

Module Big Data & Cloud Computing

Catarina FERREIRA DA SILVA

et

Mahmoud BARHAMGI

Université Claude Bernard Lyon 1

<http://tinyurl.com/IUT-BD2017>

Les objectifs de ce module

- ❑ Introduire les concepts de:
 - Big Data (son origine, ses applications et enjeux, etc.);
 - Cloud Computing (concept, applications, enjeux , etc.);
- ❑ Etudier les bases de données NoSQL utilisées dans le contexte de Big Data
- ❑ Maîtriser un SGBD NoSQL par la pratique (MongoDB)
- ❑ Introduire le Framework MapReduce

Organisation du cours

Première partie: Big Data et BDs NoSQL (16h)

- ☐ Présentation des généralités
- ☐ Apprendre MongoDB et MapReduce par la pratique
 - Travail pratique (TP)

Deuxième partie: Cloud Computing (10h)

- ☐ Lecture autonome des articles scientifiques et préparation d'une présentation
- ☐ Exposés devant le groupe

Évaluation

- ❑ 25% TP Apprendre MongoDB et MapReduce par la pratique;
- ❑ 25% DS sur machine (TP individuel);
- ❑ 25% exposé (étude des articles et exposé en groupe);
- ❑ 25% questionnaire QROC

Introduction

- Depuis les années 1970, dominance du modèle Relationnel
- Avec l'émergence des réseaux sociaux, se pose le problème du passage à l'échelle (millions d'utilisateurs interagissant avec un système donné, ex., *Facebook, Twitter*, etc.)
- Réflexions pour passer d'un système large échelle vertical à un système large échelle horizontal (ajout de machines)
- Explosion du volume de données à stocker et à traiter (apparition du phénomène "Big Data")

Introduction

Le "***Big Data***" c'est la science qui étudie la modélisation, le stockage et le traitement (analyse) d'un ensemble de données très volumineuses, croissantes et hétérogènes, dont l'exploitation permet entre autres

- L'aide à la prise de décisions
- Découverte de nouvelles connaissances
- Amélioration de la vie des gens
-

Causes

- Faible coût du stockage
- Faible coût des processeurs
- Mise à disposition des données

Introduction (continuation)

Quelques applications du Big Data

- L'analyse d'opinions politiques (e.g., la prédiction du résultat des élections États-Uniennes 2012)
- La lutte contre la criminalité et la fraude
- L'amélioration des méthodes de marketing publicitaire et de vente
- Décodage du génome humain
- La prédiction des épidémies (e.g., la prédiction de l'épidémie de grippe aviaire en 2009 par les analystes de Google quelques semaines avant apparition)
- La découverte des effets secondaires des médicaments
-

Introduction (continuation)

- ❑ L'analyse des Big Data demande des SGBD distribués (pour s'approcher des sources des données)
 - à forte disponibilité
 - résistants au morcellement

- ❑ De plus, les applications Web (qui génèrent les données)
 - possèdent des schémas dynamiques (nombre d'attributs extensible)
 - doivent gérer un nombre important de lectures/écritures

- *Les SGBD Relationnels sont moins adaptés pour le Big Data, le Web et les réseaux sociaux, ce qui a conduit au mouvement NoSQL (ou Not only SQL)*

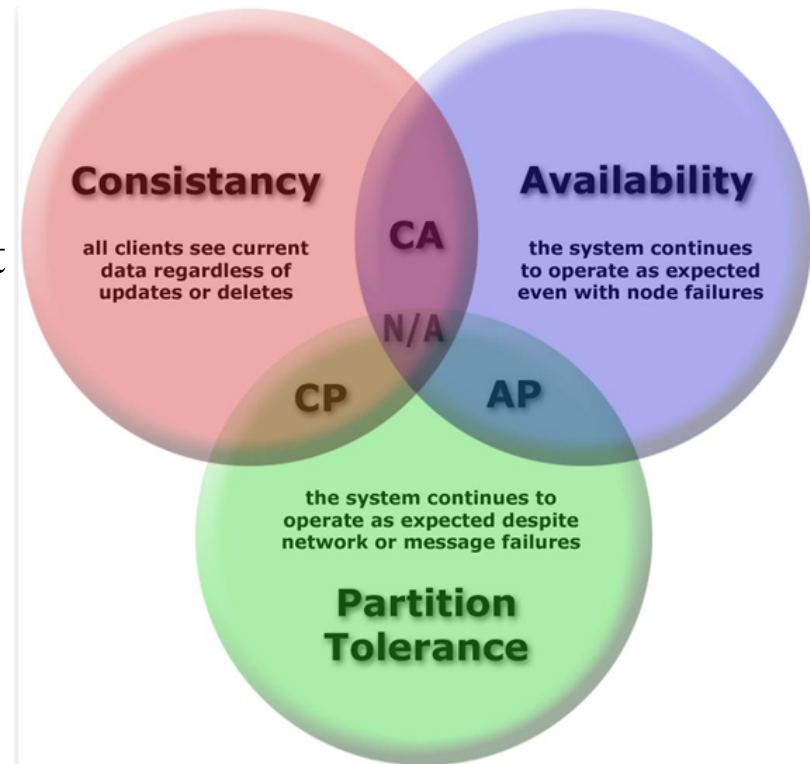
Plan

- Généralités sur les SGBD NoSQL
- MongoDB

Caractéristiques des SGBD NoRel

Caractéristiques

- Garantie de deux propriétés parmi *cohérence, disponibilité et résistance au morcellement* (souvent la *disponibilité* et la *résistance au morcellement* au détriment de la cohérence, mais pas toujours)
- Performance (scalabilité horizontale)
- Pas de schéma de table fixé
- Éviter les jointures (données dénormalisées)
- Pas de support de transactions complexes
- Pas de fonctionnalité de requêtage complexe (SQL)



