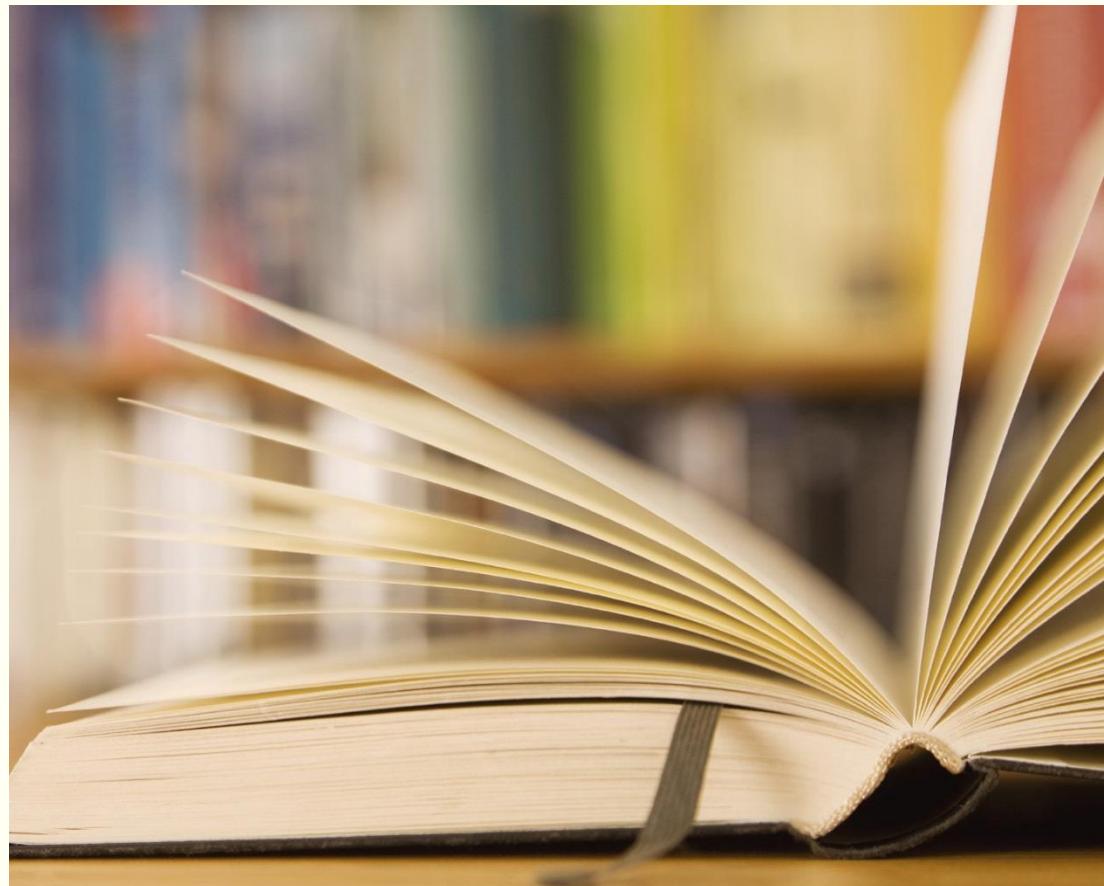




INITIATION NOSQL

Illustration avec MongoDB
Durée : 3 jours



Modules

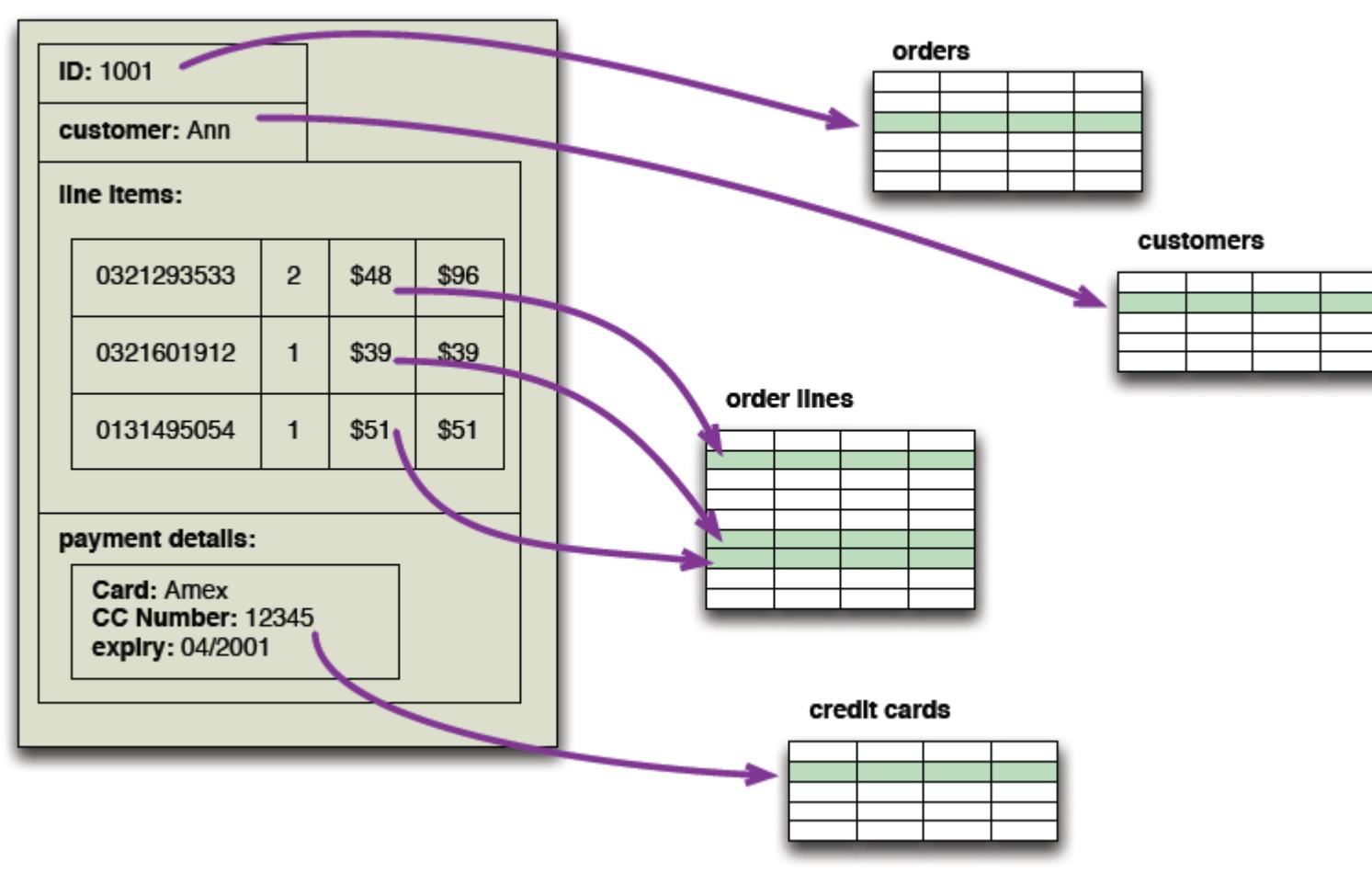
- **Module 1 : Introduction au NoSQL**
- **Module 2 : Opération CRUD**
- **Module 3 : Les requêtes**
- **Module 4 : Design et Data Modèle**
- **Module 5 : Performance**
- **Module 6 : Agrégation Framework**
- **Module 7 : Administration**
- **Module 8 : Java et MongoDB**



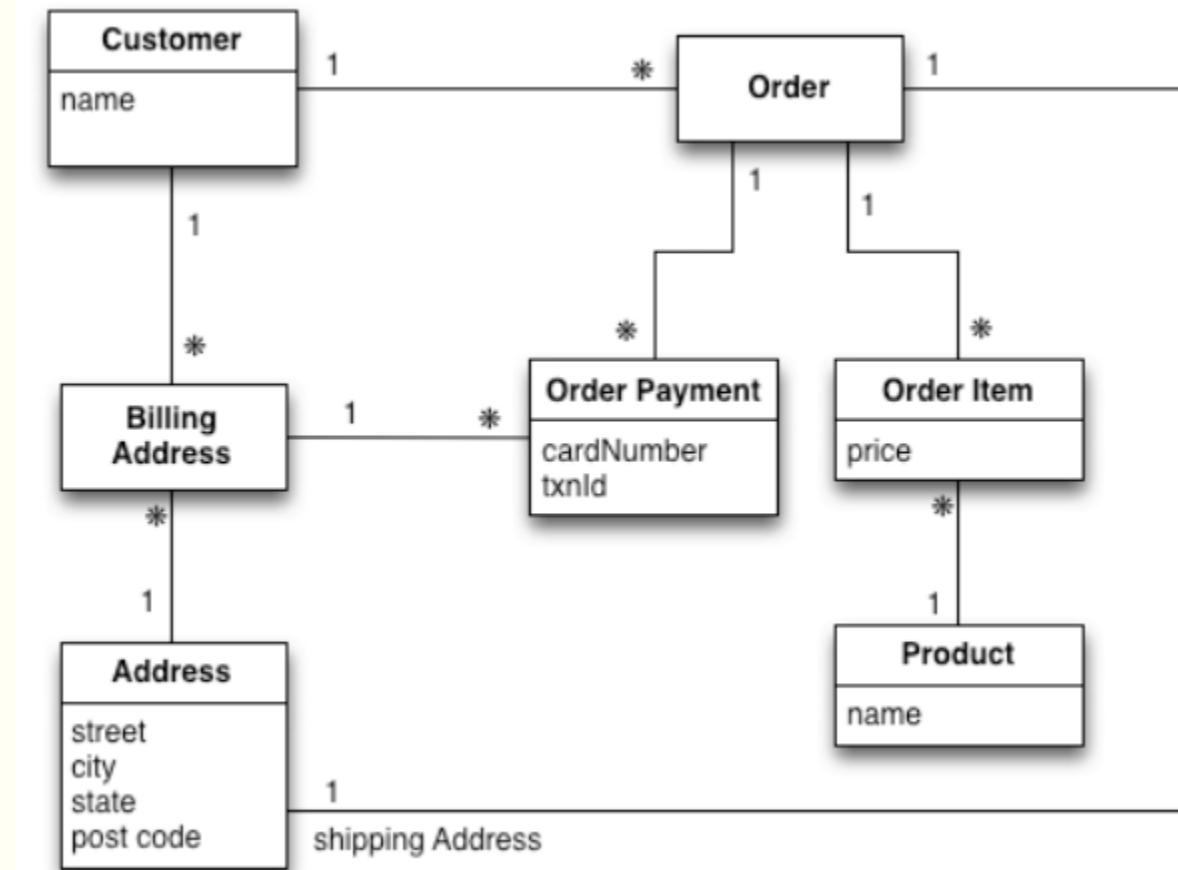
MODULE 1

Introduction au NoSQL

Impedance mismatch



Domain driven data models



RDBMS data

Customer	
Id	Name
1	Martin

Orders		
Id	CustomerId	ShippingAddressId
99	1	77

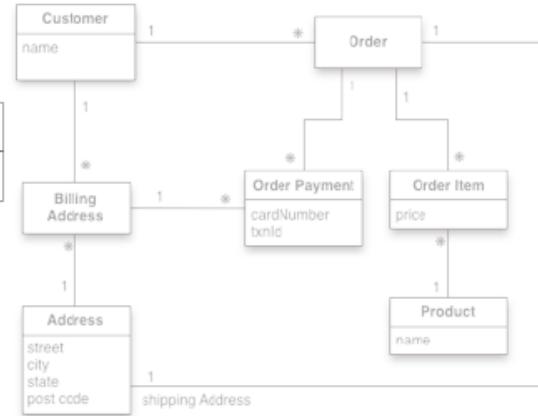
Product	
Id	Name
27	NoSQL Distilled

BillingAddress		
Id	CustomerId	AddressId
55	1	77

OrderItem			
Id	OrderId	ProductId	Price
100	99	27	32.45

Address	
Id	City
77	Chicago

OrderPayment				
Id	OrderId	CardNumber	BillingAddressId	txnid
33	99	1000-1000	55	abelif879rft

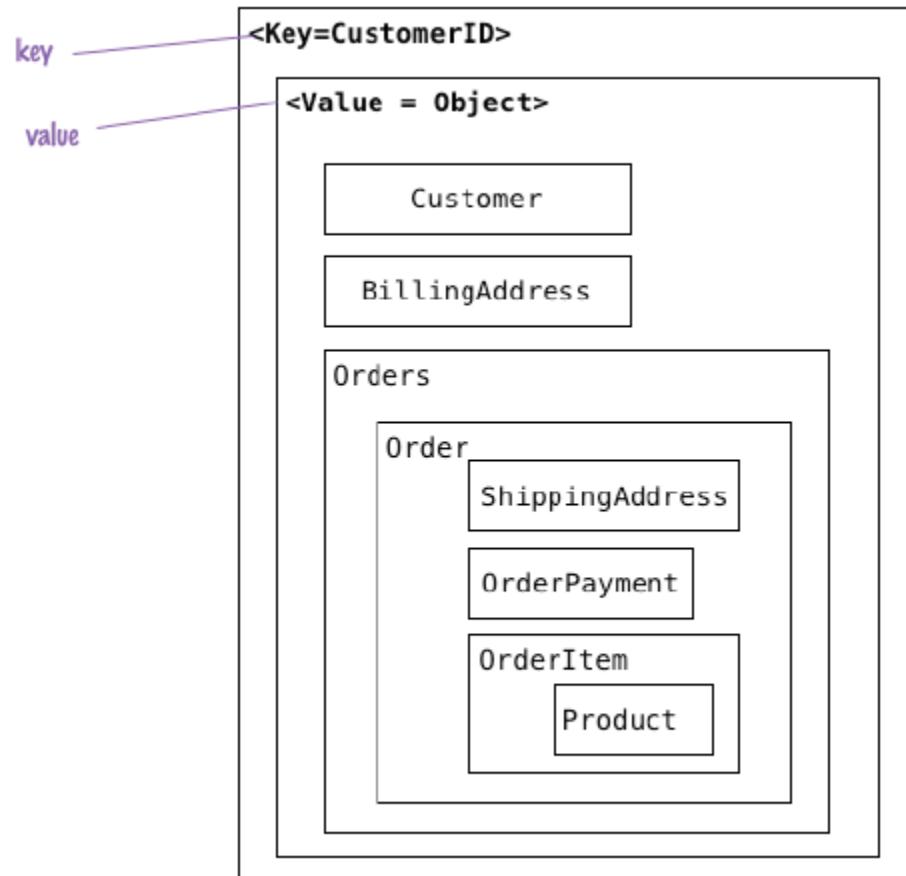


Le monde Relationnel

Caractéristiques:

- Données structurées (Table/Schéma)
- Données Normalisées (Formes Normales)
- Standard (SQL)
- Transactionnel (ACID)
- Requête Complexe (Jointure)
- Des contraintes (Intégrité des données)
- ...

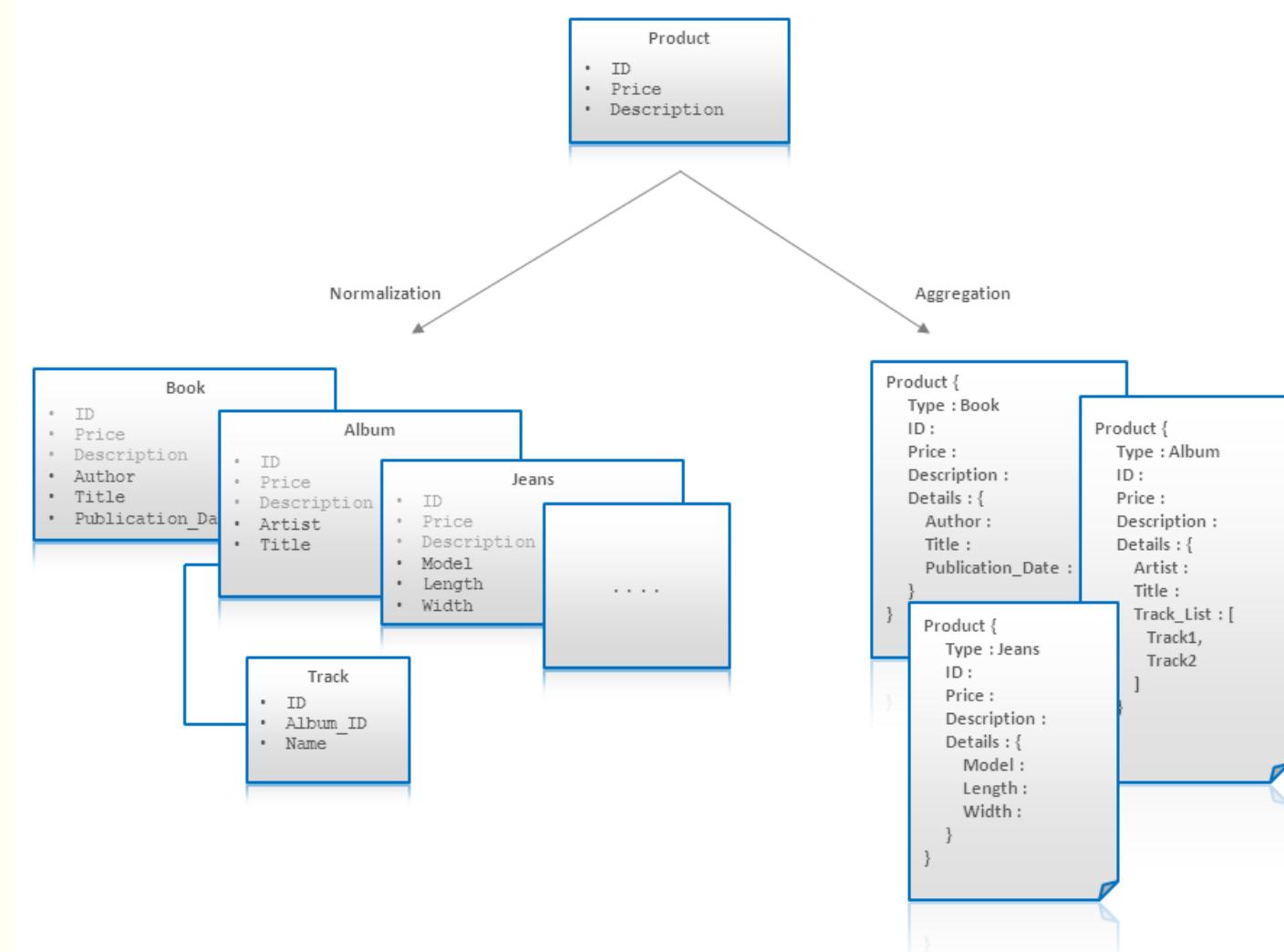
Aggregate model



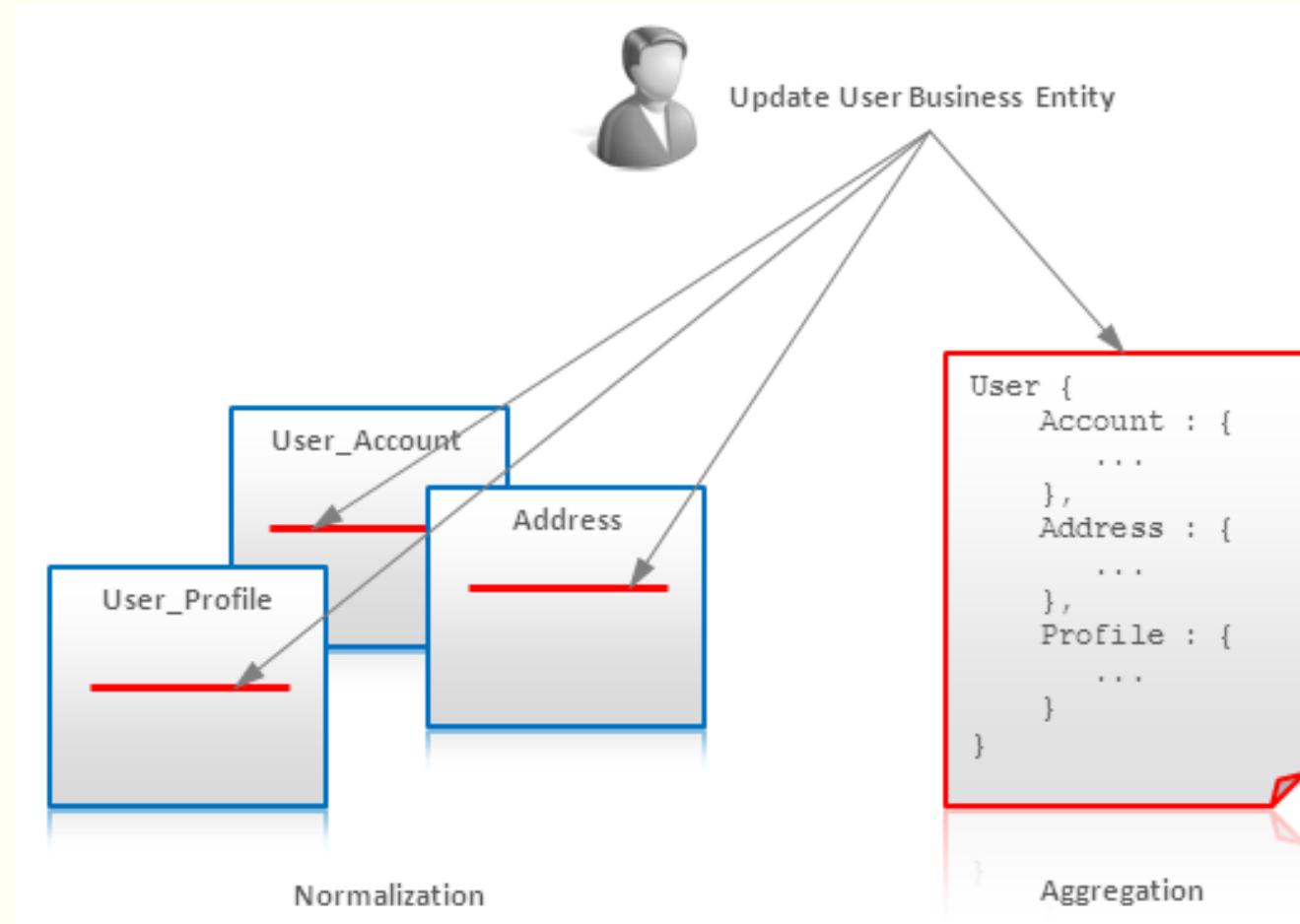
Aggregate Data

```
// in customers
{
  "customer": {
    "id": 1,
    "name": "Martin",
    "billingAddress": [{"city": "Chicago"}],
    "orders": [
      {
        "id":99,
        "orderItems": [
          {
            "productId":27,
            "price": 32.45,
            "productName": "NoSQL Distilled"
          }
        ],
        "shippingAddress": [{"city": "Chicago"}]
        "orderPayment": [
          {
            "ccinfo": "1000-1000-1000-1000",
            "txnId": "abelif879rft",
            "billingAddress": {"city": "Chicago"}
          }
        ],
      }
    ]
  }
}
```

