compte de l'ordre des documents pour l'insertion :

```
> db.nomcollection.insert( [liste documents], {"ordered" : false})
```

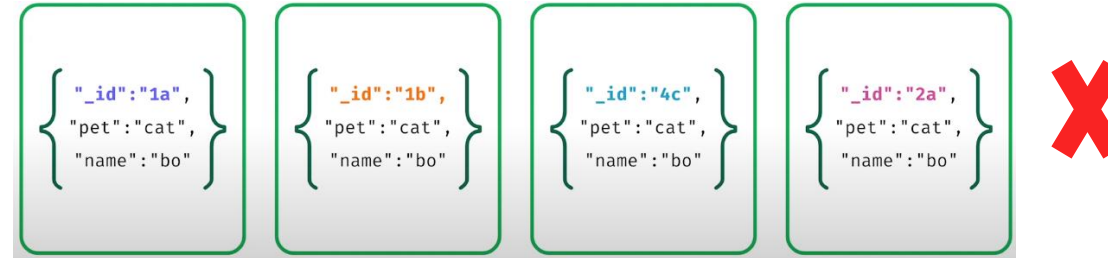
⚠ Si l'ordre d'insertion est *true* (par défaut) → toute erreur de duplication d'_id entraîne l'arrêt de l'opération d'insertion (le reste des documents ne seront pas insérés même si leur _id est unique).

⚠ Si l'ordre d'insertion n'est pas activé ("*ordered*": *false*) → tout document ayant un _id unique sera inséré dans la collection.

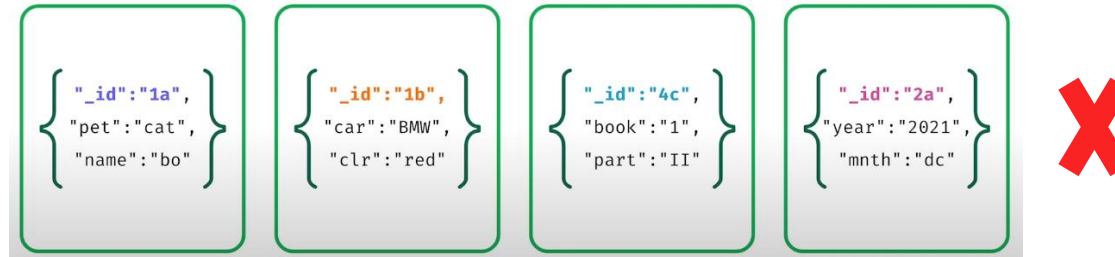


- Est-t-il possible d'insérer **le même document** plusieurs fois ?
- Est-t-il possible d'insérer des documents ayant des **champs complètement différents** ?
- **Pourquoi** ?

Il est possible d'insérer des documents ayant les **même champs/valeurs** du moment où l'ID est différent



Il est également possible d'insérer des documents de **structures différentes**



⚠️ Même si c'est possible, ces deux cas extrêmes ne permettent pas une gestion optimisée de l'espace mémoire, et ne reflètent pas une organisation optimisée des données.

⚠️ Nous allons voir les bonnes pratiques pour l'organisation des données dans le chapitre de modélisation.

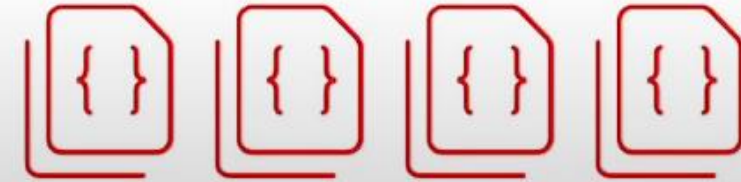
`updateOne()`

`findOne()`



`updateMany()`

`find()`



- Le même principe et la même différence entre : *findOne()* VS *find()*

```
> db.nomcollection.updateMany({"cle": "valeur"}, {MAJ})
```

- **updateOne()** : Mettre à jour un seul document.
- **updateMany()** : Mettre à jour tous les documents correspondants aux critères de filtrage.

Exemple: ajouter 5 ans à l'âge des personnes habitants à Paris.

```
> db.personnes.updateMany({"adresse.ville":"Paris"}, {"$inc":{"age":5}})  
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

Lien opérateurs de MAJ : <https://docs.mongodb.com/manual/reference/operator/update/>

❑ Suppression de documents

`deleteOne()`

`updateOne()`

`findOne()`



`deleteMany()`

`updateMany()`

`find()`



- Le même principe et la différence entre :
 - ✓ `updateOne()` VS `updateMany()`
 - ✓ `findOne()` VS `find()`

❑ Suppression de documents avec : `deleteOne()`

```
deleteOne("_id":11)
```

```
updateOne("_id":11)
```

```
findOne("_id":11)
```



```
deleteMany()
```

```
updateMany()
```

```
find()
```



- Est-t-il judicieux d'utiliser `deleteOne()` ?
- Si oui comment ?

❑ Suppression de documents avec : delete()