Les propriétés CAP (*Consistency, Availability, Partition tolerance*)

Selon le théorème de Brewer (ou théorème CAP), un système distribué ne peut pas garantir en même temps les trois propriétés suivantes

- Cohérence (*Consistency*) : tous les nœuds du système voient la même information au même moment
- Disponibilité (*Availability*) : toute requête reçoit une réponse
- Résistance au morcellement (*Partition tolerance*) : fonctionnement autonome en cas de morcellement du réseau (sauf panne totale)

Caractéristiques des SGBD Non Relationnels

Les SGBD NoSQL sont dites **BASE**, pour

- **Basically Available** (haute disponibilité)
- **Soft-state** : l'état du système continue d'évoluer, même sans mise à jour ou nouvelles entrées, pour atteindre un état final où tous les nœuds voient les mêmes données (la propriété *cohérence*)
- **"Eventually consistent"** (cohérence sur le long terme) : à un moment donné, les nœuds ne voient pas nécessairement les mêmes informations, mais à long terme, le système se stabilise et tous les nœuds voient exactement les mêmes données (la propriété de *cohérence* est relaxée pour atteindre une meilleure *disponibilité*)

Catégories des bases de données NoSQL

Classement selon le modèle de données

- BDs orientées documents
- BDs orientées colonnes
- BDs orientées graphes
- BDs orientées clé-valeur
- ...

D'autres classements sont possibles

- Selon les propriétés CAP
- Selon les propriétés fonctionnelles
- ..

Rappel du modèle relationnel

Un modèle que vous connaissez bien...

- **Concepts:** *Relations, Attributs, Tuples*, etc.
- Propriétés de cohérence et de disponibilité
- **SGBDs** : Oracle, PostgreSQL, MySQL, etc.

Exemple de tables relationnelles

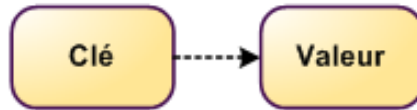
Table ECRIVAIN

ID	Nom	Pays	DateNaiss
2	John Smith	USA	20-09-1948

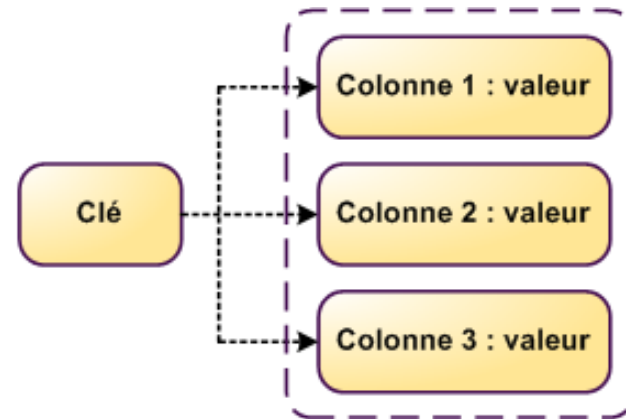
Table LIVRE

ID	Titre	Prix	DatePubli	Ecrivain
1	True Justice	30	01-01-2000	2

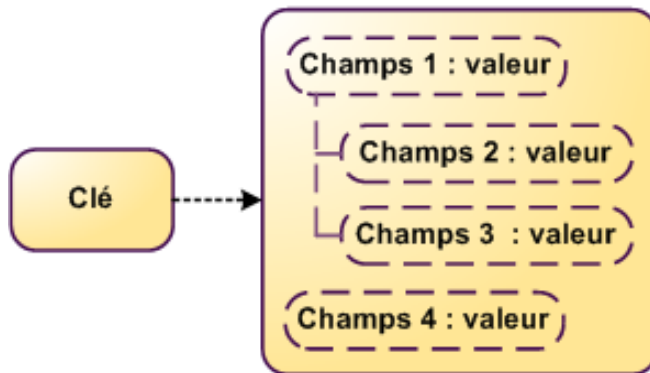
Catégories des bases de données NoSQL



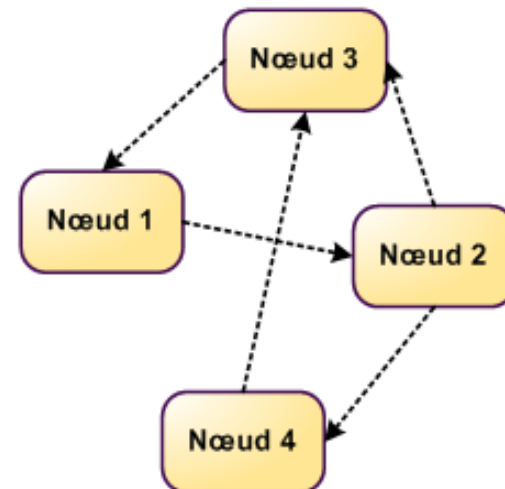
BDD Clé-Valeur



BDD Orientée colonnes



BDD Orientée document



BDD Orientée graphe

Entrepôt clé-valeur

- ❑ Ce modèle est aussi appelé "*key-value Store*" ou tableau associatif
 - La clé est un identifiant unique
 - La valeur peut être structurée ou pas
- ❑ L'implémentation minimale de ce modèle doit fournir les fonctions
 - valeur = get(clé)
 - insert(clé, valeur)
 - delete(clé)
- ❑ SGBD type entrepôt clé-valeur
 - Serveur standalone (Redis)
 - Distribué (Dynamo, Riak, Voldemort)
 -