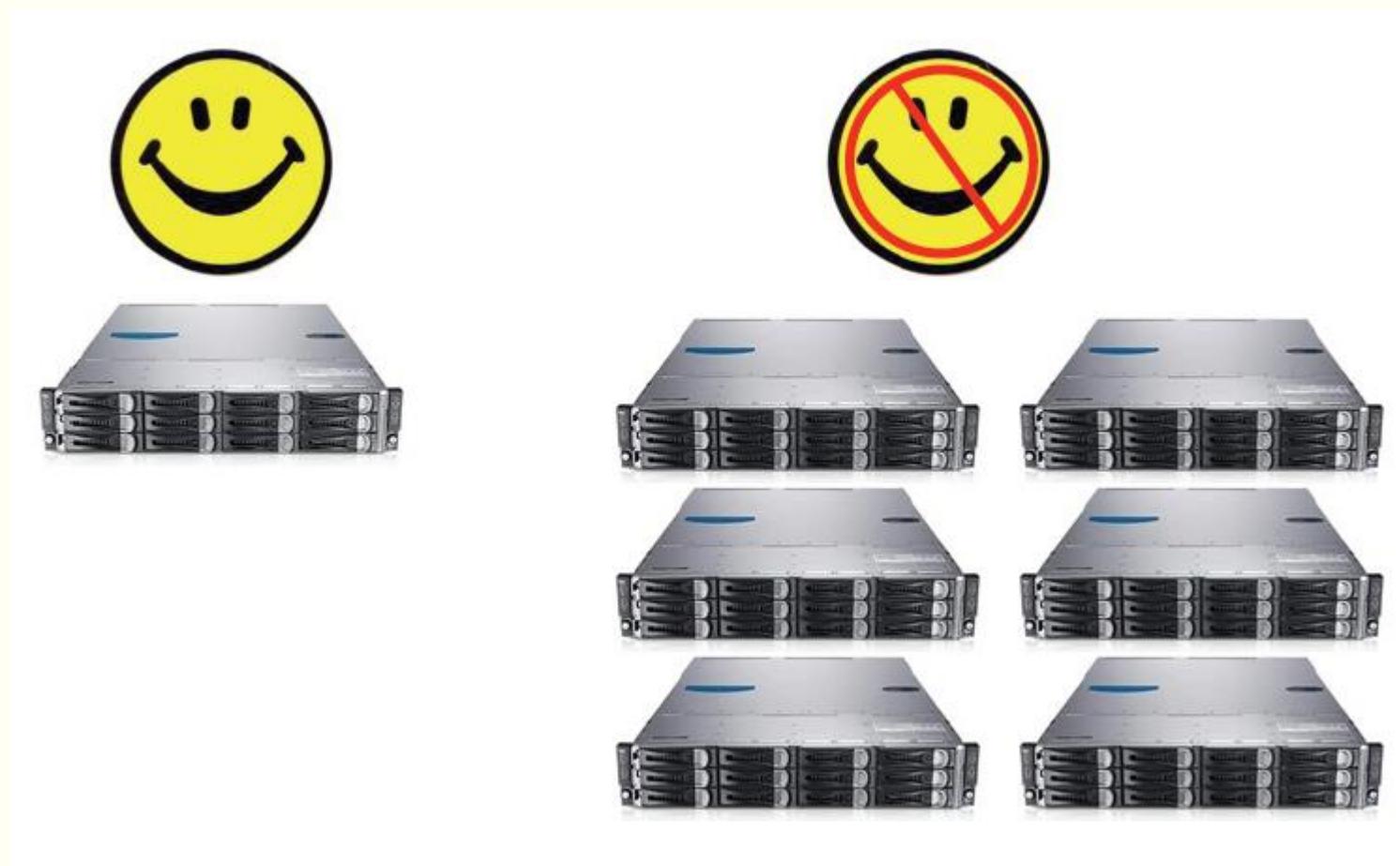
Normalisation vs Agrégation



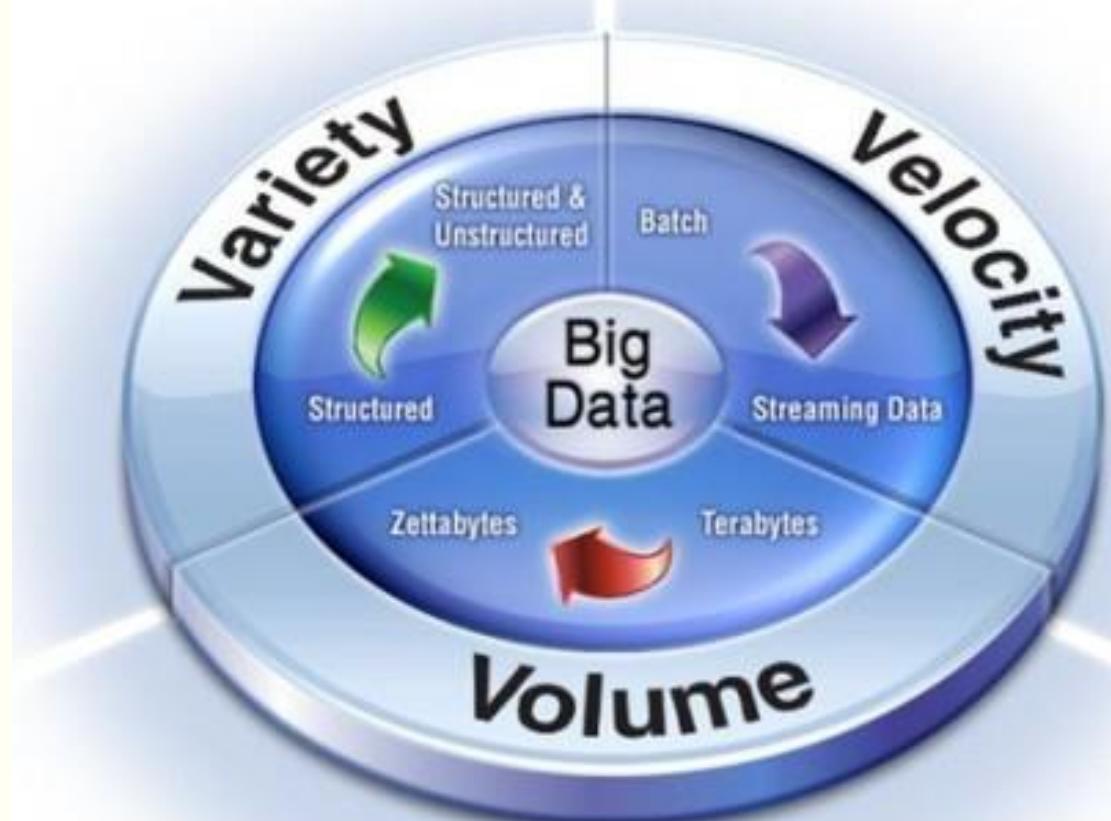
Normalisation vs Agrégation



RDBMS et Clusters



Big Data



Les évolutions...

- **Nouvelles Données :**
 - Web 2.0 : Facebook, Twitter, news, blogs,
 - Flux : capteurs, GPS, ...
- **Nouveaux Traitements :**
 - Moteurs de recherche , Extraction, analyse, ...
 - Recommandation, filtrage collaboratif, ...
- **Nouvelles Infrastructures :**
 - Cluster, réseaux mobiles, microprocesseurs multi-coeurs, ...
- → très gros volumes, données pas ou faiblement structurées
- → transformation, agrégation, indexation
- → distribution, parallélisation, redondance

Le NoSQL

Les bases NoSQL (comprendre Not Only SQL) répondant aux problématiques:

- de hautes disponibilités,
- grandes performances en lecture et/ou écriture
- le traitement de grands volumes de données.

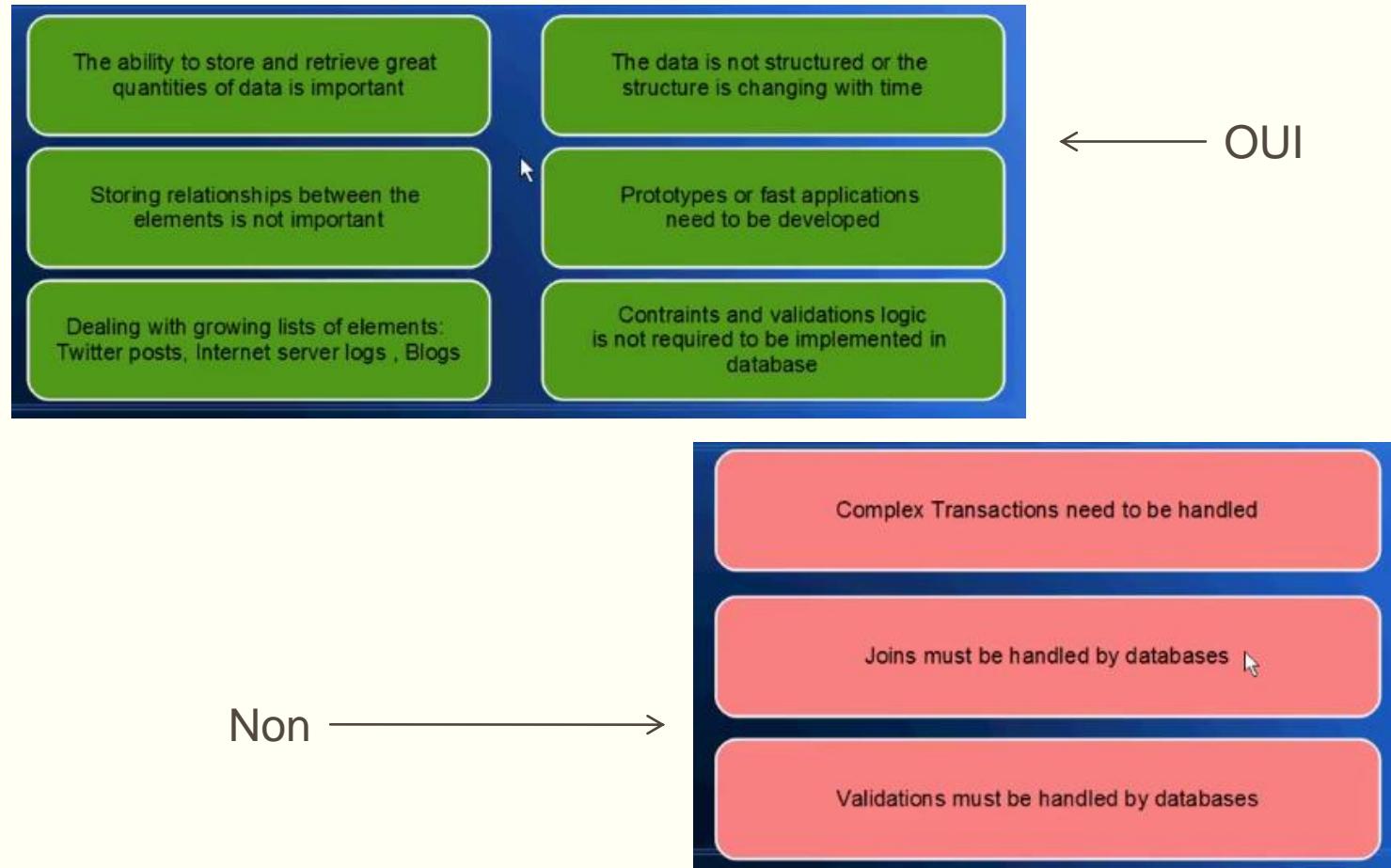
Le NoSQL

Un moteur NoSQL c'est...

- Une structure de données
- Une absence de contraintes
- Une méthode de modélisation
- Un schéma de données... flexible
- Une relation avec un modèle objet
- Du requêtage
- Une absence de transactions et des compromis

Le NoSQL

Alors, SQL versus NoSQL ?

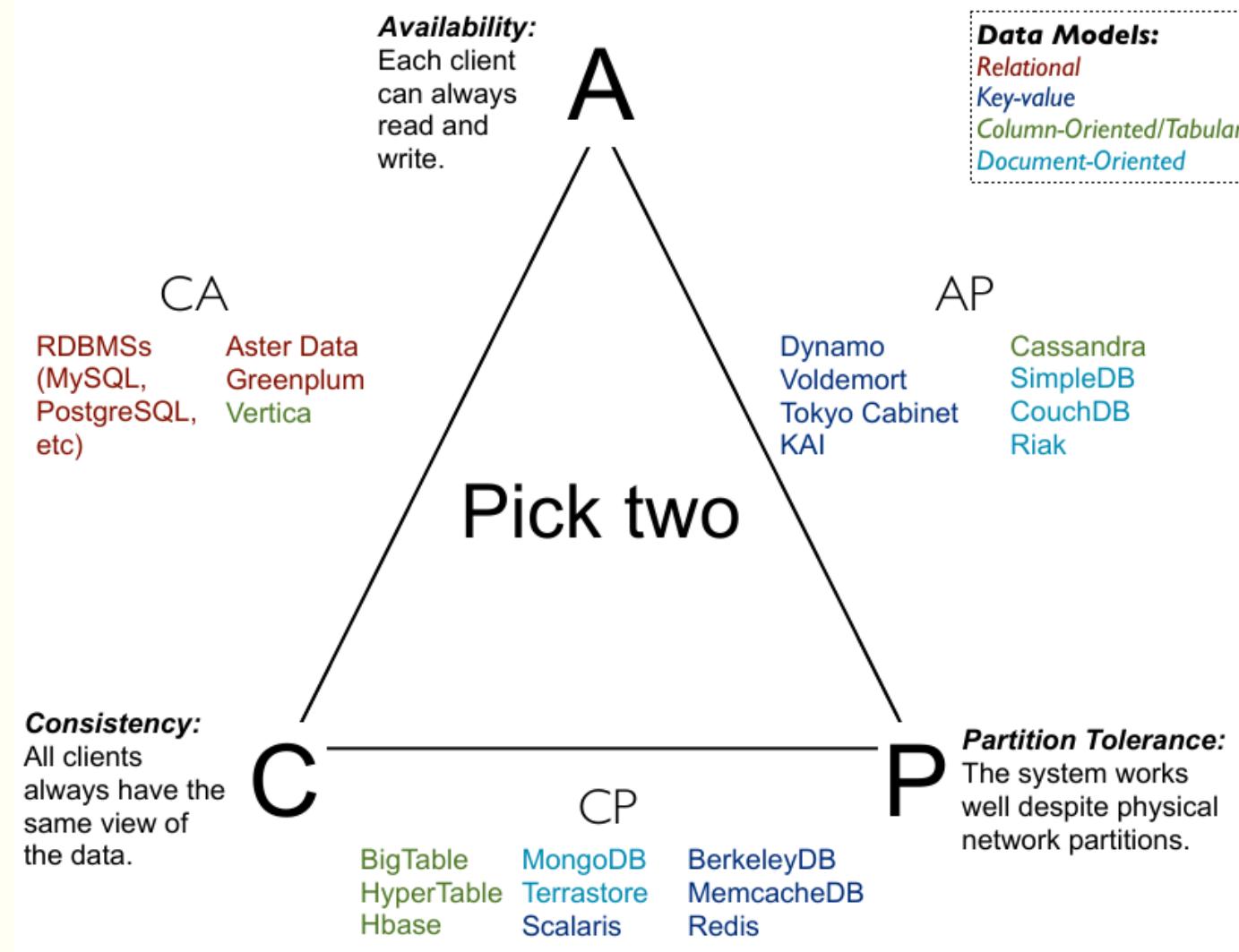


Théorème CAP

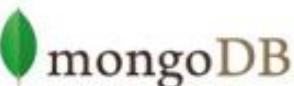
Théorème CAP (proposé par Brewer, 2000 et démontré par Gilbert et Lynch 2002)

- Dans un environnement distribué, il n'est pas possible de respecter simultanément :
 - **C** : Cohérence
 - **A** : Disponibilité
 - **P** : Résistance au morcellement
- On peut, en revanche, respecter deux contraintes

Guide visuel des solutions NoSQL



Les solutions NoSQL

Document Database	Graph Databases
  	 
Wide Column Stores	Key-Value Databases
  	   

Les solutions NoSQL

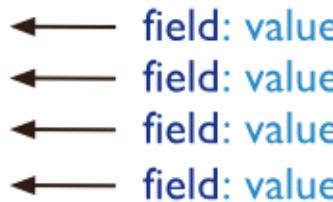
- **Clé/valeur**. à la manière des tableaux associatifs des langages de programmation. A une clé correspond une valeur.
- **Orienté colonne** : à une clé correspond un ensemble de colonne, chacune ayant une valeur.
- **Orienté document** : à une clé correspond des ensembles champs/valeurs qui peuvent être hiérarchisés
- **Orienté graphe** : les données sont modélisées sous forme de nœuds qui ont des liaisons entre eux.

MongoDB

MongoDB est une base de données orientée document sponsorisée par **10gen**.

- Un « document » est une entrée dans une base de donnée.
- Du JSON que Mongo stock en binaire(BSON)

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



MongoDB

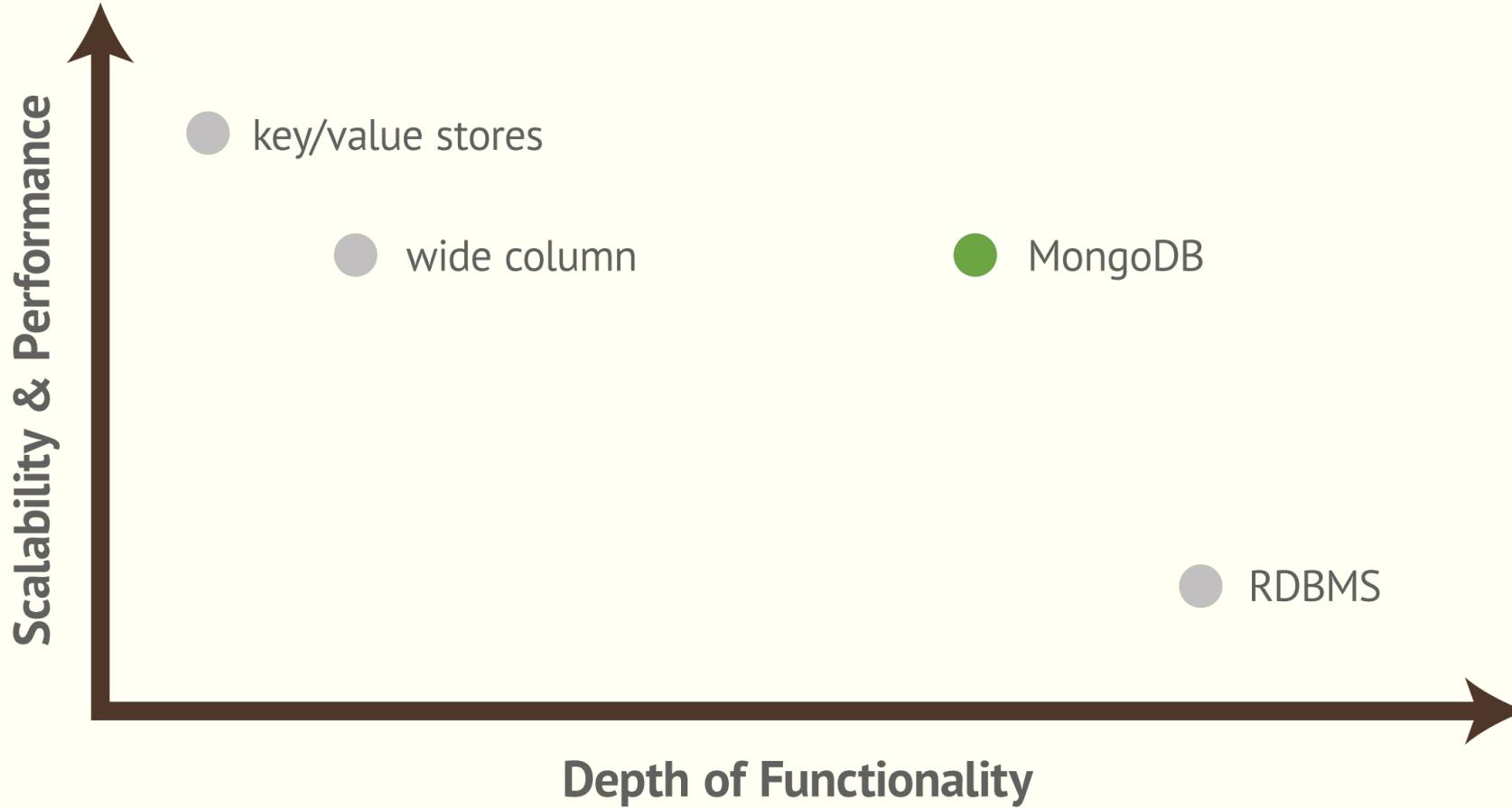
- Base orientée document signifie :
 - que les objets stockés sont représentés sous la forme d'un document **BSON**
 - permet facilement de se mapper sur les objets que l'on manipule dans nos programmes.
- MongoDB est **schemaless**.
 - aucun schéma de document n'est nécessaire pour pouvoir stocker les données.

