



# MONGODB



# 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

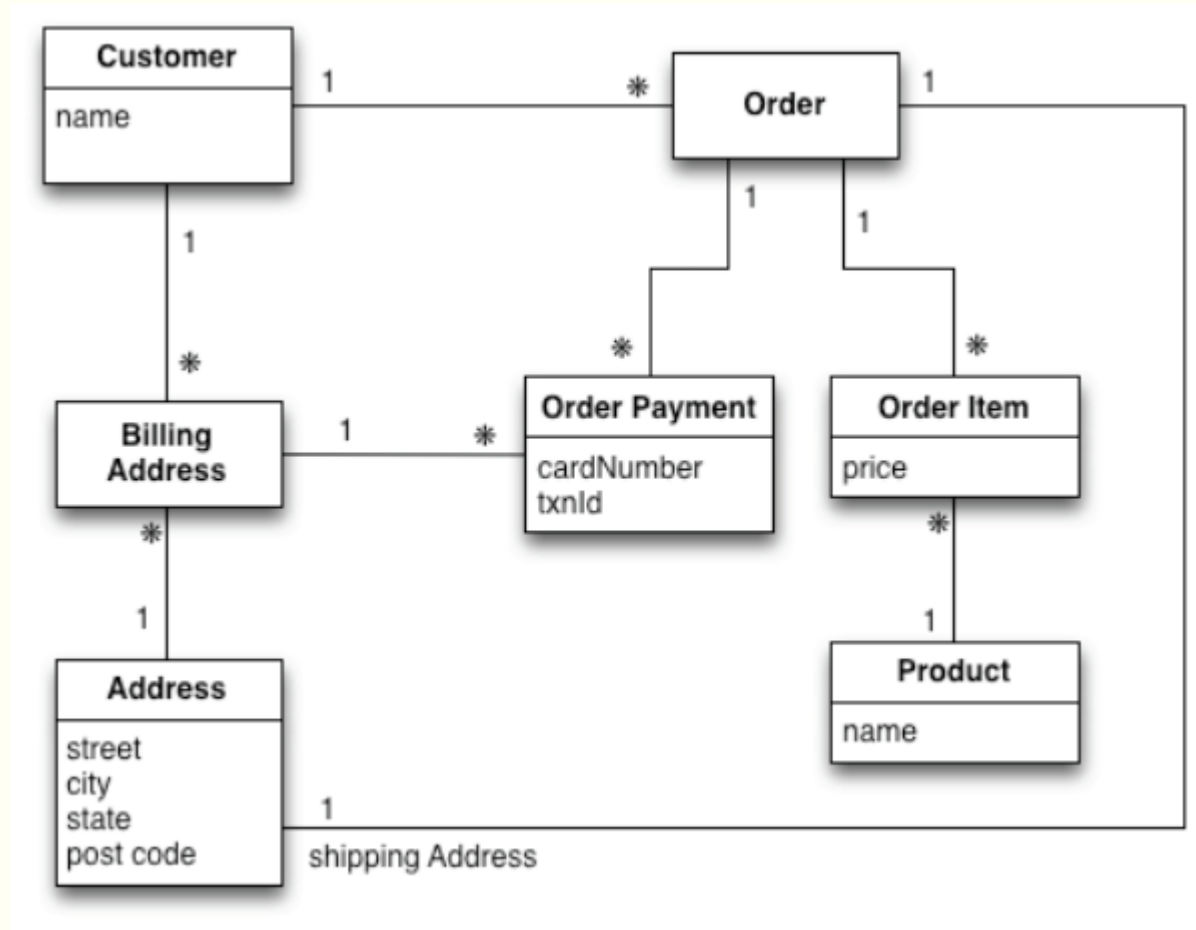