Entrepôt clé-valeur

- ❑ **Avantages et inconvénients:** *Performances (++)*, *passage à l'échelle (++)*, *flexibilité (++)*, *prise en main (++)*, *nombre de fonctionnalités (~/-)*
- ❑ Exemple d'un entrepôt clé-valeur

```
"nom-ecrivain" : " John Smith"  
"country" : " USA"  
"birthdate" : " 20-09-1948"  
"book" : " TrueJustic"  
"code" : " 30"  
"publicationDate" : " 01-01-2000"
```

BD orientée colonnes

BD orientée colonnes : organisation des données en colonnes

- Une *colonne* stocke le nom de la colonne et la valeur associée (et un timestamp)
- Une *supercolonne* stocke des colonnes
- Une famille de colonnes stocke des colonnes ou supercolonnes
- La sérialisation des données se fait par colonne (ou par supercolonnes) et non pas par tuples de données comme dans le modèle relationnel

Avantages

- Schéma dynamique (ajout de colonne)
- Pas de stockage de valeurs nulles

BD orientée colonnes

❑ Exemple d'une BD orientée colonnes

La famille de colonne *écrivain*

1	nom: John Smith	pays: USA	DateNaiss: 20-09-1948
---	--------------------	--------------	--------------------------

La famille de colonne *livre*

2	titre: True Justice	prix: 30	DatePubli: 01-01-2000	auteur: 1
---	------------------------	-------------	--------------------------	--------------

❑ SGBD orienté colonnes

- BigTable (propriété de Google)
- Cassandra (implémentation libre de BigTable)
- HBase, Hypertable (basé sur BigTable et Hadoop DFS)

BD orientée colonnes

❑ **Avantages et inconvénients:** Performances (++), passage à l'échelle (++), flexibilité(+), prise en main (+), nombre de fonctionnalités (-)

❑ **Références**

<http://cassandra.apache.org/>

<http://cassandra-php.blogspot.fr/>

<http://hbase.apache.org/>

<http://hypertable.com/>

<http://en.wikipedia.org/wiki/Hadoop>

BD orientée graphes

❑ **BD orientée graphes** un ensemble d'éléments interconnectés avec un nombre indéterminé de relations entre eux

- Noeuds pour représenter les éléments
- Arc étiqueté et directionnel entre deux noeuds
- Propriété sur un noeud ou un arc

❑ **Avantages**

- Facilité d'évolution du schéma
- Bonnes performances pour des requêtes type graphe (e.g., plus court chemin)
- Permet de représenter les 3 autres modèles
- Adéquat pour la représentation des réseaux sociaux

BD orientée graphes

❑ SGBD orienté graphes

- Neo4J (ACID, avec transactions)
- Allegro Graph (triplestore, raisonnement Prolog)
- Virtuoso (triplestore et SGBDR)
-

❑ **Avantages et inconvénients:** Performances (~), passage à l'échelle (~), flexibilité(++), **prise en main (-)**, fonctionnalités (~)

❑ Exemple d'une BD orientée graphes



BD orientée documents

❑ **BD orientée documents** est une collection de documents (clé-document)

- Chaque document a des champs et une clé
- Deux instances de document peuvent avoir des champs différents
- Organisation des documents selon des tags, métadonnées, collections

❑ **Avantages**

- Recherche de documents basée sur leur contenu
- Facilité d'évolution du schéma

❑ SGBD orienté documents

- CouchDB (stockage JSON, requêtes en Javascript via Map Reduce)
- MongoDB (documents "à la JSON", requêtes en Javascript)
- Couchbase (documents JSON)
-

❑ Avantages et inconvénients: *Performances* (++), *passage à l'échelle* (~/++), *flexibilité* (++), *prise en main* (+), *nombre de fonctionnalités* (~/-)

BD orientée documents

❑ Exemple d'une BD orientée documents

Par référence

Document 1

```
{
  id: "livre1",
  titre: "True Justice",
  prix: 30,
  ecrivain: "ecrivain2",
  datePubli: "01-01-2000"
}
```

Document 2

```
{
  id: « ecrivain2",
  nom: "John Smith",
  pays: "USA",
  dateNaiss: "20-09-1948",
}
```