MongoDB

MongoDB est un SGBD :

- orienté documents
- libre
- scalable : réplication, auto-sharding
- flexible : pas de schéma de données, full-text index
- écrit en C++.

Base de donnée opérationnelle



Modèle de donnée Document

Relationnel - Tables

PERSON

Pers_ID	Surname	First_Name	City	
0	Miller	Paul	London	
1	Ortega	Alvaro	Valencia	NO RELATION
2	Huber	Urs	Zurich	
3	Blanc	Gaston	Paris	
4	Bertolini	Fabrizio	Rome	

CAR

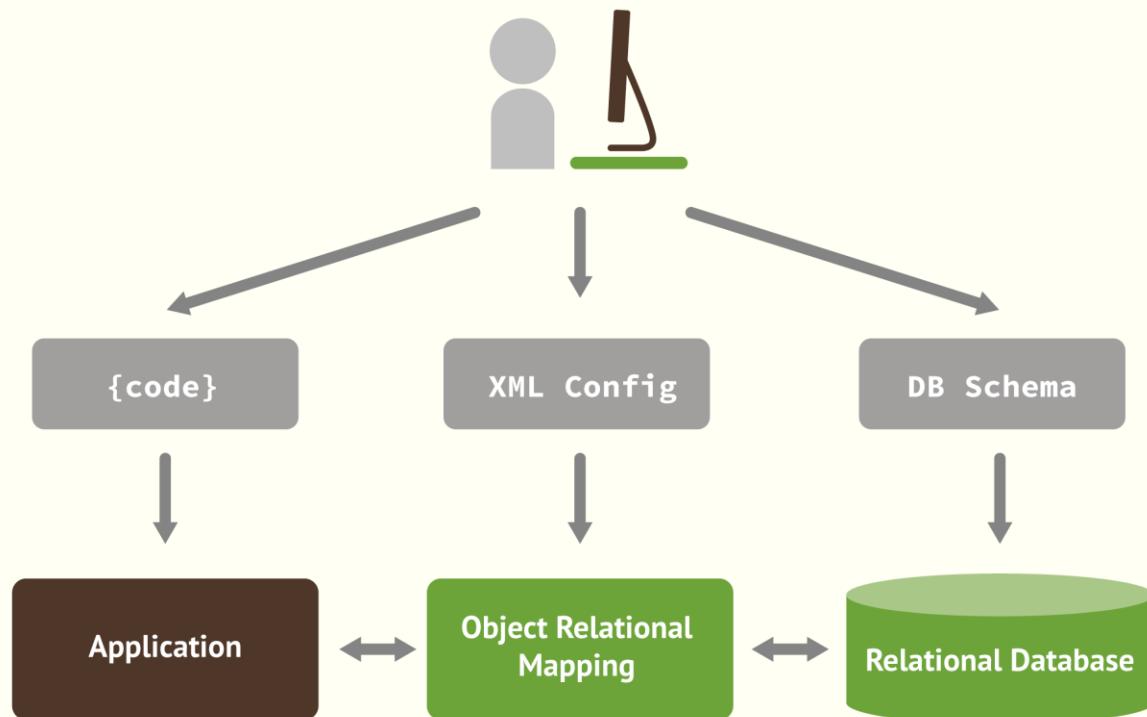
Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2



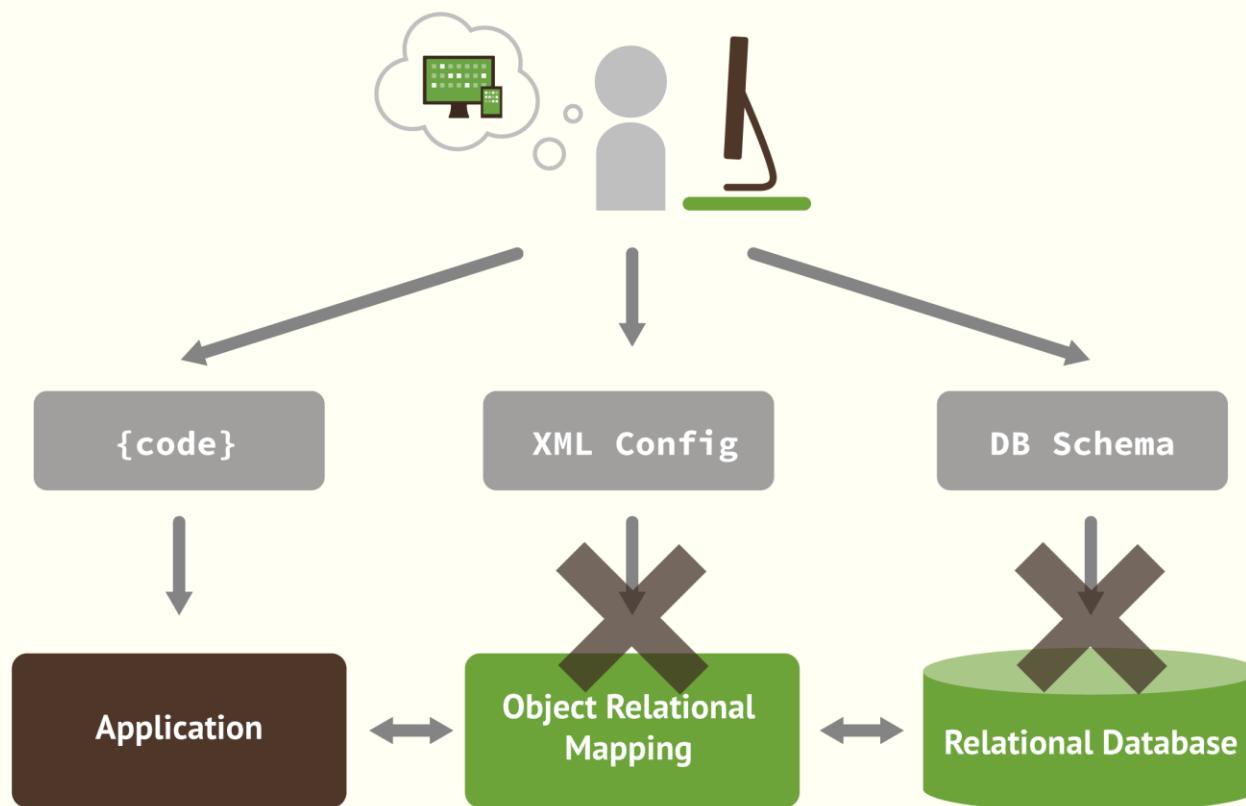
Document - Collections

```
{ first_name: 'Paul',  
  surname: 'Miller',  
  city: 'London',  
  location: {  
    type: "Point",  
    coordinates :  
      [-0.128, 51.507]  
  },  
  cars: [  
    { model: 'Bentley',  
      year: 1973,  
      value: 100000, ... },  
    { model: 'Rolls Royce',  
      year: 1965,  
      value: 330000, ... }  
  ]  
}
```

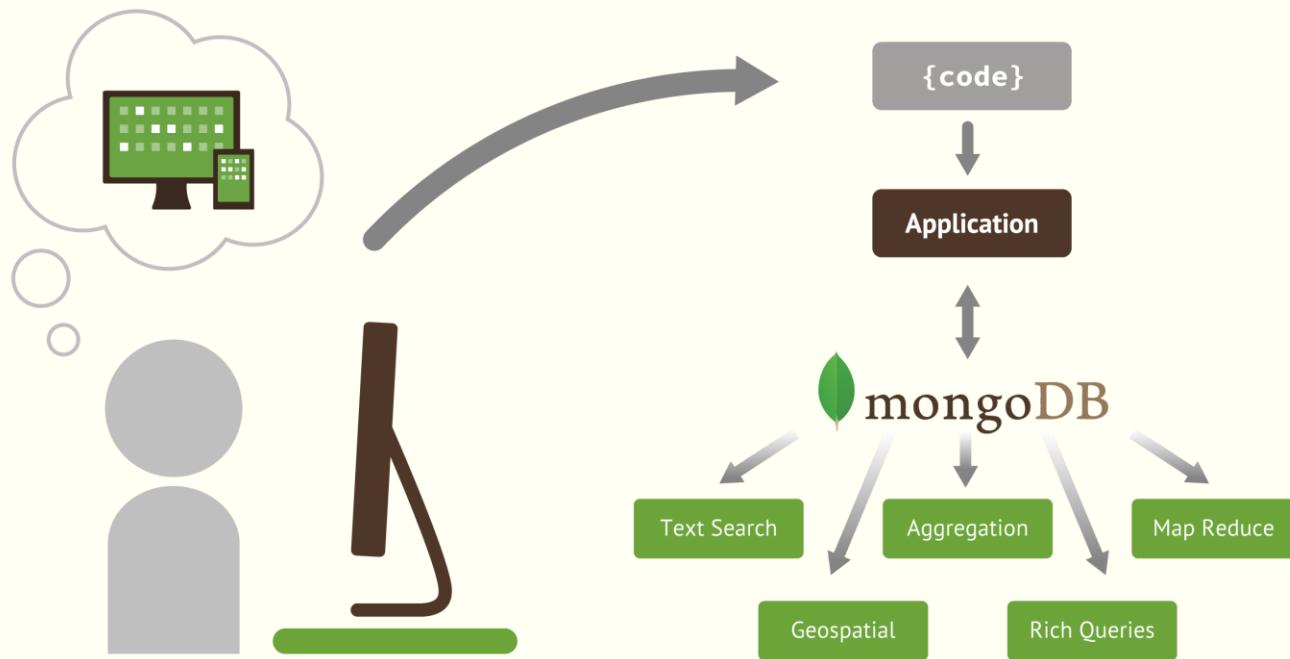
Développement traditionnel



Développement simplifié



Développement MongoDB



Shell and Drivers

Drivers

Drivers pour la majorité des langages de programmation et frameworks



Shell

Ligne de commande pour intéragir directement avec la base de données.

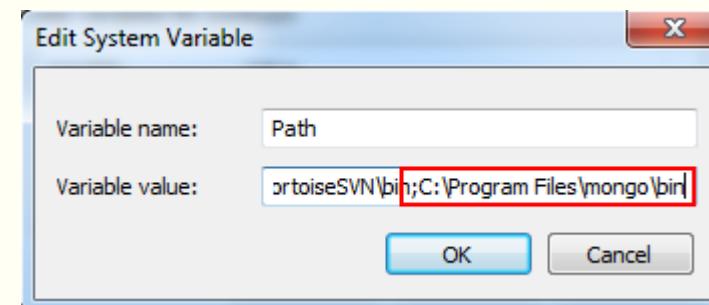
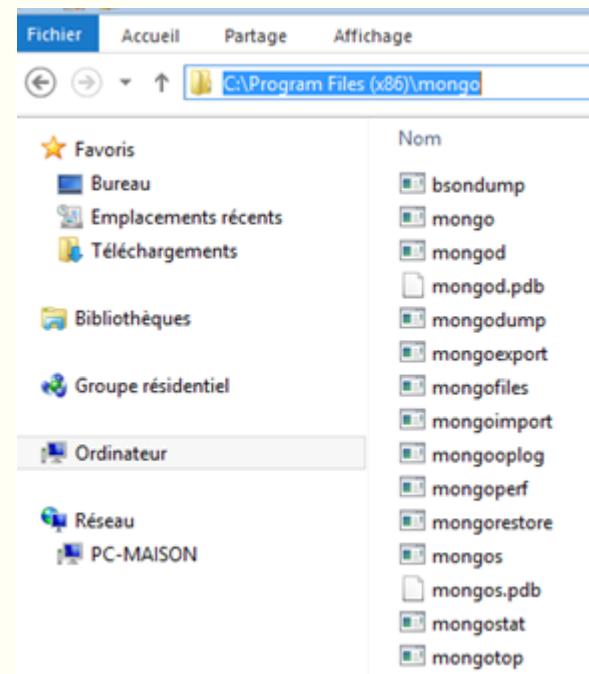
```
> db.collection.insert({company:"10gen", product:"MongoDB"})
>
> db.collection.findOne()
{
    "_id" : ObjectId("5106c1c2fc629bfe52792e86"),
    "company" : "10gen",
    "product" : "MongoDB"
}
```

Vocabulaire

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference

Installation de MongoDB

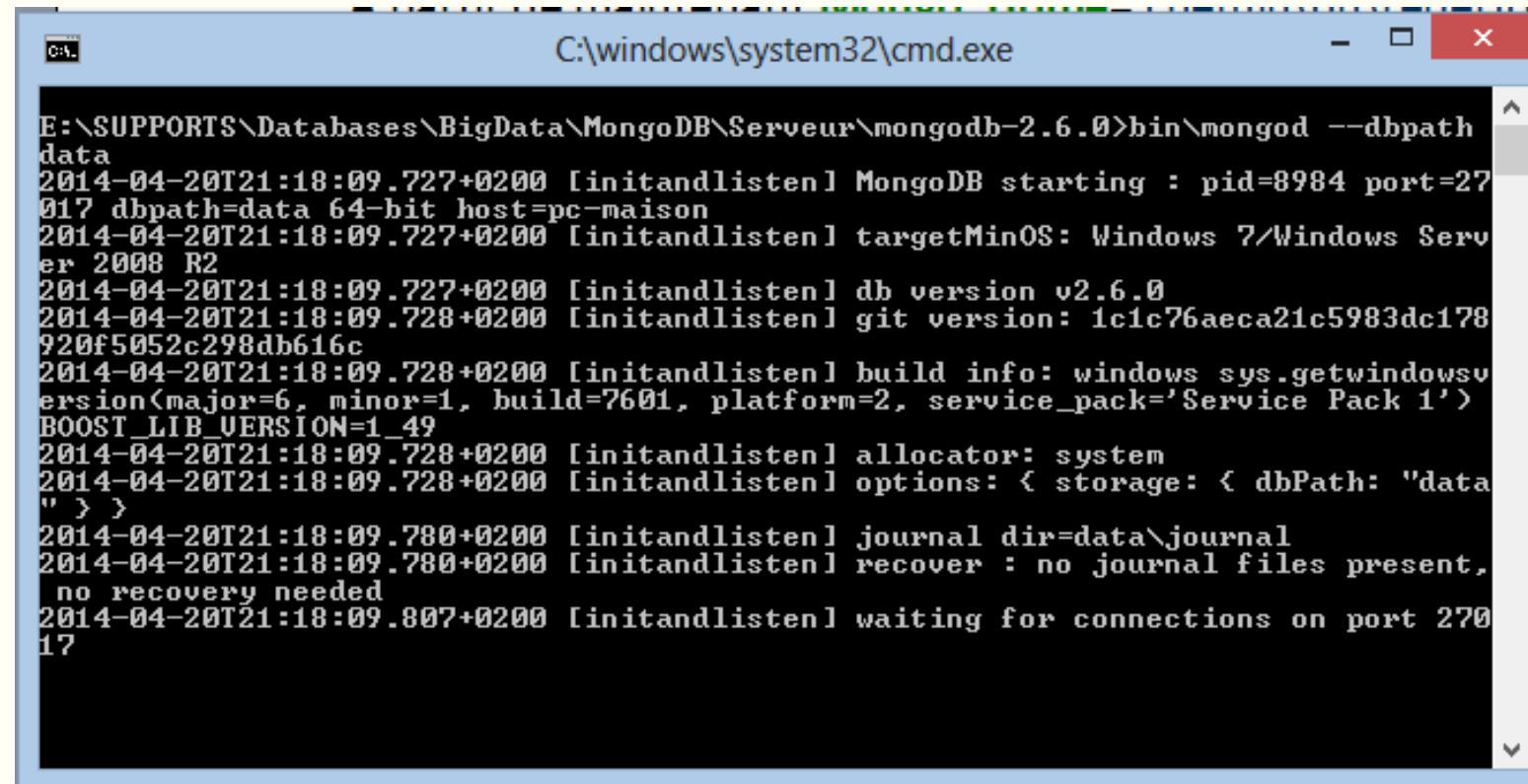
- Télécharger
 - <http://www.mongodb.org/downloads>
- Décompressez/Renommez
 - Déplacer dans par exemple:
C:\Programs Files\mongo
- Création du dossier de données
 - Dans par exemple : **c:\data**
- Configuration des variables d'environnement
 - Path: **C:\Programs Files\mongo\bin**



Démarrez une instance MongoDB

- Ouvrir un shell (ligne de commande DOS) et tapez:

mongod --dbpath c:\data



The screenshot shows a Windows Command Prompt window titled 'C:\windows\system32\cmd.exe'. The command entered is 'E:\SUPPORTS\DATABASES\BigData\MongoDB\Serveur\mongodb-2.6.0\bin\mongod --dbpath data'. The output text is as follows:

```
E:\SUPPORTS\DATABASES\BigData\MongoDB\Serveur\mongodb-2.6.0\bin\mongod --dbpath
data
2014-04-20T21:18:09.727+0200 [initandlisten] MongoDB starting : pid=8984 port=27
017 dbpath=data 64-bit host=pc-maison
2014-04-20T21:18:09.727+0200 [initandlisten] targetMinOS: Windows 7/Windows Serv
er 2008 R2
2014-04-20T21:18:09.727+0200 [initandlisten] db version v2.6.0
2014-04-20T21:18:09.728+0200 [initandlisten] git version: 1c1c76aec21c5983dc178
920f5052c298db616c
2014-04-20T21:18:09.728+0200 [initandlisten] build info: windows sys.getwindowsu
ersion(major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1')
BOOST_LIB_VERSION=1_49
2014-04-20T21:18:09.728+0200 [initandlisten] allocator: system
2014-04-20T21:18:09.728+0200 [initandlisten] options: { storage: { dbPath: "data
" } }
2014-04-20T21:18:09.780+0200 [initandlisten] journal dir=data\journal
2014-04-20T21:18:09.780+0200 [initandlisten] recover : no journal files present,
no recovery needed
2014-04-20T21:18:09.807+0200 [initandlisten] waiting for connections on port 270
17
```

Connexion

Pour utiliser MongoDB, entrez simplement **mongo** dans un terminal.

```
>mongo
```

```
MongoDB shell version: 2.6
connecting to: test
```

Nous sommes connectés à la base « test » .

Si nous souhaitons nous connecter à une autre base – peoples – nous utiliserons :

```
> use peoples
switched to db peoples
```

Commandes de base

Syntaxe

show dbs Lister les bases de données

Syntaxe

use peoples Création ou Connexion à une base

Syntaxe

db.alpha.insert({nom:"Jean"}) création de collection
“alpha” et : insertion de document **{nom:"Jean"}**

Syntaxe

db Base sur laquelle votre session est en cours

Commandes de base

Syntaxe

db.dropDatabase() Supprimer une database

Syntaxe

db.createCollection(name,option) Création de collections (Tables)

Syntaxe

db.nomCollection.drop() Supprimer une collection

Syntaxe

Crtl+C Quitter.