```
deleteOne("_id":11)
```

```
updateOne("_id":11)
```

```
findOne("_id":11)
```



```
deleteMany()
```

```
updateMany()
```

```
find()
```



```
> db.nomcollection.deleteMany({"champN": "valeurN"})
```

❑ Suppression de collection

```
> db.nomcollection.drop()
```



NB: Supprimer toutes les collections d'une base de données entraîne la suppression de la BD en question. Ainsi, BD n'apparaît plus une fois on exécute la commande : `show dbs`

```
{ <cle> : { <opérateur> : <valeur> } }
```

- **\$eq** = **EQ**qual to (égal à)
- **\$gt** = **G**reater **T**han (supérieur à)
- **\$gte** = **G**reater **T**han or **E**qual (supérieur ou égale à)
- **\$ne** = **N**ot **EQ**qual to (différent de)
- **\$lt** = **L**ess **T**han (inférieur à)
- **\$lte** = **L**ess **T**han or **E**qual (supérieur ou égale à)

✓ Exemple: afficher les produits qui pèsent moins que 500 g

```
db.produits.find( { "poids" : { "$lte" : 500 } } )
```

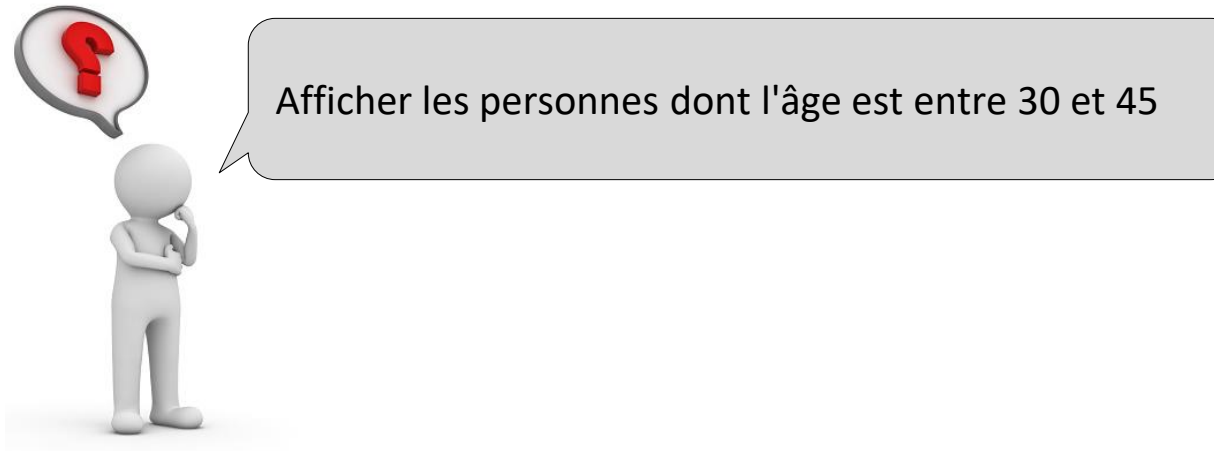
- **\$and** : Matcher toutes les clauses/conditions de la requête.
- **\$or** : Au moins une condition de la requête matche.
- **\$nor** : Echec de matcher avec toutes les options données.
- **\$not** : Négation de la condition de requête.

AND, OR, NOR

```
{ <"$opérateur"> : [ { <condition1> }, { <condition2> }, ... ] }
```

NOT

```
{ "$not" : { <condition> } }
```



Question : Afficher les personnes dont l'âge est entre 30 et 45

Solution 1

```
> db.personnes.find({ "$and" : [{"age" : { "$gt" : 20}}, {"age" : { "$lt" : 45}}]})
```

Solution 2

```
> db.personnes.find({ "age" : { "$gt" : 20}}, { "age" : { "$lt" : 45}})
```

Solution 3
(la meilleure)

```
> db.personnes.find({ "age" : { "$gt" : 20, "$lt" : 45}})
```

- En présentant les opérateurs de comparaison, nous avons comparé les valeurs des champs à des valeurs bien définies (principalement des chaînes de caractères ou des nombres).



Peut-on effectuer une comparaison entre **les champs** (i.e. clés) **d'une même collection** ?