ou

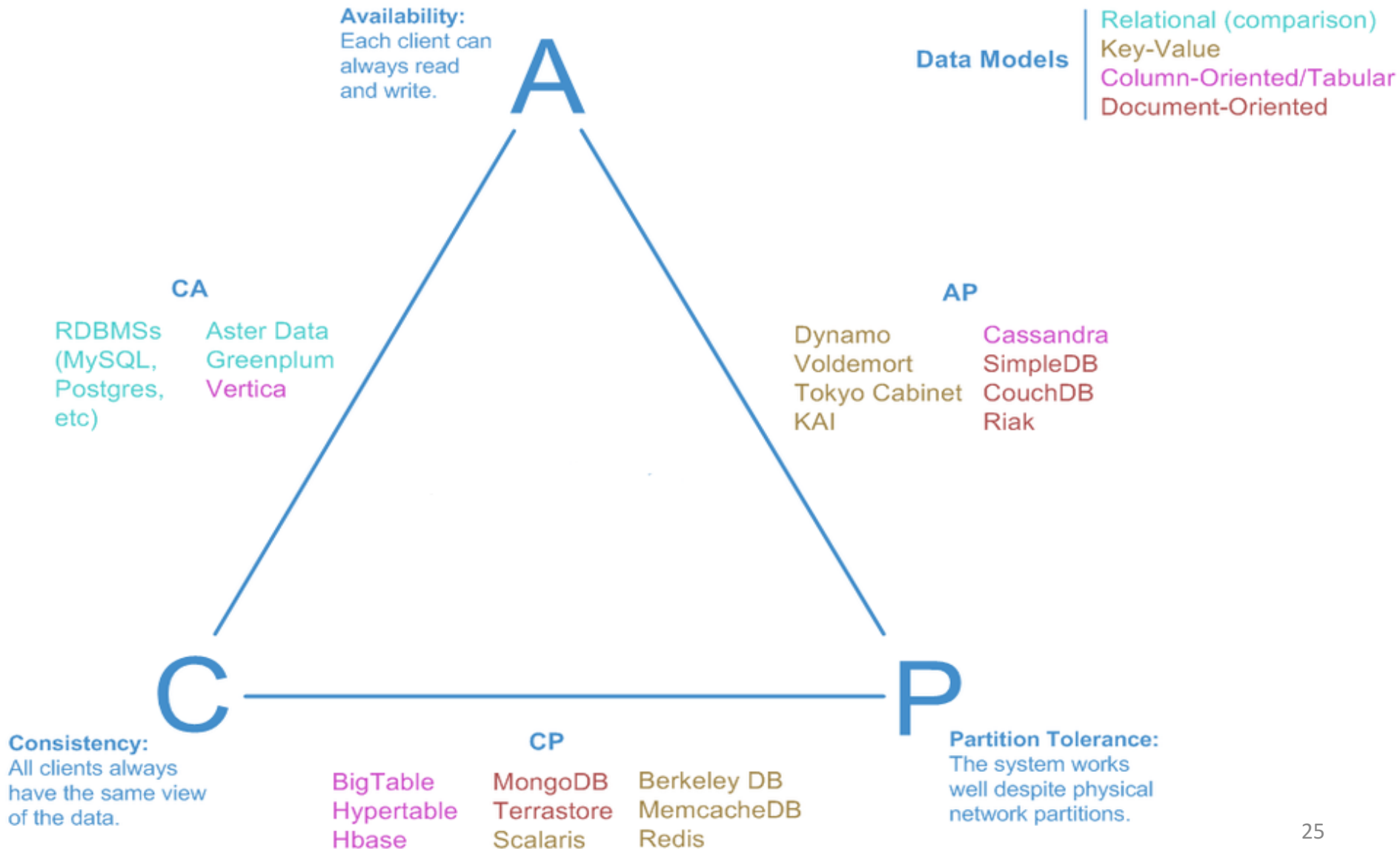
Par inclusion

Document 3

```
{
  id: « ecrivain2",
  nom: "John Smith",
  pays: "USA",
  dateNaiss: "20-09-1948",
  livres: [
    {
      titre: "True Justice",
      prix: 30,
      datePubli: "01-01-2000"
    }
  ]
}
```


Classement des SGBDs NoSQL selon les propriétés CAP

Disponibilité (A), Cohérence (C), Résistance au morcellement (P)



Lecture

- *Les origines & les grands principes du big data*, Nicolas Minelle,
<http://datascience.bluestone.fr/blog/les-origines-les-grands-principes-du-big-data>
- *Big-Data Applications in the Government Sector*, ACM Communications
- *SQL Databases versus NoSQL Databases*, Michael Stonebraker
- *Will NoSQL Databases Live Up to Their Promise*, Neal Leavitt
- *In Search of Database Consistency*, Michael Stonebraker
- <http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql/>
- *What's All the Buzz Around "Big Data?"*

Plan

- Généralités sur les SGBD NoSQL
- MongoDB

MongoDB

Caractéristiques de MongoDB

- Orienté documents
- Open-source
- Populaire (*5^{ème} SGBD le plus utilisé*)
- Passage à l'échelle horizontale et réplication
- Système CP (cohérent et résistant au morcellement)
- Journalisation
- Utilisation de Map Reduce



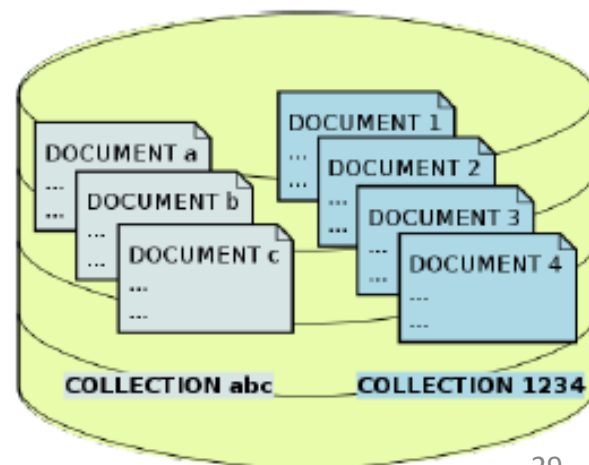
Concepts principaux

Une Base de données MongoDB est

- Un ensemble de collections
- Un espace de stockage

Collection (\approx "*Table*" en modèle Relationnel)

- Ensemble de documents qui partagent un objectif ou des Similarités
- Pas de "schéma" prédéfini



Concepts principaux

- ❑ **Document** (*≈ ligne ou tuple en modèle Relationnel*)
 - Un enregistrement dans une collection
 - Syntaxe et stockage au format BSON (Binary JavaScript Object Notation)
 - Identifiant d'un document (clé "**_id**")
- ❑ **Format BSON avec améliorations**
 - Ensemble de champs ou paires champ/valeur
 - Une valeur peut être un objet complexe (liste, document, ensemble de valeurs, etc.)
 - Représentation de nouveaux types (e.g., dates)