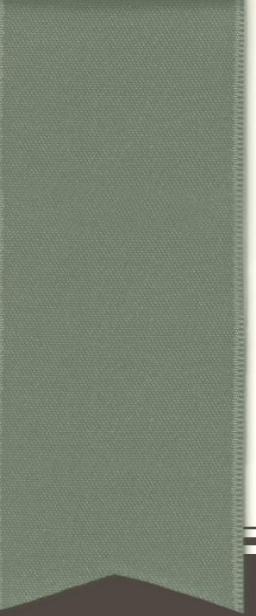
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Prise en main MongoDB

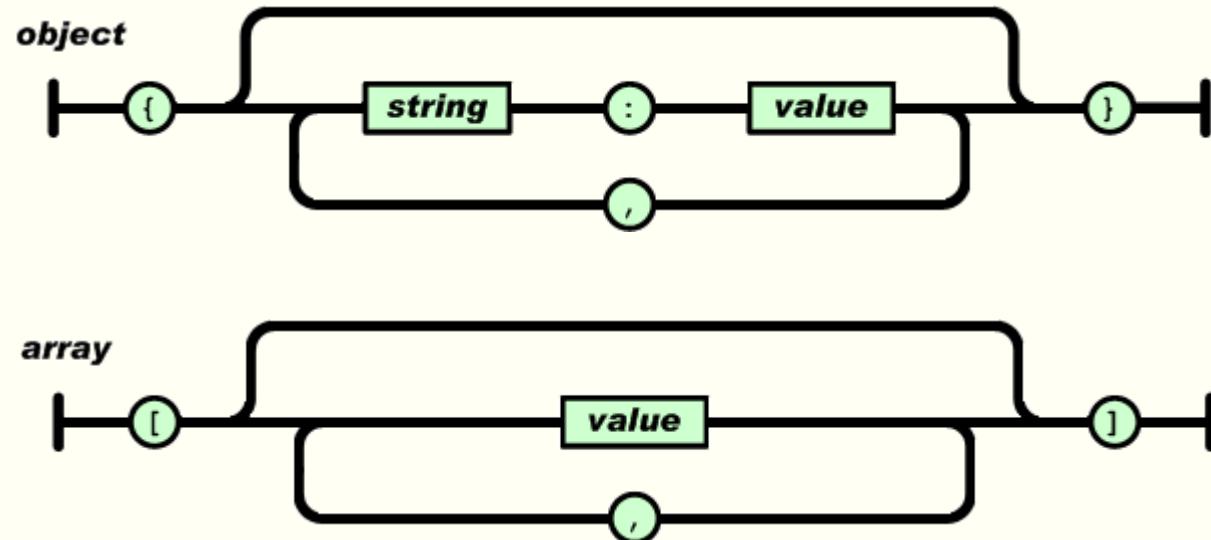


MODULE 2

Opération CRUD

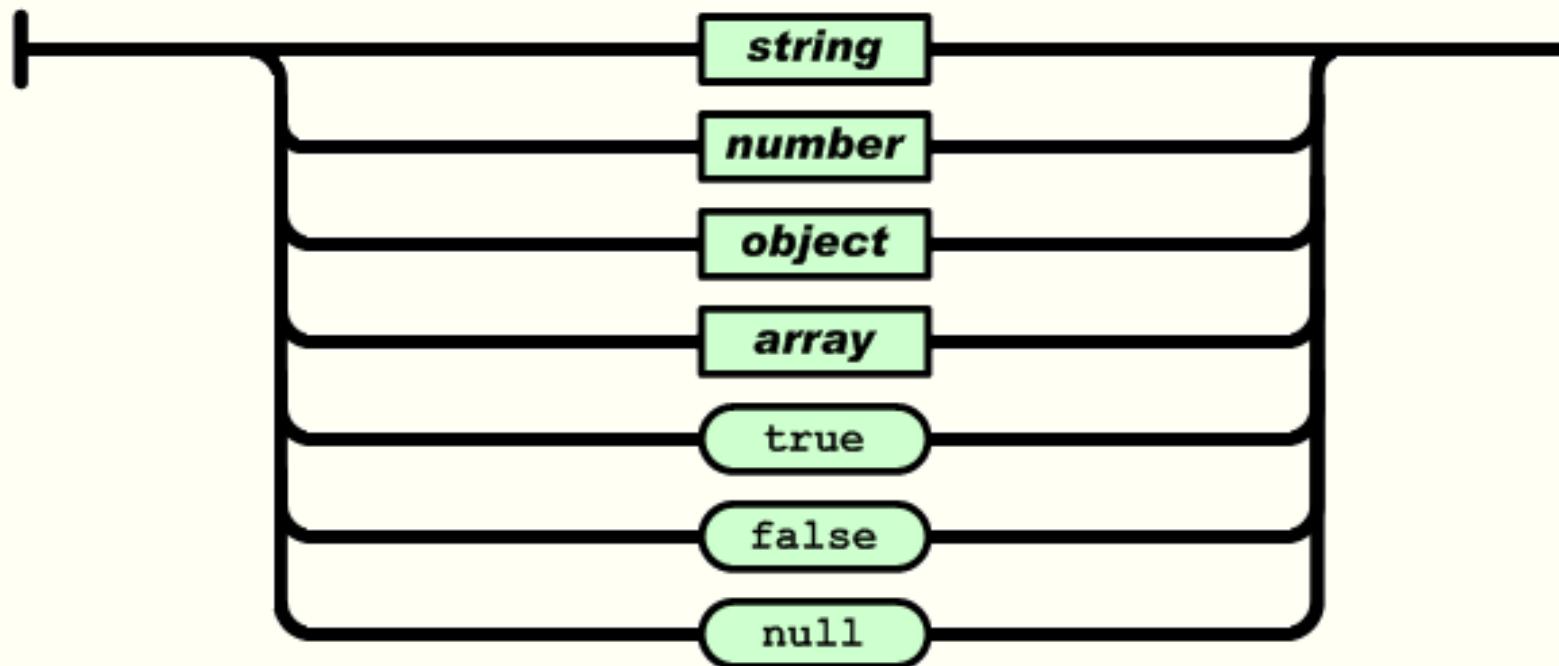
Document JSON-Format

- Official Site: <http://json.org/>
- Un document **json** est de la forme: **{ }** (object) ou **[]** (tableau).



Document JSON-value

value



Document JSON-Exemple-1

- **object:**

```
{ "firstName": "John" , "lastName": "Doe" }
```

- **array:**

```
[ "Text goes here", 29, true, null ]
```

- **Array with objects :**

```
[  
  { "name": "Dagny Taggart", "age": 39 },  
  { "name": "Francisco D'Anconia", "age": 40 },  
  { "name": "Hank Rearden", "age": 46 }  
]
```

Document JSON-Exemple-2

Object with nested array

```
{  
  "first": "John",  
  "last": "Doe",  
  "age": 39,  
  "sex": "M",  
  "salary": 70000,  
  "registered": true,  
  "interests": [ "Reading", "Mountain Biking", "Hacking" ]  
}
```

Object with nested object

```
{  
  "first": "John",  
  "last": "Doe",  
  "age": 39,  
  "sex": "M",  
  "salary": 70000,  
  "registered": true,  
  "favorites": {  
    "color": "Blue",  
    "sport": "Soccer",  
    "food": "Spaghetti"  
  }  
}
```

Document JSON-Exemple-3

Object with
nested arrays
and objects

```
{  
  "first": "John",  
  "last": "Doe",  
  "age": 39,  
  "sex": "M",  
  "salary": 70000,  
  "registered": true,  
  "interests": [ "Reading", "Mountain Biking", "Hacking" ],  
  "favorites": {  
    "color": "Blue",  
    "sport": "Soccer",  
    "food": "Spaghetti"  
  },  
  "skills": [  
    {  
      "category": "PHP",  
      "tests": [  
        { "name": "One", "score": 90 },  
        { "name": "Two", "score": 96 }  
      ]  
    },  
    {  
      "category": "CouchDB",  
      "tests": [  
        { "name": "One", "score": 32 },  
        { "name": "Two", "score": 84 }  
      ]  
    },  
    {  
      "category": "Node.js",  
      "tests": [  
        { "name": "One", "score": 97 },  
        { "name": "Two", "score": 93 }  
      ]  
    }  
  ]  
}
```

Opération CRUD

Les opération de CRUD MongoDB sont disponibles sous formes de fonctions / méthodes (API) d'un langage de programmation et non comme un langage séparé (i.e SQL).

Vocabulaire:

	MongoDB	SQL
CREATE	Insert	Insert
READ	Find	Select
UPDATE	Update	Update
DELETE	Remove	Delete

Langage MongoDB

▪ Mise-à-jour

- db.collection.insert()
- db.collection.update()
- db.collection.save()
- db.collection.remove()

▪ Interrogation

- db.collection.find()
- db.collection.findOne()

Insert, Find & Count

- **db.collection.insert({...})**

- The collection unique ID field is called “`_id`” and can be provided. If not provided an ObjectId will be generated based on the time, machine, process-id and process dependent counter.
- “`_id`” does not have to be a scalar value – it can be a document, e.g. `_id : {a:1, b:'ronald'}`

- **db.collection.find || findOne ({...}, {field1 : true, ...}).pretty()**

- //no argument will find all docs

- **db.collection.count({...})**

Création d'un document

- Trois façons d'insérer un document dans mongoDB:

- Insérer un document avec `insert`
- Insérer un document avec `update`
- Insérer un document avec `save`

Insérer un document avec **insert**

Syntaxe

```
db.collection.insert(document)  
db.collection.insert(documents)
```

- **document** est un tableau clefs / valeurs
- **documents** est une liste de tableaux clefs / valeurs
- **collection** est le nom de la collection à laquelle on souhaite ajouter le(s) document(s).

Rem: Si la collection n'existe pas au préalable, elle est créée (c'est de cette manière qu'on crée les collections)

Insérer un document avec `insert`

Syntaxe

```
db.collection.insert(document)  
db.collection.insert(documents)
```

```
db.users.insert ( ← collection  
  {  
    name: "sue", ← field: value  
    age: 26, ← field: value  
    status: "A" ← field: value  
  }  
)
```



```
INSERT INTO users ← table  
          ( name, age, status ) ← columns  
VALUES      ( "sue", 26, "A" ) ← values/row
```

Insérer un document avec **save**

Syntaxe

```
db.collection.save(document)
```

- Cas: Le document **contient** **_id** → Remplace le document(de la base) par le nouveau document.

```
> db.personne.save({_id:4, prenom:"Jean", nom:"Paul"})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- Cas : Le document **ne contient pas** **_id** → Il fait une insertion.

```
> db.personne.save({prenom:"Luc",nom:"Larue"})  
WriteResult({ "nInserted" : 1 })
```

MAJ d'un document avec **update**

Syntaxe

```
db.collection.update(criteria,donnée_maj)
db.collection.update(criteria,donnée_maj,multi)
db.collection.update(criteria,donnée_maj,upsert,multi)
```

- **criteria** est de la même forme que pour `find`
- **donnée_maj** est un tableau clefs / valeurs définissant des opérations sur les champs.
- **option**:
 - ***multi*** (booléen) :
 - `false` (défaut) : maj d'une occurrence trouvée (laquelle ?)
 - `true` : maj toutes les occurrence
 - ***upsert*** (booléen) :
 - `false` (défaut) : update
 - `true` : update or insert if not exists

MAJ d'un document avec **update**

Syntaxe

```
db.collection.update(criteria,donnée_maj,option)
```

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

```
UPDATE users           ← table  
SET     status = 'A'  ← update action  
WHERE   age > 18     ← update criteria
```

MAJ d'un tableau avec **update**

- `db.collection.update({ myQuery }, {myField: "newValue", ... })`
→ replaces the existing document
- `db.collection.update({ myQuery }, {$set : {myField: "newValue"}})`
→ Create or update myField
- `db.collection.update({ myQuery }, {$inc : {age: 1}})`
- `db.collection.update({ myQuery }, {$unset : {myField: 1}})`
- `db.collection.update({ myQuery }, {$set : {myField: "newValue"}}, {upsert: true})`
→ Create or update document specified by { myQuery } with myField

Arrays Update

- db.collection.update({ myQuery }, {\$set : {"myArray.2": "x"}}) -> Set 3rd position of Array
- db.collection.update({ myQuery }, {\$push : {myArray: "y"}})
- db.collection.update({ myQuery }, {\$addToSet : {myArray: "y"}}) //will only add if does not exist yet
- db.collection.update({ myQuery }, {\$pop : {myArray: 1}}) // pop right-most
- db.collection.update({ myQuery }, {\$pop : {myArray: -1}}) // pop left-most
- db.collection.update({ myQuery }, {\$pushAll : {myArray: ["a", "b", "c"]}})
- db.collection.update({ myQuery }, {\$pull : {myArray: "c"}}) // remove value "c"

Supprimer un document avec **remove**

Syntaxe

```
db.collection.remove()  
db.collection.remove(<query>)
```

- **sans paramètre ie {}**, tous les documents sont supprimés
(attention, donc)
- **query** est de la même forme que pour find, elle désigne les documents qui seront supprimés.

Supprimer un document avec **remove**

Syntaxe

```
db.collection.remove(<query>)
```

```
db.users.remove(  
  { status: "D" }  
)
```



```
DELETE FROM users  
WHERE status = 'D'
```



Lecture de document avec **find**

Syntaxe

```
db.collection.find()  
db.collection.find(<criteria>)  
db.collection.find(<criteria>,<projection>)
```

- **sans paramètre**, tous les documents sont renvoyés
- **criteria** est un tableau clefs / valeurs spécifiant des opérateurs sur les champs des documents recherchés
- **projection** est un tableau permettant de limiter les champs que l'on souhaite consulter dans les documents recherchés (cette option sera traitée ultérieurement)

- Exemple: `db.etudiants.find()`
`db.etudiants.find({ 'prenom': 'Camille' })`

Lecture de document avec **find**

Syntaxe

```
db.collection.find(<criteria>,<projection>)
```

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

{ age: 18, ... }
{ age: 28, ... }
{ age: 21, ... }
{ age: 38, ... }
{ age: 18, ... }
{ age: 38, ... }
{ age: 31, ... }

Query Criteria

{ age: 18, ... }
{ age: 18, ... }

Projection

{ name: "al" }
{ name: "bob" }

Results

users

Lecture de document avec **findOne**

Syntaxe

```
db.collection.findOne()  
db.collection.findOne(<criteria>)  
db.collection.findOne(<criteria>,<projection>)
```

- La fonction `findOne` fait la même chose que `find` mais sans s'embarrasser d'un curseur, lorsqu'on souhaite récupérer un document unique (par son identifiant, par exemple).
- Si la requête désigne plusieurs documents, le premier trouvé est renvoyé.

Curseurs

- `myCursor = db.collection.find(); null;`
 - append null as not to print out the cursor immediately
- `myCursor.hasNext() myCursor.next()`
- `myCursor.skip(2).limit(5).sort({name : -1}); null;`
 - modifies the query executed on the server

```
>var cursor = db.articles.find ( { 'author' : 'Tug Grall' } )

>cursor.hasNext()
true

>cursor.next()
{
  '_id' : ObjectId(...),
  'text': 'Article content...',
  'date' : ISODate(...),
  'title' : 'Intro to MongoDB',
  'author' : 'Dan Roberts',
  'tags' : [ 'mongodb', 'database', 'nosql' ]
}
```

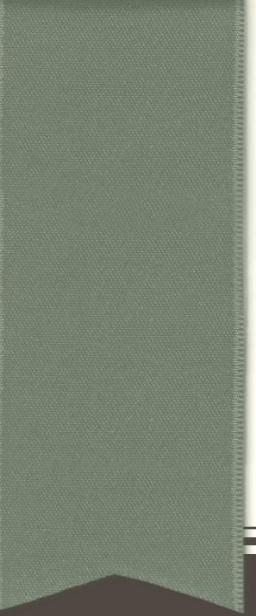
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Opérations CRUD



MODULE 3

Les requêtes

Quelques commandes pratiques

Sous le répertoire *bin* se trouvent plusieurs exécutables.

Nom	Modifié le
bsondump	07/04/2014 01:23
mongo	07/04/2014 01:10
mongod	07/04/2014 01:21
mongod.pdb	07/04/2014 01:21
mongodump	07/04/2014 01:22
mongoexport	07/04/2014 01:22
mongofiles	07/04/2014 01:23
mongoimport	07/04/2014 01:22
mongooplog	07/04/2014 01:22
mongoperf	07/04/2014 01:23
mongorestore	07/04/2014 01:22
mongos	07/04/2014 01:21
mongos.pdb	07/04/2014 01:21
mongostat	07/04/2014 01:22
mongotop	07/04/2014 01:22

mongodump

Syntaxe

```
mongodump --db databaseCible --out databaseTarget
```

Permet de faire un dump(Sauvegarde) de la base de données :

```
mongodb-2.6.0\bin>mongodump --db test --out ./data/mongodump
```

```
connected to: 127.0.0.1
```

```
2014-05-04T13:19:12.105+0200 DATABASE: test ...29467 documents.
```

➤ **A Noter :** Sous le répertoire **/data/dumpmongo** nous trouverons un nouveau dossier « test » qui contient deux fichiers : system.indexes.bson et products.bson

mongorestore

Syntaxe

```
mongorestore --db databaseTarget --drop databaseOrig
```

- Permet d'installer une base de données à partir d'un dump dans un répertoire local ou distant en utilisant le paramètre –host « l'ip remote »

```
mongorestore --db test --drop ./data/mongodump/test
```

```
connected to: 127.0.0.1
```

```
2014-05-04T13:19:12.105+0200 DATABASE: test ...29467 documents.
```

mongoimport

Syntaxe

```
mongoimport --db test --c collection < nomFile
```

Exemple:

```
>mongoimport -d students -c grades < grades.js
connected to: 127.0.0.1
2014-04-26T14:46:35.460+0200 check 9 800
2014-04-26T14:46:35.461+0200 imported 800 objects
```

Les opérateurs

\$each, \$slice, \$sort, \$inc, \$push

\$inc, \$rename, \$setOnInsert, \$set, \$unset, \$max, \$min

\$, \$addToSet, \$pop, \$pullAll, \$pull, \$pushAll, \$push

\$each, \$slice, \$sort, \$size

\$gt, \$gte, \$in, \$lt, \$lte, \$ne, \$nin

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

Opérateurs de Comparaison

Syntaxe

```
{myField: {$gt: 100, $lt: 10}}
```

- **\$gt, \$gte, \$in, \$lt, \$lte, \$ne**

Exemple:

```
> db.grades.find({score:{$gt:95}})  
{ "_id" : 1, "student_id" : 1, "type" : "quiz", "score" : 96.76851542258362 }  
{ "_id" : 2, "student_id" : 2, "type" : "homework", "score" : 97.75889721343528 }  
...  
Type "it" for more
```

```
> db.grades.find({score:{$gt:85,$lt:95},type:"quiz"})  
{ "_id" : 1, "student_id" : 20, "type" : "quiz", "score" : 92.76554684090782 }  
{ "_id" : 2, "student_id" : 22, "type" : "quiz", "score" : 86.0800081592629 }  
...
```

Opérateur sur les éléments

Syntaxe

```
{myField: {$exists: true}} , {myField: {$type: 2}}
```

- **\$exists**: checks if particular key exists in document
- **\$type**: 2=String as defined in BSON spec

Exemple:

```
> db.people.find({profession:{$exists:true}})  
{ "_id" : 1, "name" : "Smith", "age" : 30, "profession" : "hacker" }  
{ "_id" : 2, "name" : "Jones", "age" : 35, "profession" : "boulanger" }
```

```
> db.people.find({name:{$type:2}})  
{ "_id" : ObjectId("535bb427a8f64ced71fb8b74"), "name" : "Bob" }  
{ "_id" : ObjectId("535bb442a8f64ced71fb8b75"), "name" : "Charlie" }  
{ "_id" : ObjectId("535bb44ca8f64ced71fb8b76"), "name" : "Dave" }
```

Opérateurs Evalué

Utilisé pour requêter la base de données

Syntaxe

\$mod, \$regex, \$search

Exemple:

```
> db.people.find({name:{$regex:"e$"}})
{ "_id" : ObjectId("535bb442a8f64ced71fb8b75"), "name" : "Charlie" }
{ "_id" : ObjectId("535bb44ca8f64ced71fb8b76"), "name" : "Dave" }
```

```
> db.people.find({name:{$regex:"^C"}})
{ "_id" : ObjectId("535bb442a8f64ced71fb8b75"), "name" : "Charlie" }
```

Opérateurs: Logique

Syntaxe

`{$or: [{...}, {...}, ...]}`

- `$and`, `$not`, `$ne`
- `$not` - negates result of other operation or regular expression query
- tags: `{$ne: "gardening"}` // works on keys pointing to single values or arrays – inefficient – can't use indexes

```
> db.people.find({$or:[{name:$regex:"e$"},{age:$exists:true}]})  
{ "_id" : ObjectId("535bb442a8f64ced71fb8b75"), "name" : "Charlie" }  
{ "_id" : ObjectId("535bb44ca8f64ced71fb8b76"), "name" : "Dave" }
```

```
> db.people.find({$and:[{name:$gt:"C"},{name:$regex:"a"}]})  
{ "_id" : ObjectId("535bb442a8f64ced71fb8b75"), "name" : "Charlie" }  
{ "_id" : ObjectId("535bb44ca8f64ced71fb8b76"), "name" : "Dave" }  
{ "_id" : ObjectId("535bb456a8f64ced71fb8b77"), "name" : "Edgar" }
```

Requête sur une liste (embedded)

```
> db.accounts.find().pretty()
{
  "_id" : ObjectId("535bd635a8f64ced71fb8b7c"),
  "name" : "George",
  "favorites" : [
    "ice cream",
    "pretzels"
  ]
}
```

```
> db.accounts.find({favorites:"beer",name:{$gt:"H"}})
{ "_id" : ObjectId("535bd667a8f64ced71fb8b7d"), "name" : "Howard", "favorites" : [ "pretzels", "beer" ] }
```

Opérateur sur une liste

Syntaxe

\$in, \$all

```
> db.accounts.find().pretty()
{
  "_id" : ObjectId("535bd635a8f64ced71fb8b7c"),
  "name" : "George",
  "favorites" : [
    "ice cream",
    "pretzels"
  ]
}
```

```
> db.accounts.find({favorites : { $all : ["pretzels","beer"]}} )
{ "_id" : 1, "name" : "Howard", "favorites" : [ "pretzels", "beer" ] }
{ "_id" : 2, "name" : "Irving", "favorites" : [ "pretzels", "beer", "cheese" ] }
```

```
> db.accounts.find({favorites : { $in : ["pretzels","beer"]}} )
{ "_id" : 3, "name" : "George", "favorites" : [ "ice cream", "pretzels" ] }
{ "_id" : 4, "name" : "Howard", "favorites" : [ "pretzels", "beer" ] }
```

Requête avec la notation «.»

Opérateur sur une liste

Syntaxe

**\$push, \$pop, \$pull, \$pushAll, \$pullAll,
\$addToSet**

```
> db.arrays.update({_id:0 },{$set:{a:2":5}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.arrays.update({_id:0 },{$push : {a:6}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.arrays.update({_id:0 },{$pop : {a:1}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