```
{ "$expr" : { <expression> } }
```

- ❑ L'opérateur **\$expr** permet l'utilisation :
 - Des expressions d'**agrégation**,
 - Des **variables**,
 - Des **expressions conditionnelles**.

- Document exemple d'une collection contenant des données de location de vélos à NY-EU

```
{// start sample document

  "_id": {
    "$oid": "572bb8222b288919b68abf82"
  },

  "bikeid": 14785,

  "end station id": 433,
  "end station name": "E 13 St & Avenue A",
  "start station id": 518,
  "start station name": "E 39 St & 2 Ave",
  "tripduration": 812,
  "usertype": "Subscriber"
}// end sample document
```

Afficher le nombre de locataires ayant pris et rendu leurs vélos à la même station

- Document exemple d'une collection **trips** contenant des données de location de vélos à NY-EU

```
{// start sample document

  "_id": {
    "$oid": "572bb8222b288919b68abf82"
  },

  "bikeid": 14785,

  "end station id": 433,
  "end station name": "E 13 St & Avenue A",
  "start station id": 518,
  "start station name": "E 39 St & 2 Ave",
  "tripduration": 812,
  "usertype": "Subscriber"
}// end sample document
```

Afficher le nombre de locataires ayant pris et rendu leurs vélos à la même station

```
> db.trips.find({ "$expr": { "$eq": [ "$end station id", "$start station id" ] } }).count()
```

```
> db.trips.find({ "$expr": { "$eq": [ "$end station id", "$start station id" ] } }).count()
```



- ✓ Utilisation d'opérateurs
- ✓ Pointe la valeur d'un champ (clé)

433

518

NB : Quand on utilise l'opérateur \$expr, on doit utiliser la syntaxe d'agrégation pour l'opérateur de comparaison:

Syntaxe MQL : { <champ> : { <opérateur> : valeur } }

Syntaxe agrégation : { <opérateur> : { <champ>, <valeur> } }

limit(<nb>) : Limiter le nombre de documents affichés à <nbr>.

skip(<nb>) : Ne pas afficher (sauter) les <nbr> premiers document du curseur.

sort() : trier les résultats par ordre croissant (1) ou décroissant (-1)

Exemple 1: Afficher les 5 profils Instagram les plus populaires

```
db.instaAccounts.find().sort({"followers": -1}).limit(5)
```

Exemple 2: Afficher les 5 codes postaux (zip code) les plus peuplés des Etats Unis.
Trier par ordre croissant des noms villes

```
db.zips.find({}, {"zip": 1, "city":1}).sort({"pop": -1}, {"city" : 1}).limit(5)
```

```
{ "_id" : ObjectId("5c8eccc1caa187d17ca7044d"), "city" : "CHICAGO", "zip" : "60623" }  
{ "_id" : ObjectId("5c8eccc1caa187d17ca7307f"), "city" : "BROOKLYN", "zip" : "11226" }  
{ "_id" : ObjectId("5c8eccc1caa187d17ca72fa0"), "city" : "NEW YORK", "zip" : "10021" }  
{ "_id" : ObjectId("5c8eccc1caa187d17ca72fa5"), "city" : "NEW YORK", "zip" : "10025" }  
{ "_id" : ObjectId("5c8eccc1caa187d17ca6f39d"), "city" : "BELL GARDENS", "zip" : "90201" }
```

- *Limit()* sans *sort()* : fournir un résultat random sans aucune garantie sur l'ordre
- *Limit()* avant *sort()*: on risque de louper des données qui devaient être incluse suite au tri

➡ *Quel que soit l'ordre d'apparition des méthodes, MongoDB suppose toujours qu'on tri avant de limiter le nombre de documents affichés*



L'ordre pour `skip()` et `Limit()` : Si les données du curseur n'ont pas été triés avec un `sort()`, l'ordre par défaut utilisé par MongoDB est l'ordre d'insertion.

Les méthodes des curseurs :

- ✓ **`pretty()`**
- ✓ **`count()`**
- ✓ **`sort()`**
- ✓ **`limit()`**
- ✓ **`Skip()`**

- **Définition** : Une structure de données permettant d'optimiser les requêtes en termes de temps d'exécution
- **Use cases** :
 - ✓ Si on interroge une collection fréquemment en fonction d'un champ X, il est recommandé de créer un index sur le champ X.
 - ✓ Trier les données peut être couteux en termes de charge mémoire et temps d'exécution, surtout pour les BD volumineuses.

Exemple

```
> db.trips.find({ "birth year": 1989 })  
> db.trips.find({ "start station id": 476 }).sort( { "birth year": 1 } )
```

➡ Les deux requêtes peuvent bénéficier d'un index

```
> db.trips.createIndex({ "birth year": 1 })
```

- Créer un index sur l'année de naissance dans un **ordre croissant (1)**.

- Il s'agit d'une **combinaison** entre **update()** et **insert()**
- Utilisée comme étant une option de la méthode de mise à jour (update) d'un ou plusieurs documents
- Upsert est désactivée par défaut (initialisée à false)

```
> db.collection.update({<localication Doc>}, {<MAJ>}, {"upsert" : true})
```

- ➡ Si un document **matche le critère de filtrage**: upsert() effectue une **MAJ**
- ➡ Si **aucun document ne correspond au critère de filtrage** : upsert() effectue une **insertion d'un nouveau document**