Concepts principaux

Syntaxe d'un document en MongoDB

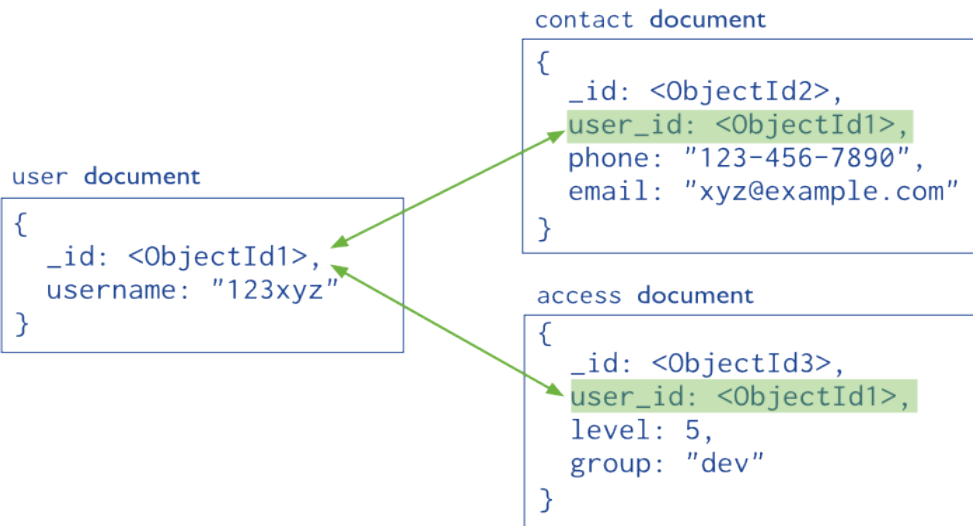
- ❑ **_id** est un identifiant (généré ou manuel)
- ❑ **att-1** est un attribut dont la valeur est une chaîne de caractères
- ❑ **att-2** est un attribut dont la valeur est un entier
- ❑ **att-3** est un attribut dont la valeur est une liste de valeurs
- ❑ **att-n** est un attribut dont la valeur est un document inclus

```
{  
  _id: <un identifiant>,  
  "att-1": "val-1",  
  "att-2": val-2,  
  "att-3": ["val-31", "val-32", ...]  
  ...  
  
  "att-n": { "val-n1": "val-n1",  
             ....  
           }  
}
```

Relations entre documents

Relation entre les documents de différentes collections

- ❑ **Par référence** : l'identifiant d'un document (son "_id") est utilisé comme valeur attributaire dans un autre document
 - se rapproche du modèle de données normalisées
 - nécessite des requêtes supplémentaires côté applicatif



- ❑ **Par inclusion ("embedded")** : un "sous-document" est utilisé comme valeur
 - philosophie "Non-Relationnel" (pas de jointures)
 - meilleures performances



Les opérations sur les documents

- ❑ Les opérations CRUD deviennent IFUR
 - INSERT
 - FIND
 - UPDATE
 - REMOVE
- ❑ Toute opération sur un seul document est atomique
- ❑ Lors d'insertion et mises à jour, la base de données et la collection sont automatiquement créées si elles n'existent pas

Références

<http://docs.mongodb.org/manual/core/crud-introduction/>

Opération INSERT

Syntaxe pour insérer un document dans une collection *coll*

```
db.collectionName.insert(  
    <doc>,  
    <options>  
)
```

- ❑ *<doc>*, un document ou un tableau de documents à insérer
- ❑ *<options>*, un document pouvant contenir
 - un booléen *ordered* (insertion de plusieurs documents stoppée en cas d'erreur)
 - un document *writeConcern* (type de garantie d'écriture)

Opération INSERT

- ❑ Concernant l'identifiant du document à insérer
 - Si la clé "_id" est présente dans le document, s'assurer de l'unicité de sa valeur
 - Sinon, MongoDB ajoute un identifiant automatiquement (type *ObjectID* sur 12 octets)
- ❑ Le document optionnel *writeConcern* décrit la garantie du succès d'une opération avec les attributs
 - *W*: nombre de confirmations d'écriture (sur le nœud principal et ses répliquas) (e.g., 0, 1, n, "majority")
 - *J*: un booléen pour écrire dans le journal de la base de données
 - *wtimeout*: une limite de temps en ms (que l'opération ne doit pas dépasser)
- *Référence*: <http://docs.mongodb.org/manual/reference/write-concern/>

Opération INSERT

Collection
↓
`db.users.insert(`

Document
↓

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

)

Document

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

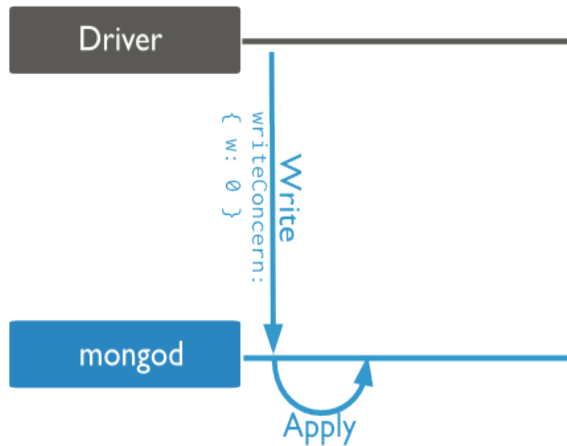
insert

Collection

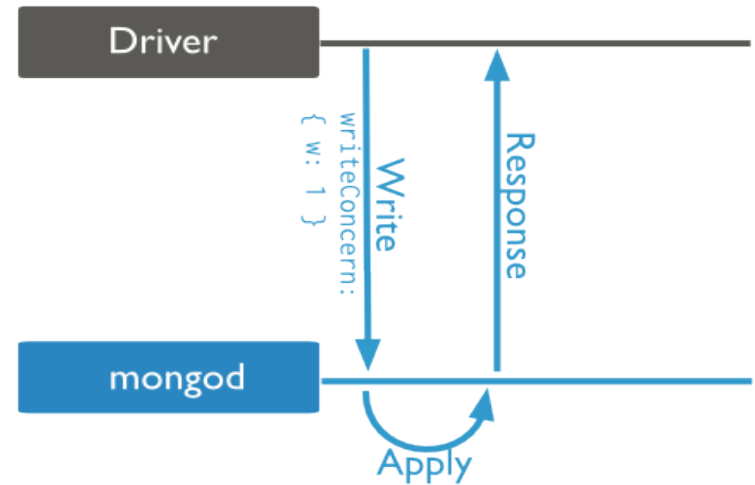
{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }