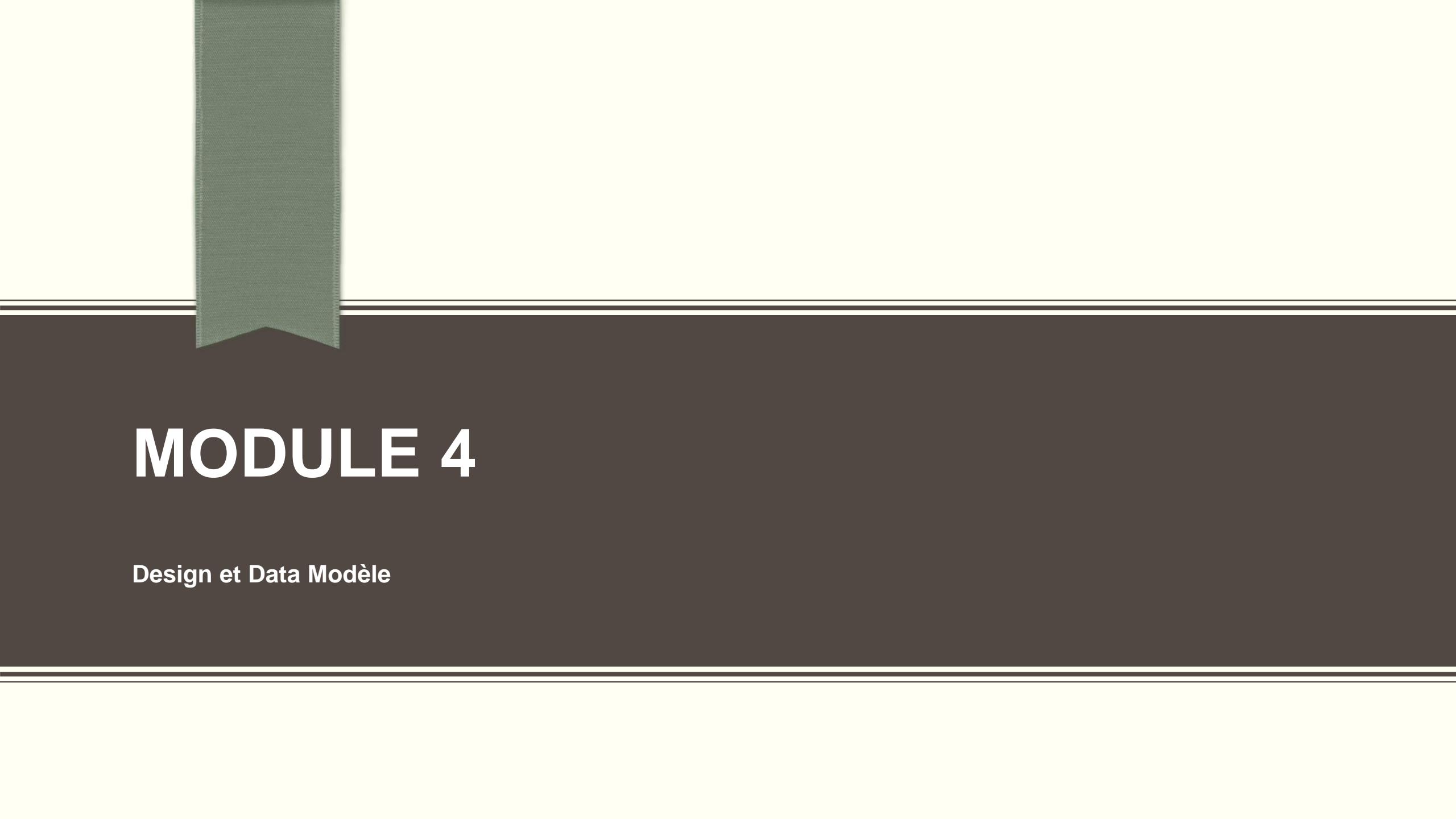
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Opérateurs



MODULE 4

Design et Data Modèle

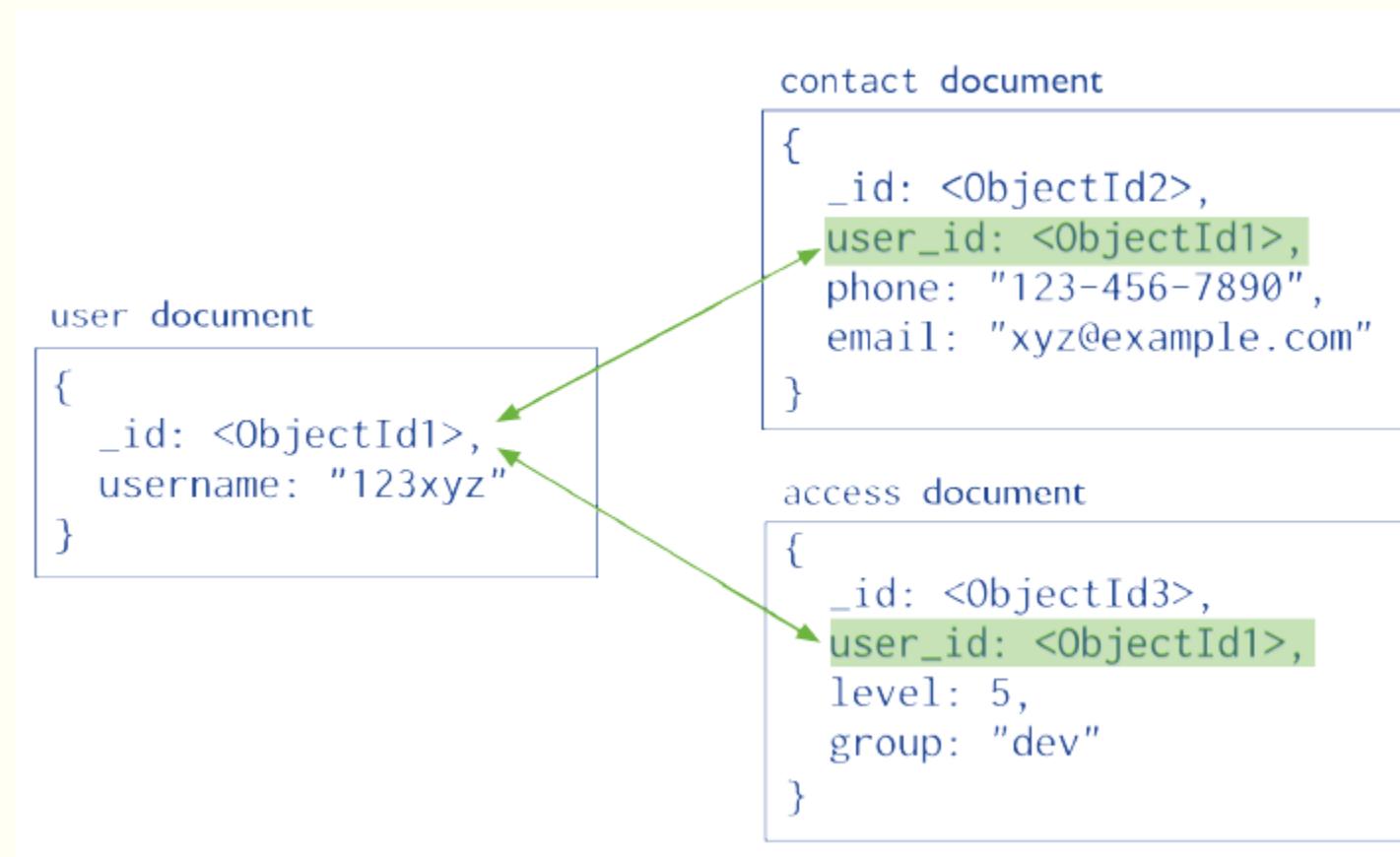
Vocabulaire

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference

There are some patterns

- **Embedding**
- **Linking(par reference)**

Linking



Embedding & Linking

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```



Embedded sub-document

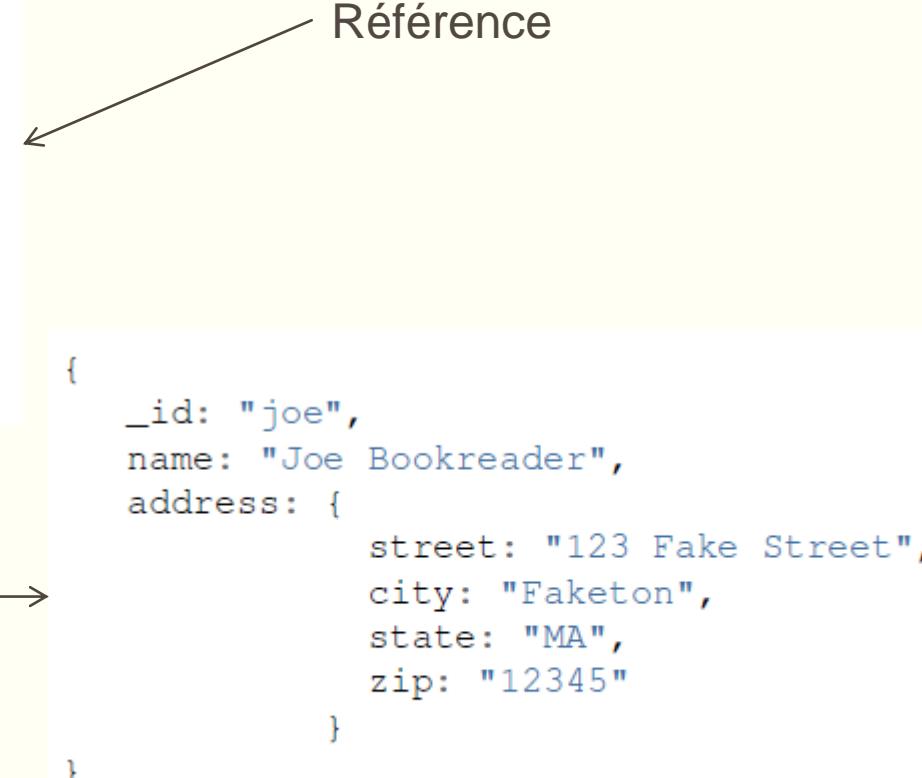
Embedded sub-document

One to One Pattern

```
{  
  _id: "joe",  
  name: "Joe Bookreader"  
}  
  
{  
  patron_id: "joe",  
  street: "123 Fake Street",  
  city: "Faketown",  
  state: "MA",  
  zip: "12345"  
}  
  
{  
  _id: "joe",  
  name: "Joe Bookreader",  
  address: {  
    street: "123 Fake Street",  
    city: "Faketown",  
    state: "MA",  
    zip: "12345"  
  }  
}
```

Embedded →

Référence



One to many Pattern

3 solutions possibles

- Embedding
- One To Few
- One To Many

One to many vs Embedding

```
{  
  title: "MongoDB: The Definitive Guide",  
  author: [ "Kristina Chodorow", "Mike Dirolf" ],  
  published_date: ISODate("2010-09-24"),  
  pages: 216,  
  language: "English",  
  publisher: {  
    name: "O'Reilly Media",  
    founded: 1980,  
    location: "CA"  
  }  
}  
  
{  
  title: "50 Tips and Tricks for MongoDB Developer",  
  author: "Kristina Chodorow",  
  published_date: ISODate("2011-05-06"),  
  pages: 68,  
  language: "English",  
  publisher: {  
    name: "O'Reilly Media",  
    founded: 1980,  
    location: "CA"  
  }  
}
```

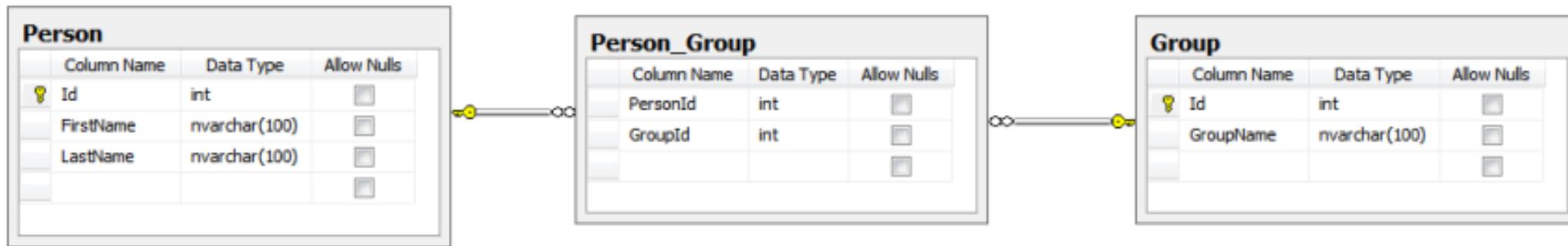
One to Few

```
{  
  name: "O'Reilly Media",  
  founded: 1980,  
  location: "CA",  
  books: [12346789, 234567890, ...]  
}  
  
{  
  _id: 123456789,  
  title: "MongoDB: The Definitive Guide",  
  author: [ "Kristina Chodorow", "Mike Dirolf" ],  
  published_date: ISODate("2010-09-24"),  
  pages: 216,  
  language: "English"  
}  
  
{  
  _id: 234567890,  
  title: "50 Tips and Tricks for MongoDB Developer",  
  author: "Kristina Chodorow",  
  published_date: ISODate("2011-05-06"),  
  pages: 68,  
  language: "English"  
}
```

Linking vs. Embedding

- Embedding représente une pré-jointure de données.
- Des opérations au niveau du document sont faciles à gérer pour le serveur.
- Embarquer lorsque les objets «Many» apparaissent toujours avec leurs parents.
- Relier lorsque vous avez besoin de plus de flexibilité.

Many to many relationship



Many to many relationship

```
{  
  "_id": "4e54ed9f48dc5922c0094a43",  
  "firstName": "Joe",  
  "lastName": "Mongo",  
  "groups": [ "4e54ed9f48dc5922c 0094a42",  
             "4e54ed9f48dc5922c0094a41" ]  
}  
  
{  
  "_id": "4e54ed9f48dc5922c0094a40",  
  "firstName": "Sally",  
  "lastName": "Mongo",  
  "groups": [ "4e54ed9f48dc5922c0094a42" ]  
}  
  
{  
  "_id": "4e54ed9f48dc5922c0094a42",  
  "groupName": "mongoDB User",  
  "persons": [  
    "4e54ed9f48dc5922c0094a43",  
    "4e54ed9f48dc5922c0094a40" ]  
}  
  
{  
  "_id": "4e54ed9f48dc5922c0094a41",  
  "groupName": "mongoDB Administrator",  
  "persons": [  
    "4e54ed9f48dc5922c0094a43" ]  
}
```

Many to many

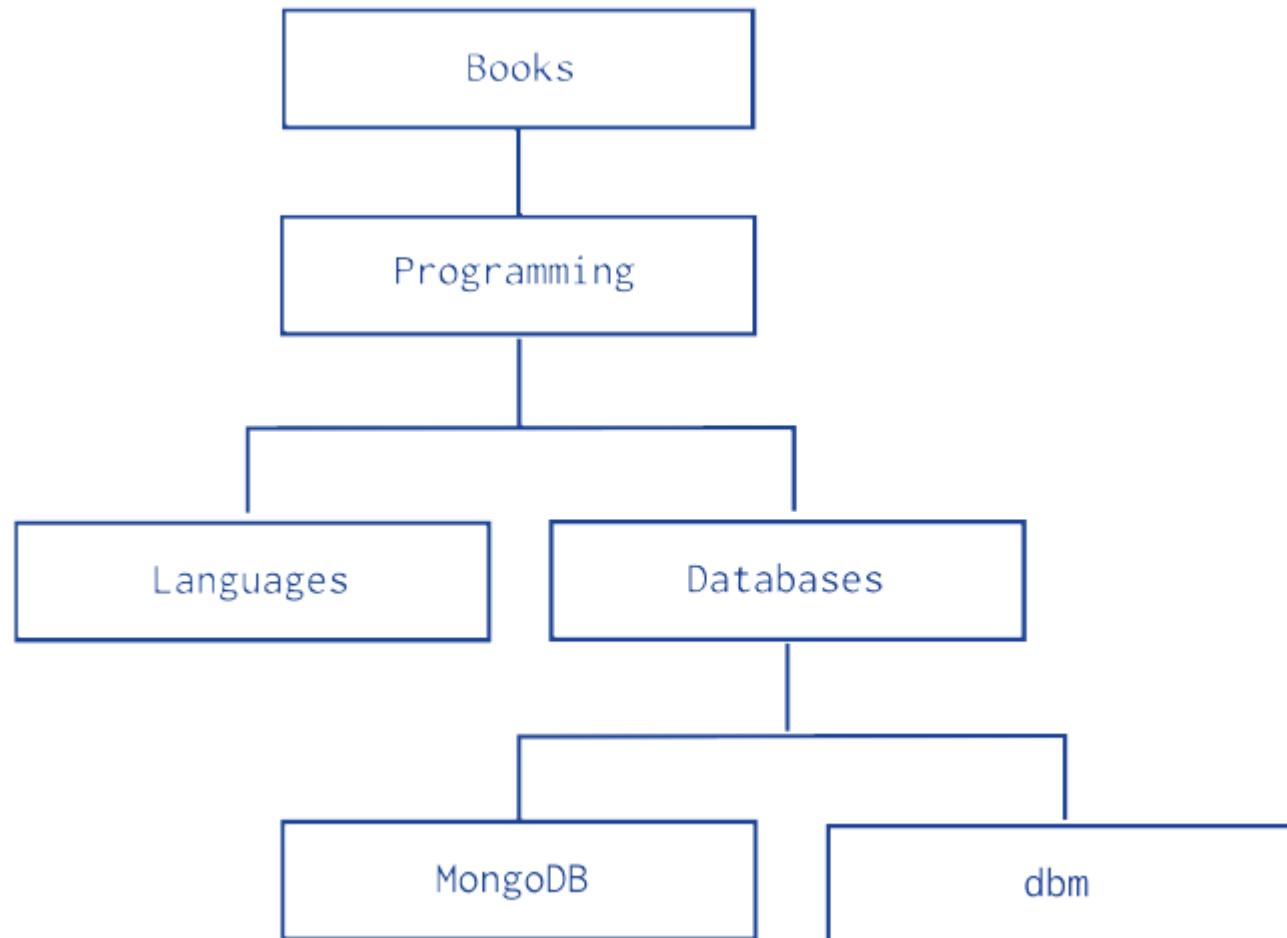
```
// Get all persons in the "mongoDB User" group  
db.person.find({"groups": "4e54ed9f48dc5922c0094a42"});
```

```
// Get all persons in the "mongoDB Administrator" group  
db.person.find({"groups": "4e54ed9f48dc5922c0094a41"});
```

```
// Get all groups for "Joe Mongo"  
db.groups.find({"persons": "4e54ed9f48dc5922c0094a43"});
```

```
// Get all groups for "Sally Mongo"  
db.groups.find({"persons": "4e54ed9f48dc5922c0094a40"});
```

Modèle hiérarchique



Modèle hiérarchique

Solution 1 : Lien Parent

```
db.categories.insert( { _id: "MongoDB", parent: "Databases" } )
db.categories.insert( { _id: "dbm", parent: "Databases" } )
db.categories.insert( { _id: "Databases", parent: "Programming" } )

db.categories.insert( { _id: "Languages", parent: "Programming" } )
db.categories.insert( { _id: "Programming", parent: "Books" } )
db.categories.insert( { _id: "Books", parent: null } )
```

Pour récupérer le parent

```
db.categories.findOne( { _id: "MongoDB" } ).parent
```

Pour récupérer les enfants

```
db.categories.find( { parent: "Databases" } )
```

Modèle hiérarchique

Solution II : Liste enfants

```
db.categories.insert( { _id: "MongoDB", children: [] } )
db.categories.insert( { _id: "dbm", children: [] } )
db.categories.insert( { _id: "Databases", children: [ "MongoDB", "dbm" ] } )
db.categories.insert( { _id: "Languages", children: [] } )
db.categories.insert( { _id: "Programming", children: [ "Databases", "Languages" ] } )
db.categories.insert( { _id: "Books", children: [ "Programming" ] } )
```

Pour récupérer le parent

```
db.categories.find( { children: "MongoDB" } )
```

Pour récupérer les enfants

```
db.categories.findOne( { _id: "Databases" } ).children
```

Modèle hiérarchique

Solution III Liste ancêtres

```
db.categories.insert( { _id: "MongoDB", ancestors: [ "Books", "Programming", "Databases" ], parent: "Databases" } )  
  
db.categories.insert( { _id: "dbm", ancestors: [ "Books", "Programming", "Databases" ], parent: "Databases" } )  
  
db.categories.insert( { _id: "Databases", ancestors: [ "Books", "Programming" ], parent: "Programming" } )  
  
db.categories.insert( { _id: "Languages", ancestors: [ "Books", "Programming" ], parent: "Programming" } )  
  
db.categories.insert( { _id: "Programming", ancestors: [ "Books" ], parent: "Books" } )  
  
db.categories.insert( { _id: "Books", ancestors: [ ], parent: null } )
```

Pour récupérer les ancêtres

```
db.categories.findOne( { _id: "MongoDB" } ).ancestors
```

Pour récupérer les descendants

```
db.categories.find( { ancestors: "Programming" } )
```

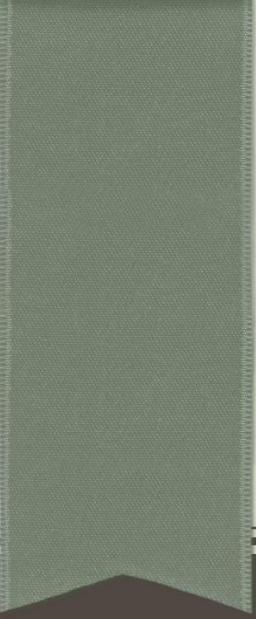
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Opérateurs



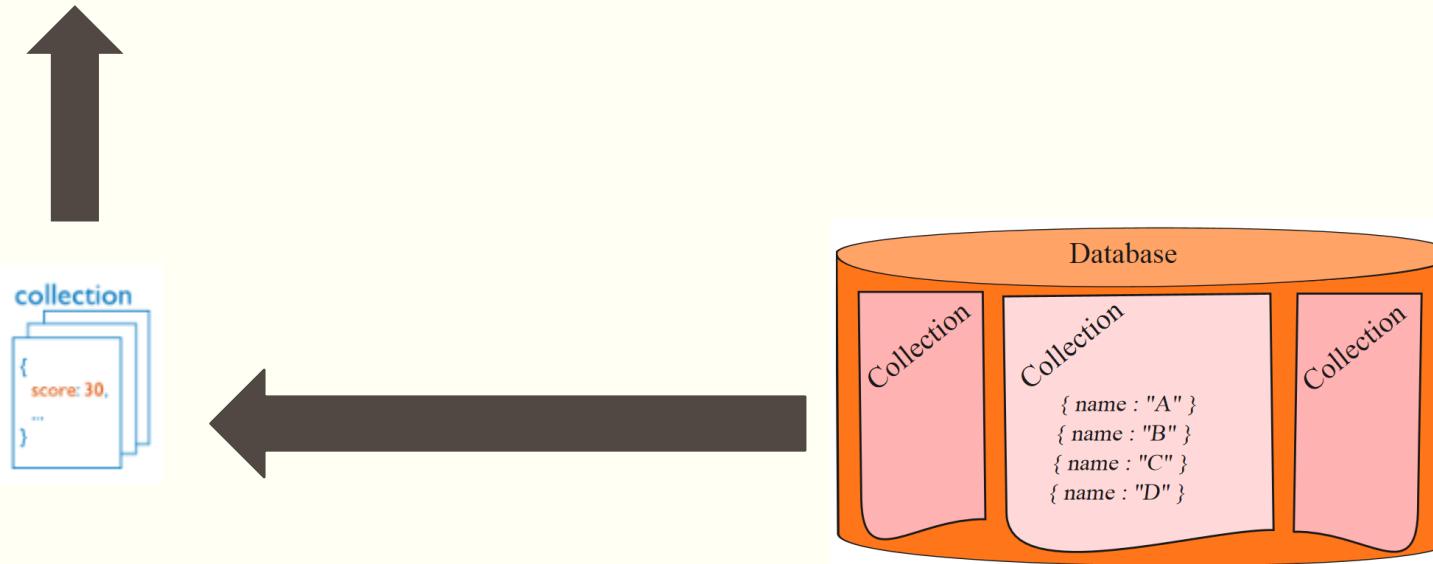
MODULE 5

Performance

Index

- Que fait la base de données MongoDB quand nous l'interrogeons?
 - MongoDB doit analyser chaque document
 - Processus inefficace lorsqu'on a un grand volume de données

```
db.users.find( { score: { "$lt" : 30} } )
```



Definition of Index

► **Definition:** Les index sont des structures de données particulières qui stockent une petite partie des données d'une collection dans un format simple et facile.