users

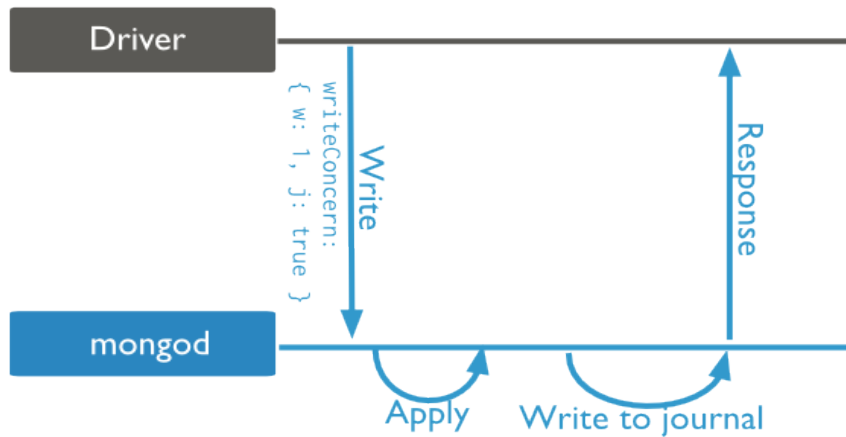
Opération INSERT



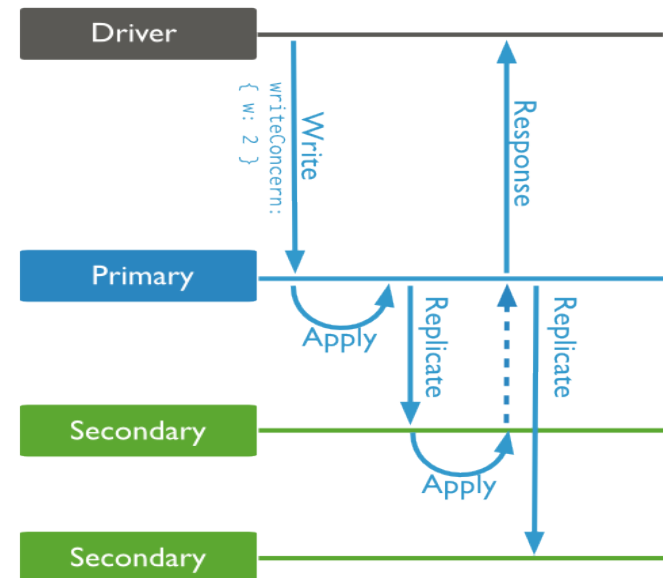
Ecriture sans garantie du succès: $w : 0$



Ecriture avec garantie du succès sur le nœud principal: $w : 1$



$w : 1, j : true$



$w : 2$

Opération FIND

- ❑ Syntaxe pour rechercher des documents dans une collection *coll*

```
db.collectionName.find(  
    <critères>,  
    <projection>  
)
```

- *<critères>*, optionnel, un document contenant les critères de sélection des documents pertinents
 - *<projection>*, optionnel, un document contenant les champs qui seront présents dans les documents résultats
- ❑ Cette opération retourne un ensemble de documents (plus exactement un curseur vers ces documents)
 - ❑ Références <http://docs.mongodb.org/manual/reference/method/db.collection.find/>

Opération FIND

Le document *<critères>* permet de

- Retourner tous les documents d'une collection

```
db.collectionName.find( {} )
```

- Retourner des documents dont la valeur d'une clé est égale à une constante

```
db.collectionName.find( {clé1 : <const>} )
```

Exemples

```
db.users.find( {nom : "Dupont"} )  
db.users.find( {age: 30} )
```

- Retourner des documents en utilisant des opérateurs

```
db.collectionName.find( {clé1 : <opérateur>, ... , clék : <opérateur>} )
```

Opération FIND

Différents types d'opérateurs

- Comparaison de valeur (*\$ne*, *\$gt*, *\$gte*, *\$lt*, *\$lte*)

Clients âgés plus de 30 ans

```
db.clients.find({age : {$gt : 10}})
```

Produits avec une quantité inférieure ou égale à 5

```
db.products.find({quantité : {$lte : 5}})
```

Produits avec une quantité différente de 50

```
db.products.find({quantité : {$ne : 50}})
```

- Comparaison avec un tableau de valeurs (*\$in*, *\$nin*)

Clients avec un job parmi Prof, Taxi driver, Doctor

```
db.clients.find({job : {$in : ["Prof", "Taxi driver", "Doctor"]}})
```


Opération FIND

- Opérateurs de comparaison logique (*\$and*, *\$or*, *\$not*, *\$nor*)

Documents avec un prix à 5 et une quantité supérieure à 1

```
db.coll.find( { $and : [{prix : 5}, {quantité : {$gt : 1}} ] } )
```

Documents avec un prix qui ne soit pas supérieur à 5

```
db.coll.find({{prix : {$not : {$gt : 5}}})
```

- Opérateurs d'existence ou type d'un élément (*\$exists*, *\$type*)

Documents avec un champ "prix"

```
db.coll.find({$prix : {$exists : true}})
```

Opération FIND

- Opérateurs d'évaluation (*\$mod*, *\$regex*, *\$text*, *\$where*)

Documents dont l'un des attributs contient abc ou cde

```
db.coll.find({$text : {$search : "abc cde"}})
```

Documents avec un champ nom contenant abc

```
db.coll.find({nom : {$regex : '*abc*'}})
```

```
db.coll.find({nom : {$regex : /*abc*/}})
```

Idem mais avec option insensible à la casse (i)

```
db.coll.find({nom : {$regex : '*abc*', $options : 'i' }})
```

```
db.coll.find({nom : {$regex : /*abc*/i }})
```

- De tableaux (*\$all*, *\$elemMatch*, *\$size*), spatiaux (*\$near*, etc.)
 - Voir <https://docs.mongodb.com/manual/reference/operator/query-array/>

Opération FIND

Le document *<projection>*

- Contient soit une liste de champs projetés, soit une liste de champs exclus
- Syntaxe : { att-1 : <boolean>, . . . , att-k : <boolean> }
- La valeur du booléen détermine la projection de l'attribut
 - 1/true : *attribut projeté (dans les documents résultats)*
 - 0/false : *attribut non projeté (exclu)*
- Champ "*_id*" *inclus par défaut, mais possibilité de l'exclure (y compris dans un document qui liste des champs projetés)*

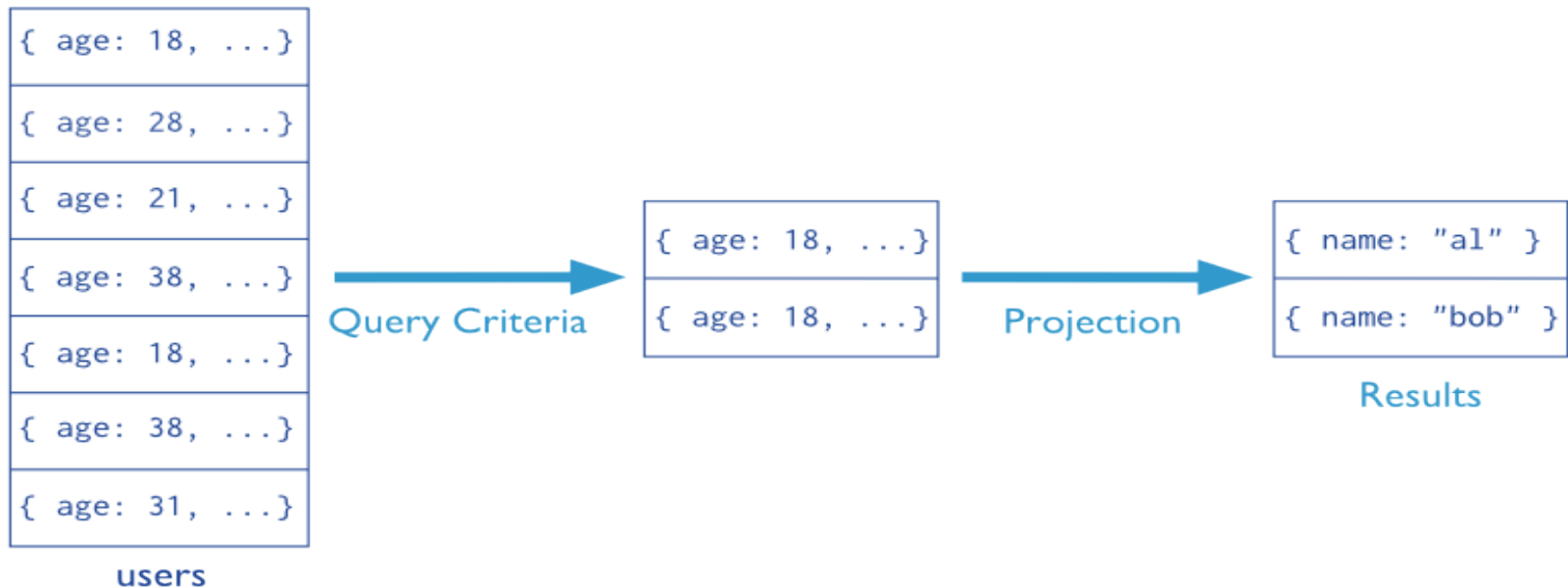
Tous les champs sauf *prix et titre* : {*prix* : false, *titre* : false}

Uniquement le champ *prix* : {*prix* : 1, *_id* : 0}

Opération FIND

- ❑ *Exemple de requête avec sélection et projection : la requête retourne les noms des usagers âgés de 18, sans leur id*

Collection Query Criteria Projection
`db.users.find({ age: 18 }, { name: 1, _id: 0 })`



Opération FIND

La méthode FIND retourne un curseur vers une liste de documents, sur lequel il est possible d'appliquer des méthodes *modifiers*

1. SORT($\{ \text{att-1} : 1/-1, \dots, \text{att-k} : 1|-1 \}$) : *les documents sont triés dans l'ordre naturel (1 pour ascendant, pour -1 descendant), selon le premier champ, puis le second en cas d'égalité, et ce jusqu'au champ k*
2. SKIP(N) : *les n premiers documents sont supprimés du résultat*
3. LIMIT(N) : *n documents sont retournés au maximum*