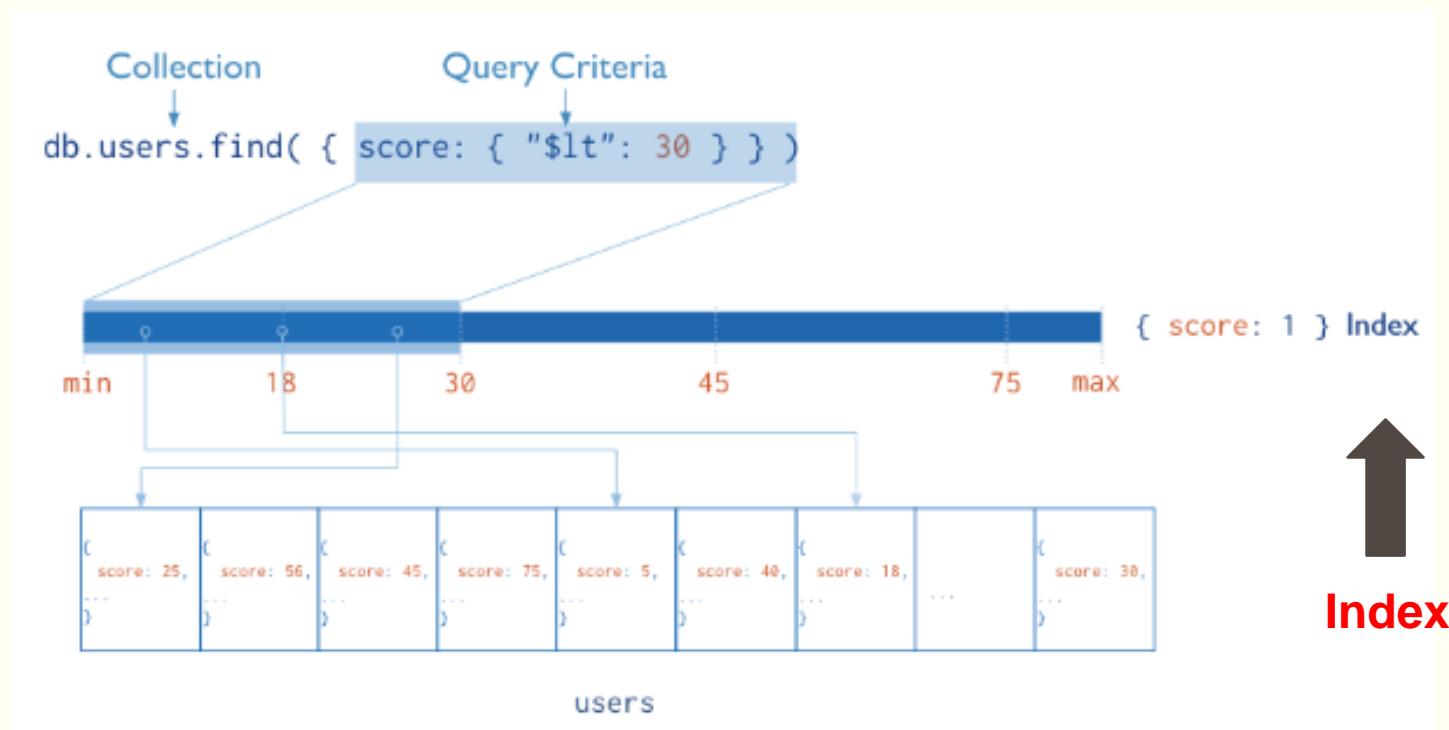


Diagram of a query that uses an index to select

Index in MongoDB

Operations

► Creation index

► `db.users.ensureIndex({ score: 1 })`

► Show existing indexes

► `db.users.getIndexes()`

► Drop index

► `db.users.dropIndex({score: 1})`

► Explain—Explain

► `db.users.find().explain()`

► Returns a document that describes the process and indexes

► Hint

► `db.users.find().hint({score: 1})`

► Overide MongoDB's default index selection

Index Types

- **Single Field Indexes (Index champ Unique)**
- **Compound Field Indexes(Index a champ composé)**
- **Multikey Indexes (Index à clef multiple)**

Création des Indexes

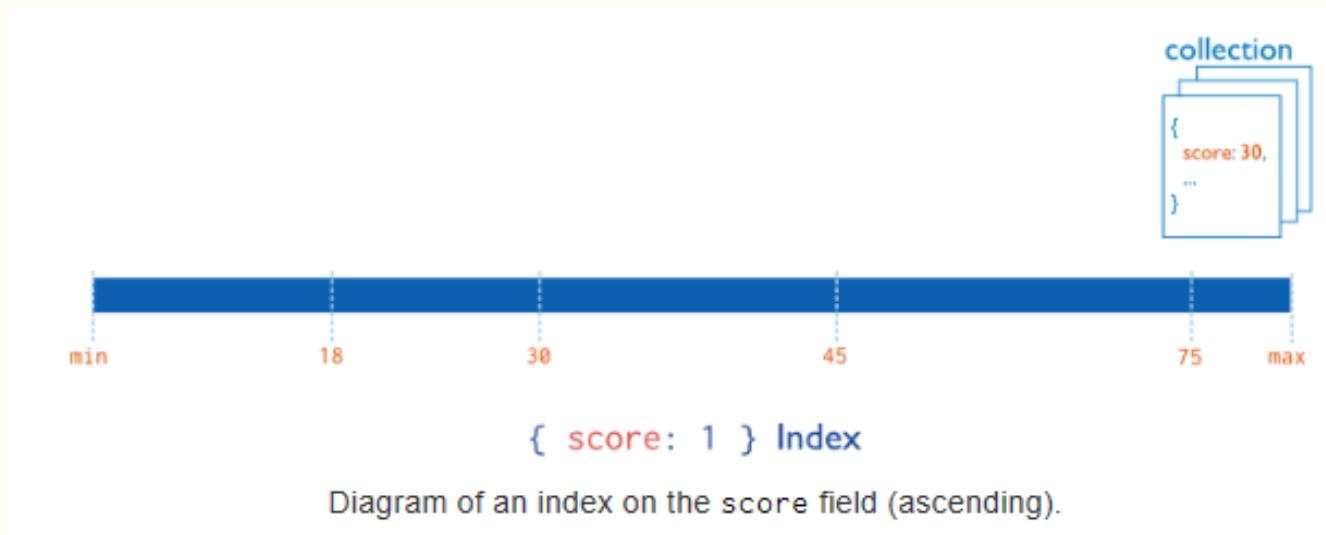
- ▶ **Single Field Index**
- ▶ **Compound Field Indexes**
- ▶ **Multikey Indexes**

```
db.zips.ensureIndex({pop: -1})
db.zips.ensureIndex({state: 1, city: 1})
db.zips.ensureIndex({loc: -1})
```

Single Field Indexes

▪ Single Field Indexes

- `db.users.ensureIndex({ score: 1 })`



Single Field Indexes

- `db.collection.ensureIndex({student_id:1})`
- `1`=ascending,
- `-1`=descending
 - important for sorting not so much for searching
- Sorting can also use a reverse index if the sort criteria are exactly the reverse of an (simple or compound) index

Compound Field Indexes

▪ Compound Field Indexes

- `db.users.ensureIndex({ userid:1, score: -1 })`

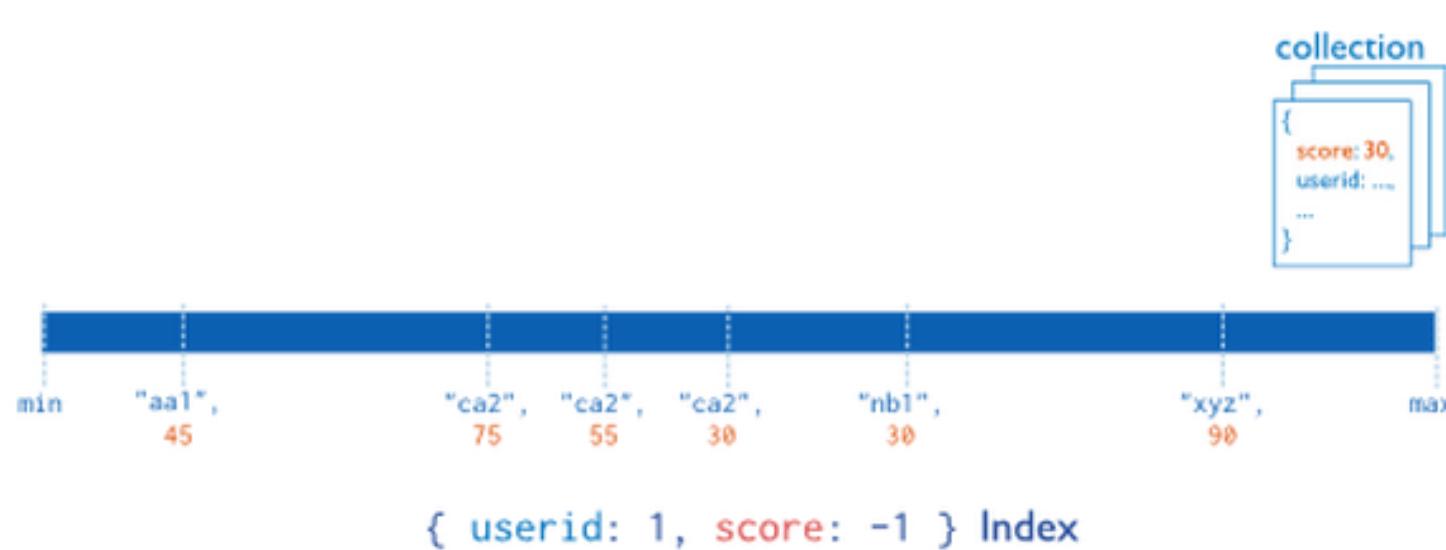


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

General rule: A Query where one term demands an exact match and another specifies a range requires a compound index where the range key comes second.

Index generality

Unique Index: db.collection.ensureIndex({student_id:1}, {unique: true})

// dropDups: true → dangerous

By default index creation is done in the foreground which is fast but blocking all other writers to the same DB.

Background index creation `{background: true}` will be slow but it will not block the writers

We want indexes to be in memory. Find out the index size: `db.col.stats()` or `db.col.totalIndexSize()`

`db.system.indexes.find()` → finds all indexes of the current db

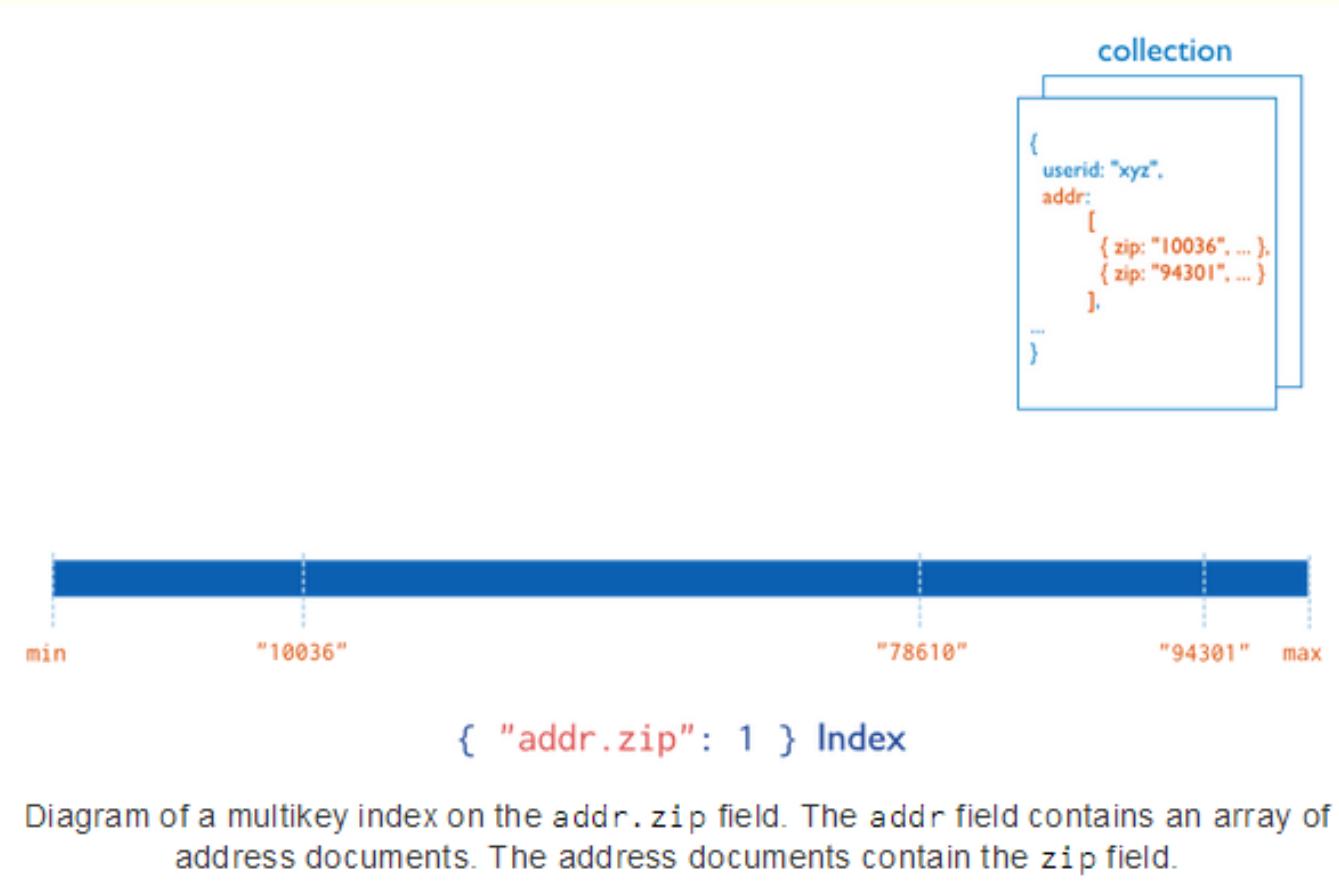
`db.collection.getIndexes()` → all indexes of collection

`db.collection.dropIndex({student_id:1})`

Multikey Indexes

■ Multikey Indexes

- db.users.ensureIndex({ addr.zip:1})



Multikey Indexes

- A multi key index is an index on an array field of a document, e.g. a student document has array of teacher-ids. One can add a multi key index on the teachers-array, which indexes all of the values in the array for all the documents.
- Multi key indices are one of the reason that linking works so well in MongoDB
- It is not possible to have a compound index with two array (multi key) fields

Sparse Index

- Missing index key in documents map to null
 - unique key not possible because multiple nulls are not allowed
- Sparse indexes only index documents that have a key set for the key being indexed `{unique: true, sparse: true}`
- On a sorted find the non-indexed documents will not be found when the sparse index is used for the sort

Explain & Hint

- `db.collection.find({...}).explain()`
- `db.collection.find({...}).explain(true)` //shows all possible plans
- `db.collection.find({...}).hint({a:1, b:1})` // use specified index
- `db.collection.find({...}).hint({$natural:1})` // use no index
- In Java:
 - `.find(query).hint("IndexName")` OR
 - `.find(query).hint(new BasicDocument(a, 1).append(b, 1))`

Explain vs sans index

```
> db.zips.dropIndexes()
{
  "nIndexesWas" : 4,
  "msg" : "non-_id indexes dropped for collection",
  "ok" : 1
}
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
  "cursor" : "BasicCursor",
  "isMultiKey" : false,
  "n" : 19,
  "nscannedObjects" : 29467,
  "nscanned" : 29467,
  "nscannedObjectsAllPlans" : 29467,
  "nscannedAllPlans" : 29467,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 33,
  "indexBounds" : {
  },
  "server" : "g:27017"
}
```

Explain vs avec index

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
  "cursor" : "BtreeCursor state_1_city_1",
  "isMultiKey" : false,
  "n" : 19,
  "nscannedObjects" : 19,
  "nscanned" : 19,
  "nscannedObjectsAllPlans" : 19,
  "nscannedAllPlans" : 19,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 0,
  "indexBounds" : {
    "state" : [
      [
        "TN",
        "TN"
      ]
    ],
    "city" : [
      [
        "NASHVILLE",
        "NASHVILLE"
      ]
    ]
  },
  "server" : "g:27017"
}
```

Efficiency of indexes

`$gt, $lt, $ne, $nin, $not($exists)` might be inefficient even if an index is used because still many index items (indexed documents) need to be scanned

may be a good idea to use a hint to use a diff. index

`$regex` can only use an index if it is stemmed on the left side, e.g. `/^abc/`

Profiling slow queries

- MongoDB logs slow queries (>100ms) by default into the logfile
- Use pofiler:
- `mongod --profile 1 --slowms 10` // logs all queries taking longer than 10ms to `system.profile` collection
- Levels: 0=off (default) 1=log slow queries 2=log all queries (general debugging feature for dev.)
- Mongo shell: `db.getProfilingLevel()` `db.getProfilingStatus()` `db.setProfilingLevel(level, slowms)`
- `mongod --notablescan` option: Set `notablescan = true` on your dev or test machine to find operations that require a table scan

mongotop & mongostat

- **mongotop 3** // runs every 3 seconds showing you in which collection how much time (read, write, total) is spent
 - **mongostat** // shows inserts, queries, updates, deletes, ... per second
- idx miss % = index which could not be accessed in memory

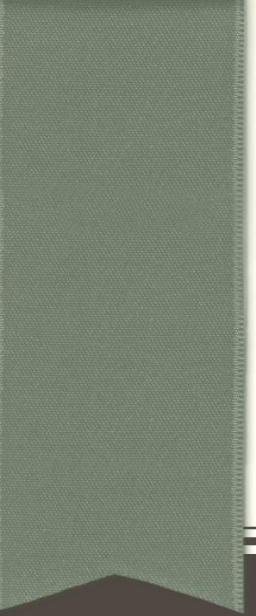
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Opérateurs



MODULE 6

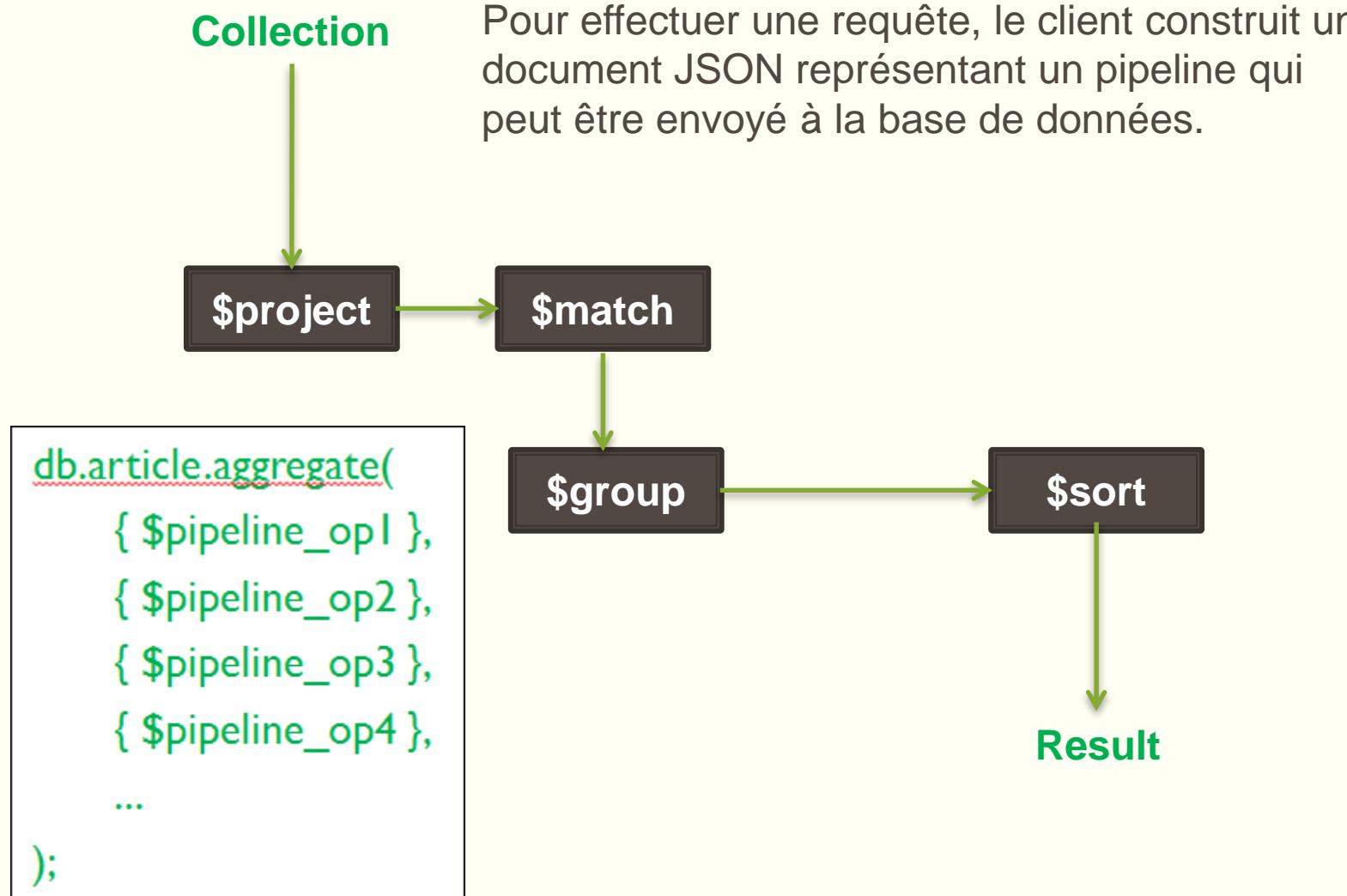
Agrégation Framework

Framework d'agrégation

Le framework d'agrégation étend les capacités de requêtes et de traitement des données.

Avec les nouveaux opérateurs, les développeurs pourront trier et agréger par groupes les données sur lesquelles portent les requêtes et leur appliquer diverses opérations.

Pipeline



Exemple 1

Collection
↓
db.orders.aggregate([
 \$match phase → { \$match: { status: "A" } },
 \$group phase → { \$group: { _id: "\$cust_id", total: { \$sum: "\$amount" } } }
]

```
{  
  cust_id: "A123",  
  amount: 500,  
  status: "A"  
}  
  
{  
  cust_id: "A123",  
  amount: 250,  
  status: "A"  
}  
  
{  
  cust_id: "B212",  
  amount: 200,  
  status: "A"  
}  
  
{  
  cust_id: "A123",  
  amount: 300,  
  status: "D"  
}
```

\$match

```
{  
  cust_id: "A123",  
  amount: 500,  
  status: "A"  
}  
  
{  
  cust_id: "A123",  
  amount: 250,  
  status: "A"  
}  
  
{  
  cust_id: "B212",  
  amount: 200,  
  status: "A"  
}
```

\$group

Results

```
{  
  _id: "A123",  
  total: 750  
}  
  
{  
  _id: "B212",  
  total: 200  
}
```

orders

Exemple II

```
db.zips.aggregate([
  {$match: {state: {$in: ["CA", "NY"]}}},
  {$group: {_id: {city: "$city", state: "$state"}, pop: {$sum: "$pop"}}},
  {$match: {pop: {$gt: 25000}}},
  {$group: {_id: null, avg: {$avg: "$pop"}}}
])
```

Pipeline Opérations

- **\$match** : requête, équivalent du find
- **\$project** : met en forme les résultats, notamment en enlevant/ajoutant des champs
- **\$unwind** : permet de faire du streaming de tableau, chaque élément du tableau sera traité comme un document
- **\$group** : agrège les données
- **\$sort** : fonctionnalités de trie
- **\$limit** : limite le nombre de documents renvoyés
- **\$skip** : exclue certains documents du résultat

Cardinalité

Opérateur	Cardinalité
<code>\$match</code>	N : 1
<code>\$project</code>	1 : 1
<code>\$unwind</code>	1 : N
<code>\$group</code>	N : 1
<code>\$sort</code>	1 : 1
<code>\$limit</code>	N : 1
<code>\$skip</code>	N : 1

\$match

- Utilisation équivalente au `find({...})`

```
{  
  "_id" : "35004",  
  "city" : "ACMAR",  
  "loc" : [  
    -86.51557,  
    33.584132  
,  
  "pop" : 6055,  
  "state" : "AL"  
}
```

`db.zips.aggregate([{$match: { state:"NY" } }])`

`db.zips.aggregate([{$match: { pop: { $gt:1000} } }])`

\$sort, \$limit, \$skip

- **\$sort** : Tri les documents

- Memory intensive; Can't use index (at least after grouping)

```
db.zips.aggregate([ {$match: { state:"NY" } }, {$sort: { population:-1 } } ])
```

- **\$limit** : limite le nombre de documents

- **\$skip** : exclue certains documents du résultat

- Makes only sense when you do a sort first
 - First skip – then limit (order of the stages in the pipeline matter)

```
db.zips.aggregate([  
  {$match: { state:"NY" } },  
  {$sort: { population:-1 } },  
  {$skip: 10},  
  {$limit: 5}  
])
```

\$project

\$project : permet de remanier un document en ajoutant, supprimer, renommer ... des champs.

```
db.products.aggregate([
  {$project:
    {
      title:1, /* champs à inclure, s'il existe */
      author:1, /* champs à inclure, s'il existe */
      'details.tag':1
    }
  }
])
```

```
db.products.aggregate([
  {$project:
    {
      title:0, /* champs à exclure */
      author:0 /* champs à exclure */
    }
  }
])
```

\$project

- Remove keys - If you don't mention a key, it is not included, except for `_id`, which must be explicitly suppressed `{$project: {_id: 0, ...}}`
- Add keys (also possible to create new subdocuments)
- Keep keys: `{$project: {myKey: 1, ...}}`
- Rename keys / Use functions: `$toUpper`, `$toLower`, `$add`, `$multiply`

\$project

Autre Exemple

```
{  
  "_id" : 1,  
  "name" : "iPad 16GB Wifi",  
  "manufacturer" : "Apple",  
  "category" : "Tablets",  
  "price" : 499  
}
```

```
db.products.aggregate([  
  {$project:  
    {  
      _id:0,  
      'maker': {$toLower:'$manufacturer'},  
      'details': {'category': '$category',  
                  'price' : {"$multiply": ["$price",10]}  
                },  
      'item':'$name'  
    }  
  }  
])
```

\$unwind

```
{ a:1, b:2, c:['apple','pear', 'orange']}
```



```
{ a:1, b:2, c:'apple'}
```

```
{ a:1, b:2, c:'pear'}
```

```
{ a:1, b:2, c:'orange'}
```

\$group

```
> db.products.aggregate([
...   {$group:
...     {
...       _id: { "manufacturer": "$manufacturer", "category" : "$category"},
...       num_products:{$sum:1}
...     }
...   }
... ])
```

Opération sur les groupes

- **\$sum**: Add one to a key
 - mySum: {\$sum:1} pour compter le nombre d'éléments
 - sum_prices:{\$sum:"\$price"} : somme de variables
- **\$avg, \$min, \$max**: Average, Minimum or maximum value of a key
- Create arrays: **\$push, \$addToSet**
 - categories: {\$addToSet: "\$category"}
- Only useful after a sort: **\$first, \$last**
 - {\$group:{_id:"\$_id.state", population:{\$first:"\$population"}}}

Limitations of the aggregation framework

- A result document can only be 16GB
- One can only use up to 10% of the memory on a machine
- In sharded environment: After first `$group` / `$sort` the next phase have to be performed on the mongos router
- Alternative to aggregation framework: map-reduce

MongoDB

SQL Statement	Mongo Statement
<pre>SELECT COUNT(*) FROM users</pre>	<pre>db.users.aggregate([{ \$group: { _id:null, count:{\$sum:1}} }])</pre>
<pre>SELECT SUM(price) FROM orders</pre>	<pre>db.users.aggregate([{ \$group: { _id:null, total:{\$sum:"\$price"} } }])</pre>
<pre>SELECT cust_id,SUM(price) FROM orders GROUP BY cust_id</pre>	<pre>db.users.aggregate([{ \$group: { _id:"\$cust_id",total:{\$sum:"\$price"} } }])</pre>
<pre>SELECT cust_id,SUM(price) FROM orders WHERE active=true GROUP BY cust_id</pre>	<pre>db.users.aggregate([{ \$match:{active:true} }, { \$group:{_id:"\$cust_id",total:{\$sum:"\$price"} } }])</pre>

SQL Comparison:

<http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/>

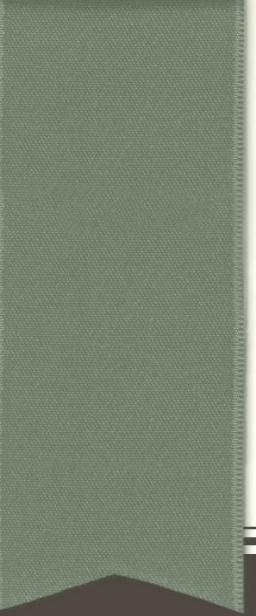
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

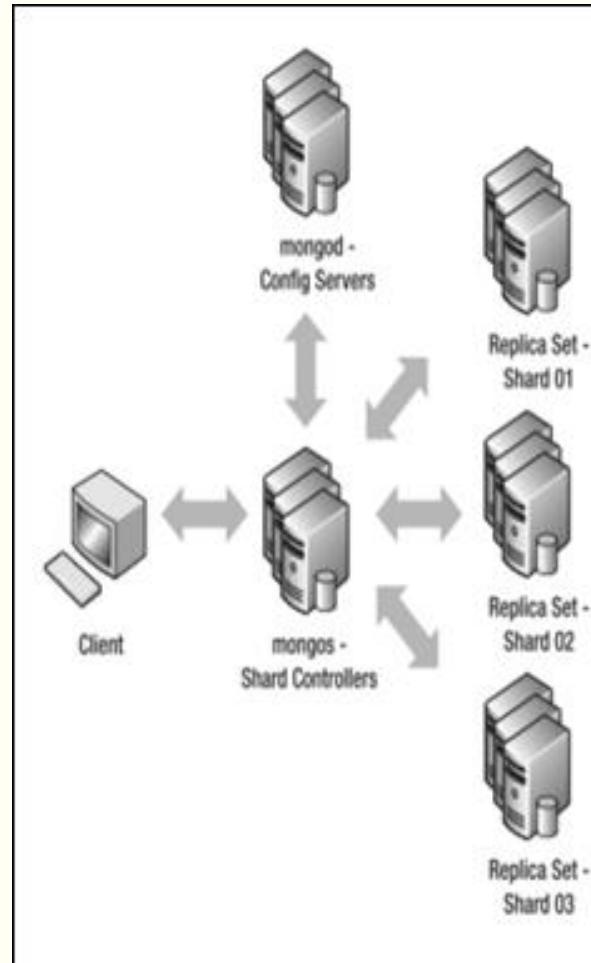
Agrégation



MODULE 7

Administration

Replication & Sharding



MongoDB réPLICATION

- Qu'est ce qu'est une réPLICATION?
- But de la replication/redondance
- Tolerance de panne
 - Disponibilité
 - augmenter la capacité de lecture

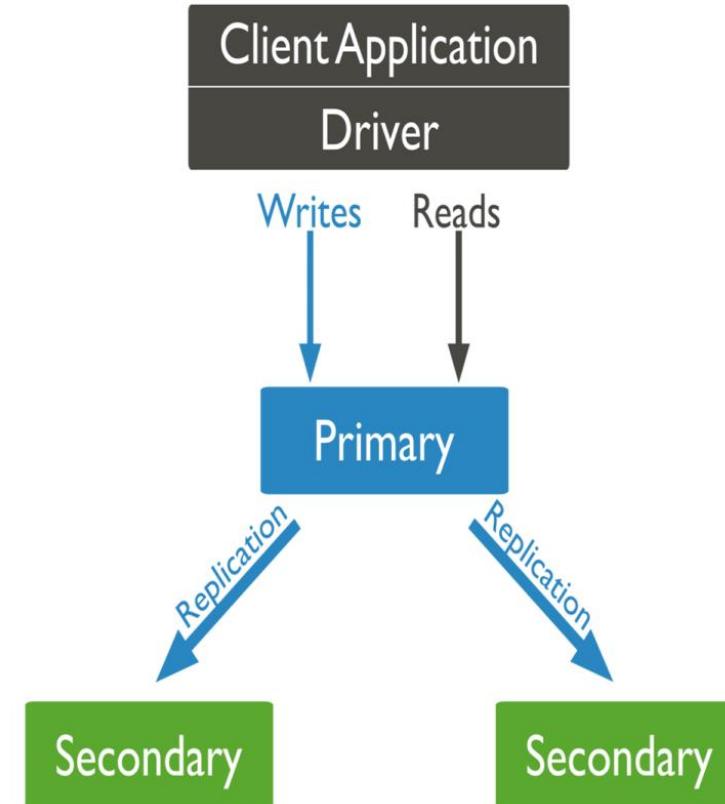


Figure 1: Diagram of default routing of reads and writes to the primary.

RéPLICATION

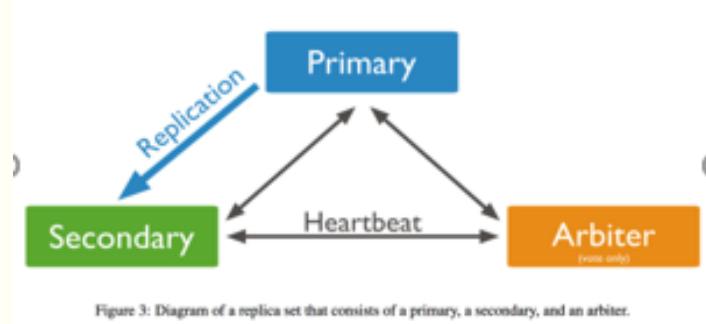
- Qu'apporte la réPLICATION ?
 - Redondance
 - Simplification de tâches (backups, ...)
 - Augmentation de la capacité de lecture
- Un replica set est un cluster d'instances MongoDB.
- Stratégie maître / esclaves
- Il doit TOUJOURS y avoir un unique maître.
- Les clients effectuent les écritures sur l'instance ... ?

RéPLICATION

- La réPLICATION du maître vers les esclaves est asynchrone.
- Quels sont les avantages et inconvénients ?
 - Synchrone : Bloquant / Coûteux / Forte cohérence
 - Asynchrone : Non bloquant / Rafraîchissement des données obligatoires.

Replica Set Members

- ▶ Primary
 - ▶ Read, Write operations
- ▶ Secondary
 - ▶ Asynchronous Replication
 - ▶ Can be primary
- ▶ Arbiter
 - ▶ Voting
 - ▶ Can't be primary
- ▶ Delayed Secondary
 - ▶ Can't be primary



Arbiter nodes exist for voting purposes, e.g. if you have an even number of regular (= primary & secondary) nodes it can ensure a majority in an election

Allows you to have only two regular nodes.

Delayed nodes are disaster recovery nodes: Can be set to be whatever time behind the regular nodes. Can't participate in elections

Hidden node (e.g. for reporting) can't become the primary but can participate in elections

RéPLICATION

- Tolérance aux pannes
- Un replica set est tolérant aux pannes.
- Si le nœud primaire tombe, les nœuds secondaires peuvent élire un nouveau nœud primaire.
- Comment rendre l'élection automatique ?

RéPLICATION

- Comment rendre l'élection automatique ?
 - Détection de la mort du noeud primaire (ping / heartbeat)
 - Lancement d'une élection
 - Le nœud ayant reçu une majorité de vote devient le nœud primaire, grâce à une priorité .

Replication dans MongoDB

- Automatic Failover(prise en charge automatique par une autre machine)
 - Heartbeats
 - Elections
- The Standard Replica Set Deployment
- Déployer un nombre impair de membres Rollback
- Replication is done via a capped collection called oplog.rs in the “local” database
- Secondaries ask the primary for any items since a certain timestamp

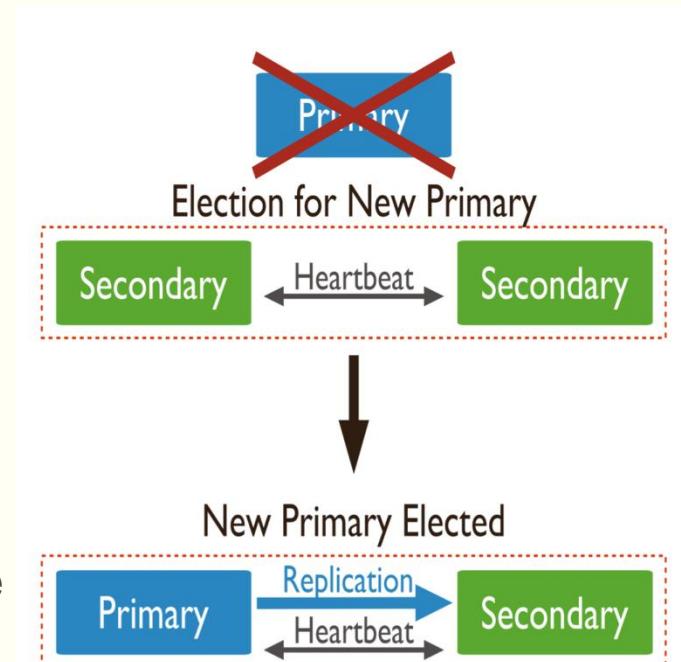


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

RéPLICATION - CONSISTANCE

- Que se passe t'il si un nœud primaire accepte une écriture et tombe en panne avant la réPLICATION de l'écriture ?
- On perd la donnée, et le replica set devient inconsistent.
- Comment corriger cela ?

RéPLICATION - CONSISTANCE

- Idée inspirée des SGBDR : Le rollback
- Le noeud primaire écrit en local les opérations demandées lorsqu'il accepte une écriture.
- Lors de son retour, soit il relance les opérations, soit il les annule (les rollback).

RéPLICATION - Préoccupations

- Lors de la mise en place d'un replica set , deux paramètres sont à prendre en compte
 - **Write Concern** : Message envoyé pour vérifier la validité d'une opération.
 - **Read Preferences** : Favoriser les lectures sur les nœuds secondaires.

RéPLICATION - Write Concern

- Qualité de chaque opération d'écriture et décrit le montant de préoccupation d'une application pour l'écriture.
- Plus la préoccupation augmente, plus les performances augmentent, plus la cohérence diminue.

RéPLICATION - Type de Write Concern

- **Erreurs ignorés** : Opérations non acquittées. Pas de notification d'erreurs (réseau, ...)
- **Sans acquittement** : Opérations non acquittées. Au courant des erreurs réseaux.
- **Acquittement** : Opérations acquittées. Ne résiste pas au failover.
- **Journalisé** : Opérations valides si acquittées et écrites dans le journal.
- **Acquittement du réplica** : Tous les noeuds secondaires acquittent les opérations.

RéPLICATION - Read Preferences

- Par défaut, les opérations de lecture sont envoyées au nœud primaire.
- Les lectures sur le nœud primaire garantissent d'obtenir toujours les données les plus fraîches.
- Les lectures sur les nœuds secondaires améliorent le débit de lecture en distribuant les lectures.

RéPLICATION - Read Preferences

- Penser à modifier cela lorsque :
 - Opérations n'affectant pas le front-end (backup, reporting, ...).
 - Application distribuée géographiquement. On envoie le client sur le noeud secondaire le plus proche.

RéPLICATION - Types de Read

- Les différents types de read preferences sont :
 - **primary** : Toujours utiliser le nœud primaire. Exception si pas de nœud primaire.
 - **primaryPreferred** : Toujours utiliser le nœud primaire. On utilise les nœuds secondaires si pas de nœud primaire.
 - **secondary** : Toujours les nœuds secondaires. Exception si pas de nœuds secondaires.
 - **secondaryPreferred** : Toujours les nœuds secondaires. On utilise le nœud primaire si pas de nœuds secondaires.
 - **nearest** : On prend le nœud le plus proche, selon le choix fait par l'utilisateur

Replication Configuration

- Start a replication set: mongod -repSet m101 --logpath "1.log" --dbpath /data/rs1 -fork
 - Register replica set nodes in the mongo shell:

```
config = { _id: "m101", members:[ { _id : 0, host : "localhost:27017", priority:0, slaveDelay:5}, { _id : 1, host : "localhost:27018"}, { _id : 2, host : "localhost:27019"} ] }
```
 - config = { _id: "m101", members:[{ _id : 0, host : "localhost:27017", priority:0, slaveDelay:5}, { _id : 1, host : "localhost:27018"}, { _id : 2, host : "localhost:27019"}] }

Replication Configuration

- **rs.initiate(config)** ← Initializes the replica set // can't be executed on a node which can't become primary
- **rs.status()** ← Gives you the status information about the replica set
- **rs.isMaster()** ← Tells you if you are the primary node
- **rs.slaveOk()** ← If issued on a secondary node it allows you to read from this secondary node
- **rs.stepDown()** ← Forces primary node to step down as a primary node

Failover and Rollback

Scenario:

- When the primary dies a secondary which becomes elected as a new primary which does not have the latest entries from the old primaries oplog
- When the former primary node comes back up as a secondary node it will request the oplog data from the new primary and roll back the writes the current primary does not have and write them to a rollback file which can be applied manually
- If the oplog of the new primary has looped during the time the old primary was down the entire dataset will be copied from the new primary
- The risk of losing data due to a rollback can be avoided by waiting till the majority of the nodes have the data → set the write concern

Connecting from the Java Driver

Provide a **seed list** to the MongoClient instance