Tri sur le prix puis titre : `db.coll.find().sort({prix : -1, titre : 1})`

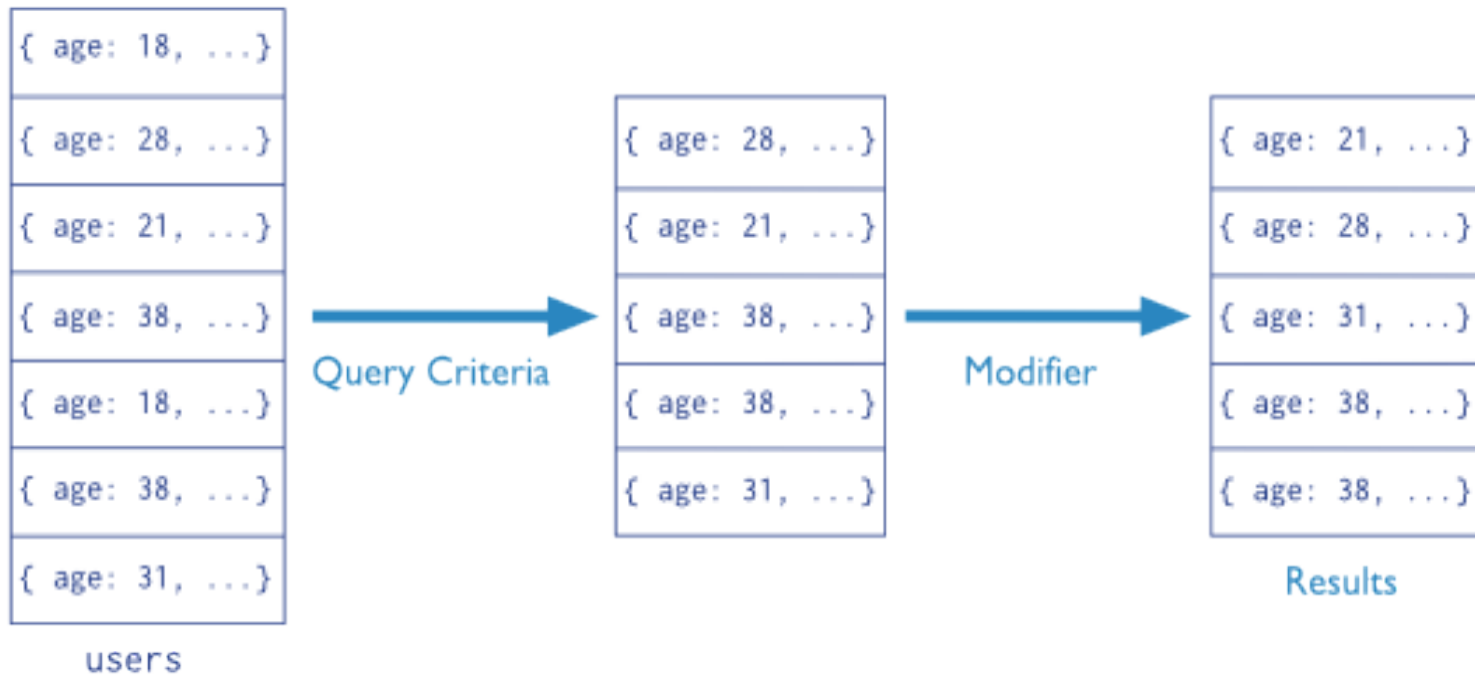
Limite de 10 documents : `db.coll.find().limit(10)`

Combinaison : `db.coll.find().sort({prix : -1}).skip(10).limit(50)`

Opération FIND

Exemple

Collection Query Criteria Modifier
`db.users.find({ age: { $gt: 18 } }).sort({age: 1 })`



Opération FIND

Exemple

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

Opération UPDATE

Syntaxe pour mettre à jour un document dans une collection *coll*

```
db.coll.update(  
    <query>,  
    <màj>,  
    <options>  
)
```

- *<query>*: une requête pour sélectionner les documents à mettre à jour
- *<màj>*, un document avec les mises à jour
- *<options>*, un document pouvant contenir
 - un booléen *upsert* (création d'un document si aucun résultat pour query)
 - un booléen *multi* (pour mettre à jour plusieurs documents)
 - un document *writeConcern* (type de garantie d'écriture)

Opération UPDATE

Contenu du document *<màj>*

❑ Uniquement des opérateurs de mise à jour

➤ le(s) document(s) retourné(s) par *<query>* ont leurs champs mis à jour

➤ opérateurs sur champs (e.g., *\$inc*, *\$rename*, *\$set*)

➤ opérateurs sur tableaux (e.g., *\$pop*, *\$push*, *\$addToSet*)

❑ Uniquement des paires de clé/valeur

➤ le document *<màj>* remplace l'intégralité du premier document répondant aux critères de *<query>*

➤ la valeur de "*_id*" est conservée


❖ Référence <http://docs.mongodb.org/manual/reference/operator/update/>

Opération UPDATE

Exemple

Mise à jour du statut (nouvelle valeur "A") pour tous les documents (option "multi") contenant un champ âge supérieur à 18

```
db.users.update(  
  { age: { $gt: 18 } },  
  { $set: { status: "A" } },  
  { multi: true }  
)
```



- ← collection
- ← update criteria
- ← update action
- ← update option

Opération REMOVE

- ❑ Syntaxe pour supprimer une base de données *db*

```
db.dropDatabase()
```

- ❑ Syntaxe pour supprimer une collection *coll*

```
db.coll.remove()
```

Opération REMOVE

- ❑ Syntaxe pour supprimer des documents de la collection *coll*

```
db.coll.remove(  
    <query>,  
    <options>  
)
```

- *<query>* : un document utilisant des opérateurs de requête
- *<options>* : un document pouvant contenir
 - un booléen *justOne* (pour supprimer un seul document)
 - un document *writeConcern* (type de garantie d'écriture)

Opération REMOVE

Exemple

Exemple de suppression de tous les produits de la collection "products" dont la quantité est supérieure à 20

```
db.products.remove( { qty: { $gt: 20 } } )
```