```
new MongoClient(Arrays.asList(  
    new ServerAddress("localhost", 27017),  
    new ServerAddress("localhost", 27018),  
    new ServerAddress("localhost", 27019),  
));
```

➔ Will work even if the primary is not part of the seed list. The Java Client starts a background thread which pings all nodes from the seed list and all discovered nodes to find out which one is the primary

Read Preferences

- If you read from secondaries you might get stale reads. This might be OK if different applications do the writes and the reads
- For any read preference (except Primary) the driver will look at ping times and will only send reads to nodes which are within the latency window (15ms) of the fastest
- `db.collection.setReadPreference(ReadPreference.primaryPreferred())`
- `collection.find().setReadPreference(ReadPreference.nearest());`

Disposition de titre et de contenu avec liste

Question ?

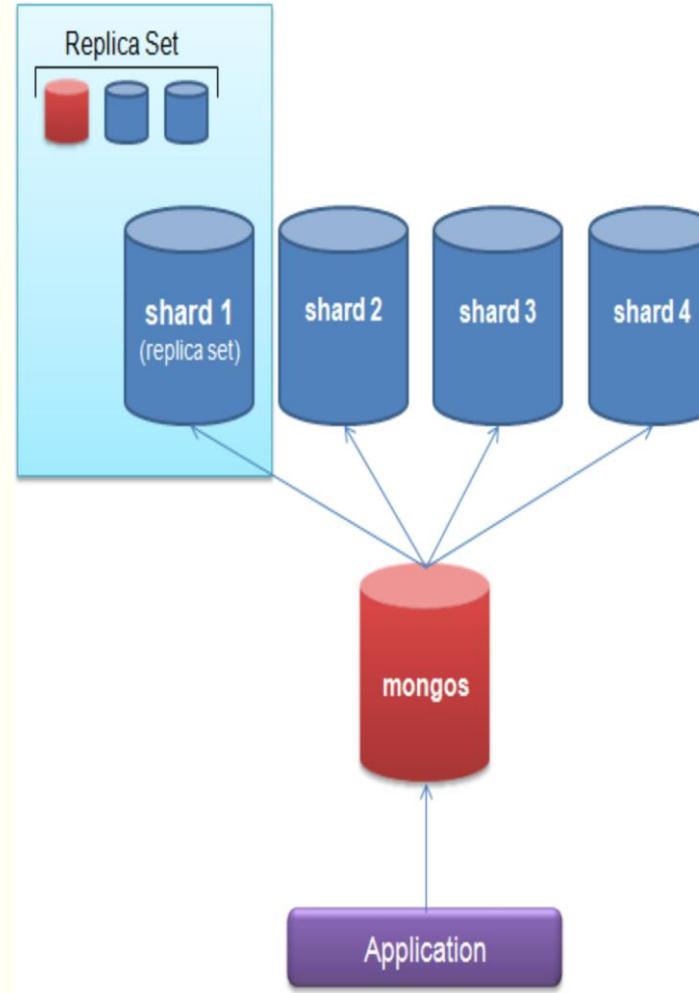
Disposition de titre et de contenu avec liste

TP

ReplicatSet

Sharding

- Qu'est ce que c'est ?
- But du sharding
 - Horizontal scaling out
- Query Routers
 - mongos
- Shard keys
 - Range based sharding
 - Cardinality
 - Avoid hotspotting

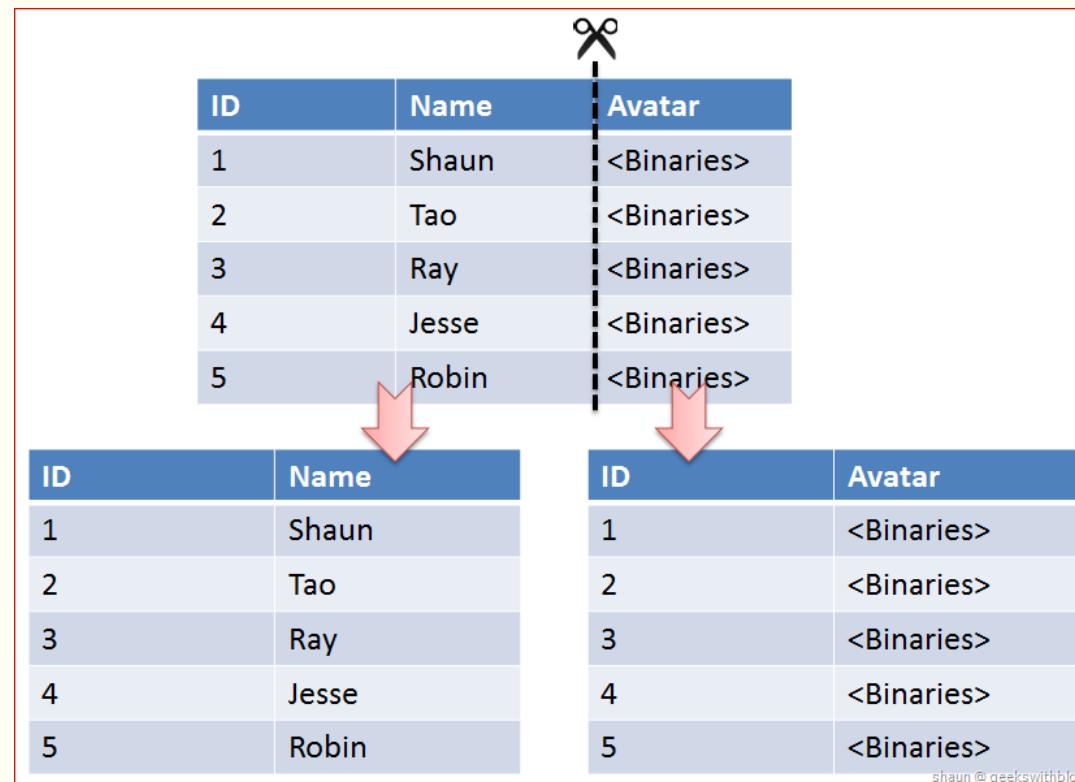


Sharding : Passage à l'échelle

- Ensemble de techniques qui permet de répartir les données sur plusieurs machines pour assurer la scalabilité de l'architecture.
- Il existe de fait 2 façons de partitionner (« sharder ») la donnée :
 - Verticalement
 - ou horizontalement.

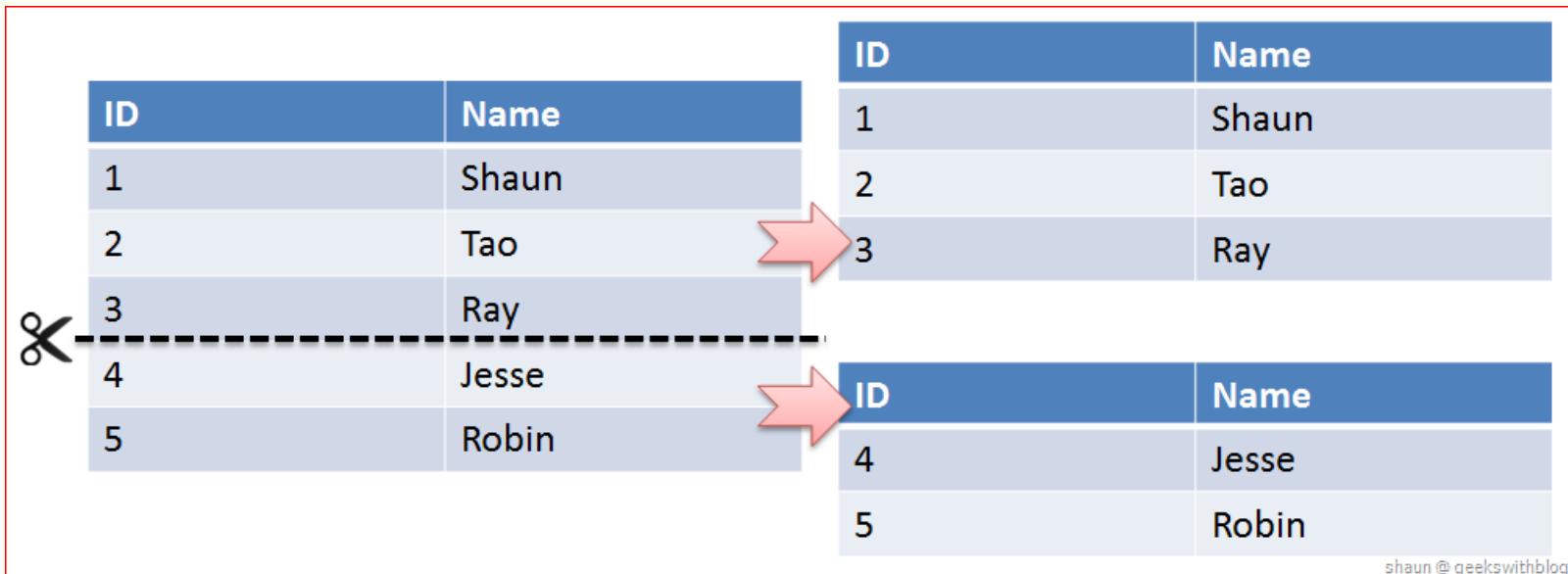
Le partitionnement vertical

- Le partitionnement vertical est le plus communément utilisé : il s'agit d'isoler, de séparer des concepts métier.
- Par exemple, on décidera de stocker les clients sur une base et leur contrat sur une autre.



Le partitionnement horizontal

La notion de partitionnement horizontal renvoie à l'idée de répartir l'ensemble des enregistrements d'une table (au sens d'une base de données) sur plusieurs machines.



Ainsi, on choisira par exemple de stocker

- les clients de A à M sur une machine #1
- et les clients de N à Z sur une autre machine #2.

Le sharding horizontal nécessite une clé de répartition – la première lettre du nom dans l'exemple.

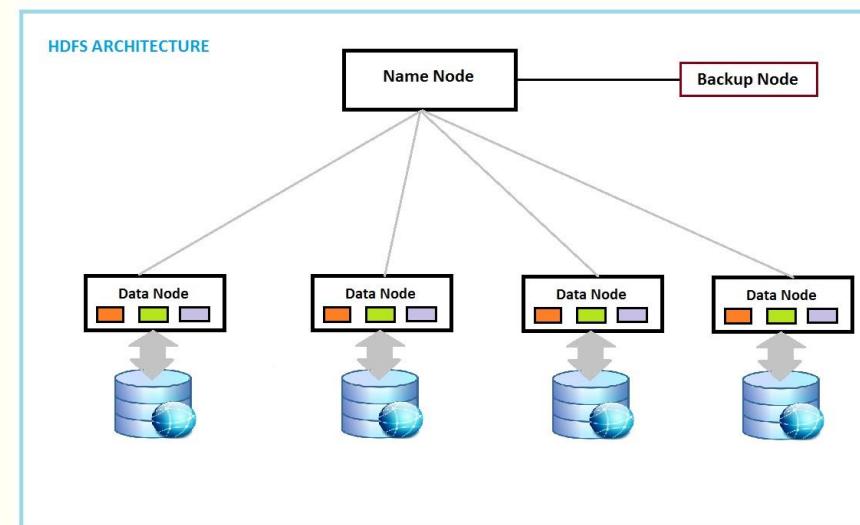
Les techniques liées au sharding

- Les grands du web ont, sur la base de ce choix de scalabilité horizontale, développé des solutions spécifiques (NoSQL) répondant à ces enjeux et ayant les caractéristiques suivantes :
 - une implémentation à partir de machines de grande série
 - Basée sur l'allocation de ressources : HDFS
 - une distribution (sharding) des données gérées au niveau du logiciel
 - Basée sur une structure arborescente : Index non-dense
 - Basée sur le hachage : Hachage Cohérent

Sharding : Allocation de ressources

HDFS1

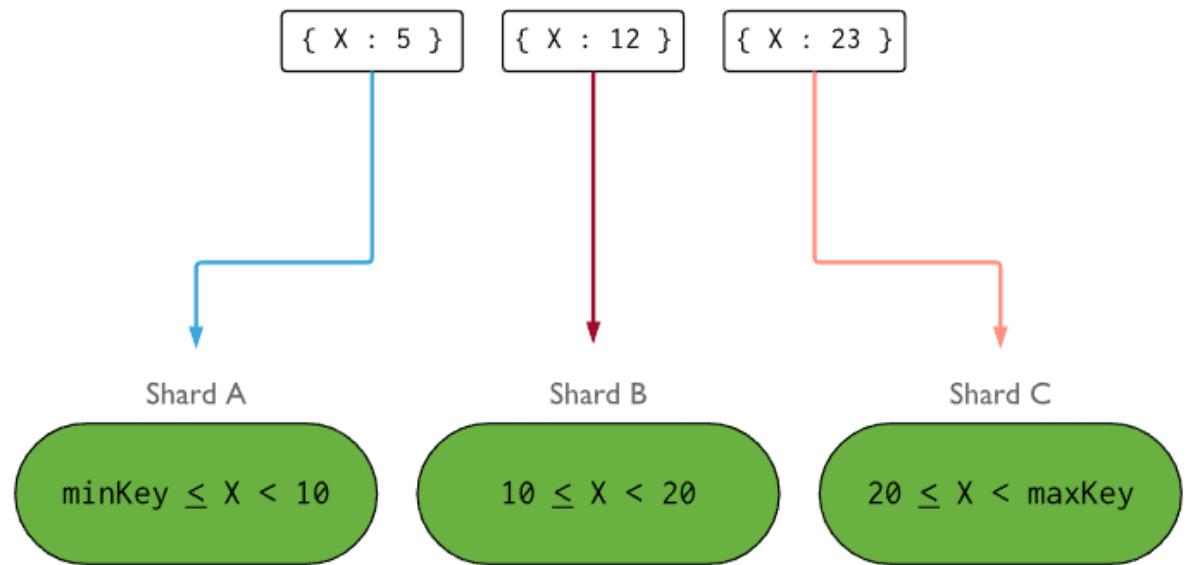
- Système de fichier distribué
 - Dépend de la charge des serveurs
 - Distribution, tolérance aux pannes
 - Allocation dynamique et optimisée
- Ex: BigTable (Google), Hadoop, Spark, Flink...



Sharding : Arborescent

Index non-dense distribué

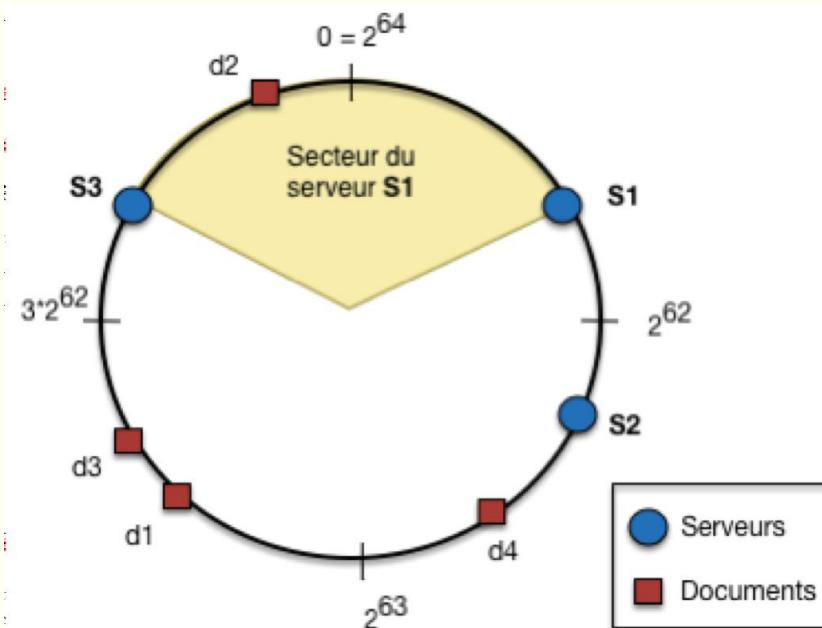
- Données triées physiquement
 - Découpées en blocs sur les noeuds (souvent 256Mo)
- Distribution, tolérance
 - Requêtes par intervalles
 - **Bien choisir la clé pour le tri**
- Ex : MongoDB, Hbase, CouchDB, CouchBase, Jackrabbit



Sharding : Consistent Hashing

Hachage cohérent (DHT3)

- Technique de hachage unique et dynamique pour les données et les serveurs
- Distribution en anneau (virtuel)
- Ex: DynamoDB, Cassandra, Redis, Voldemort, MongoDB (hash)...



Sharding dans MongoDB

- Enables horizontal scalability
- Shards are typically itself replica sets
- mongos is the sharding router which distributes data to the individual shards
- The application (and also the mongo shell) connects to mongos instead of mongod
- There can be multiple mongos - mongos typically runs on the same server as the application

Sharding

- If a mongos goes down the application will connect to a different one – similar to replica sets
- Shard key determines to which shard a document goes
- Sharding is at database level – but you can define if you want to shard or not shard a specific collection
- Config servers (which are mongod) keep track of where the shards are – in production you typically use 3 of them

Building a sharded environment

Set up two shards each a replica set of three mongod nodes

- Set up shard as a replication set:
 - mongod --replSet s0 --logpath "s0-r0.log" --dbpath /data/shard0/rs0 --port 37017 --fork --shardsvr
- Set up config server:
 - mongod --logpath "cfg-a.log" --dbpath /data/config/config-a --port 57040 --fork --configsvr
- Set up mongos router with information about the config servers:
 - mongos --logpath "mongos-1.log" --configdb localhost:57040,localhost:57041,localhost:57042 --fork
 - mongos now listens on the default mongod port

Building a sharded environment

- On the mongo shell – tell the config servers (via the mongos) about the shards:
- `db.adminCommand({ addshard : "s0/"+"localhost:37017" });`
- ... add further shards
- `db.adminCommand({enableSharding: "test"})`
 - enable sharding on test DB
- `db.adminCommand({shardCollection: "test.grades", key: {student_id:1}});`
 - shard collection “grades” with the shard key “student_id”
- `sh.help()`
 - Will display all the shard commands available in the mongo shell, e.g. `sh.status()`

Sharding implications

- Needs an index on first element of the shard key (can be compound but not multi-key)
- Each document needs the shard key
- The shard key is immutable
- On an update you need to specify the shard key or specify multi: true
- A find with no shard key will go to all shards
- The key used in most queries should be the shard key
- You can't have a unique index unless it is part of/starts with the shard key
- Write concerns are still important in a sharded setup

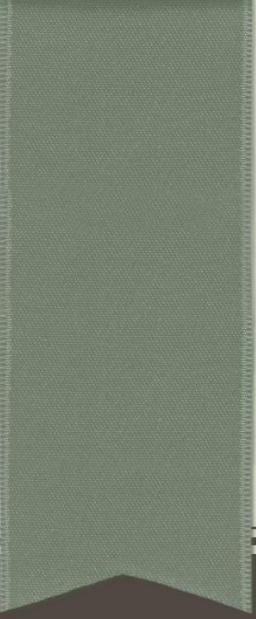
Sharding key

- Sufficient cardinality (variety of values)
- Avoid monotonically increasing keys to avoid hotspotting in writing (e.g. order_id, order_date)
- Compound sharding key is possible

Disposition de titre et de contenu avec liste

TP

Sharding



MODULE 8

Java et MongoDB

Java Driver

MongoDB offre un bon support et une intégration avec de nombreux langages.

Le driver Java propose une API pour:

- réaliser la transformation des objets en documents,
- la connexion à la base,
- les CRUD,
- les requêtes,
- etc.

Java - Connexion à la base MongoDB

```
package com.formation;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class InsertMongo {
    private final static String HOST = "localhost";
    private final static int PORT = 27017;

    public static void main(String args[]) {
        try {
            MongoClient mongoClient = new MongoClient(HOST, PORT);
            // Now connect to the test database
            MongoDatabase db = mongoClient.getDatabase("test");
            System.out.println("Connect to database successfully ");

        } catch (Exception e) {
            System.err.println(e.getClass().getName() + ": " + e.getMessage());
        }
    }
}
```

CRUD-Insertion

```
{  
  "name" : "john",  
  "age" : 25,  
  "phone": "321-654-987"  
}
```

```
public static void main(String args[]) {  
    try {  
        MongoClient mongoClient = new MongoClient(HOST, PORT);  
        // Now connect to the test database  
        MongoDB db = mongoClient.getDatabase("test");  
        System.out.println("Connect to database successfully ");  
  
        MongoCollection<Document> coll = db.getCollection("users");  
        Document doc = new Document("name", "john").append("age", 25).append("phone", "321-654-987");  
        coll.insertOne(doc);  
  
    } catch (Exception e) {  
        System.err.println(e.getClass().getName() + ": " + e.getMessage());  
    }  
}
```

CRUD-Read

```
// Retrouver un ensemble de documents
DBCursor cursor = coll.find();
int i = 1;
while (cursor.hasNext()) {
    System.out.println("Inserted Document: " + i);
    System.out.println(cursor.next());
    i++;
}
```

CRUD-Update

```
//éléments à maj
DBObject upDocument = new BasicDBObject();
upDocument.put("count", 2);

// criteres de selection des documents à maj
DBObject criteriaDocument = new BasicDBObject();
criteriaDocument.put("name", "MongoDB");

// update un seul document par défaut
coll.update(criteriaDocument, upDocument);
```

CRUD-Delete

```
DBObject myDoc = coll.findOne();
coll.remove(myDoc);
```