# MODULE 1

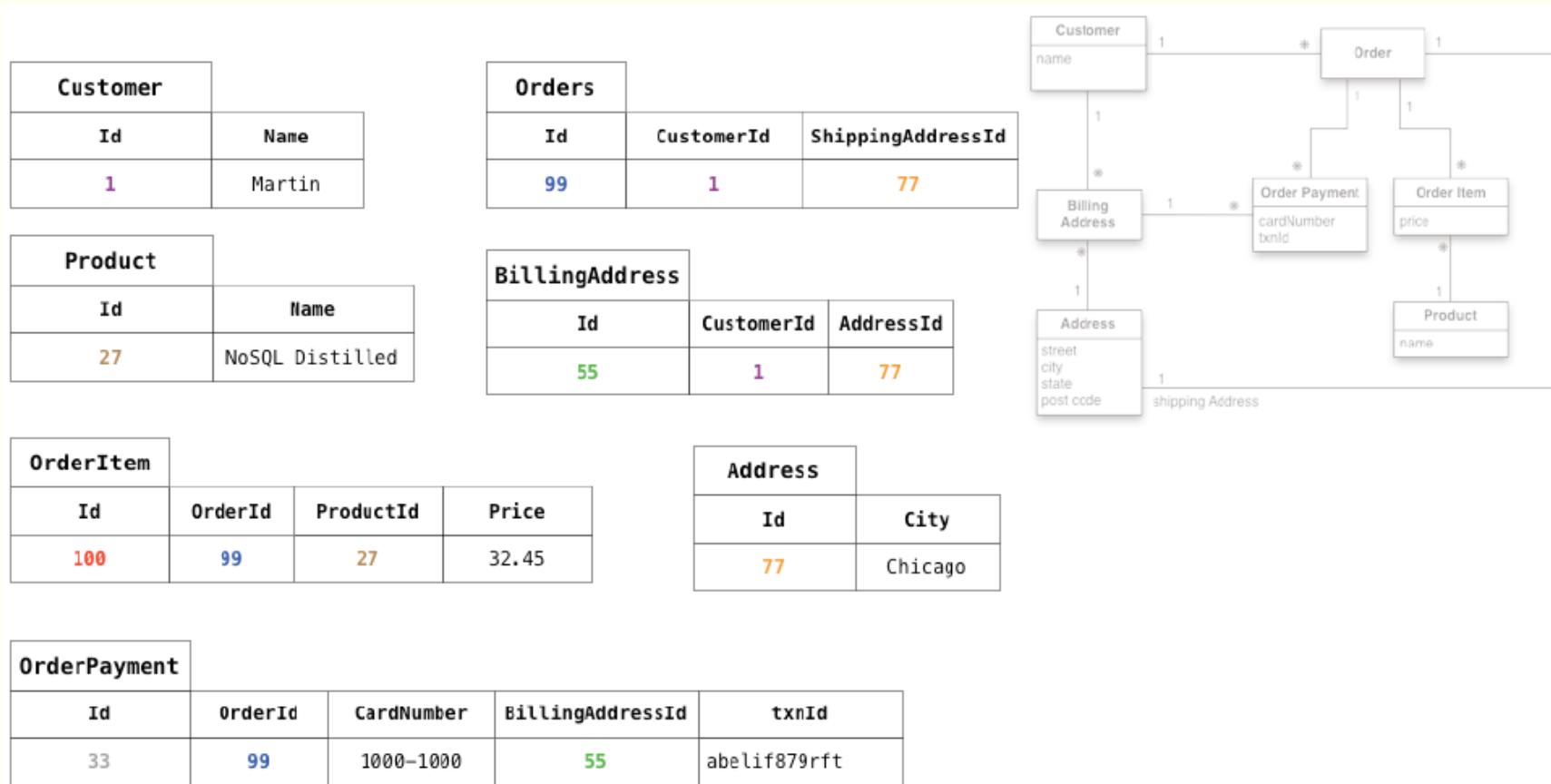
Introduction au NoSQL

# Domain driven data models

---



# RDBMS data



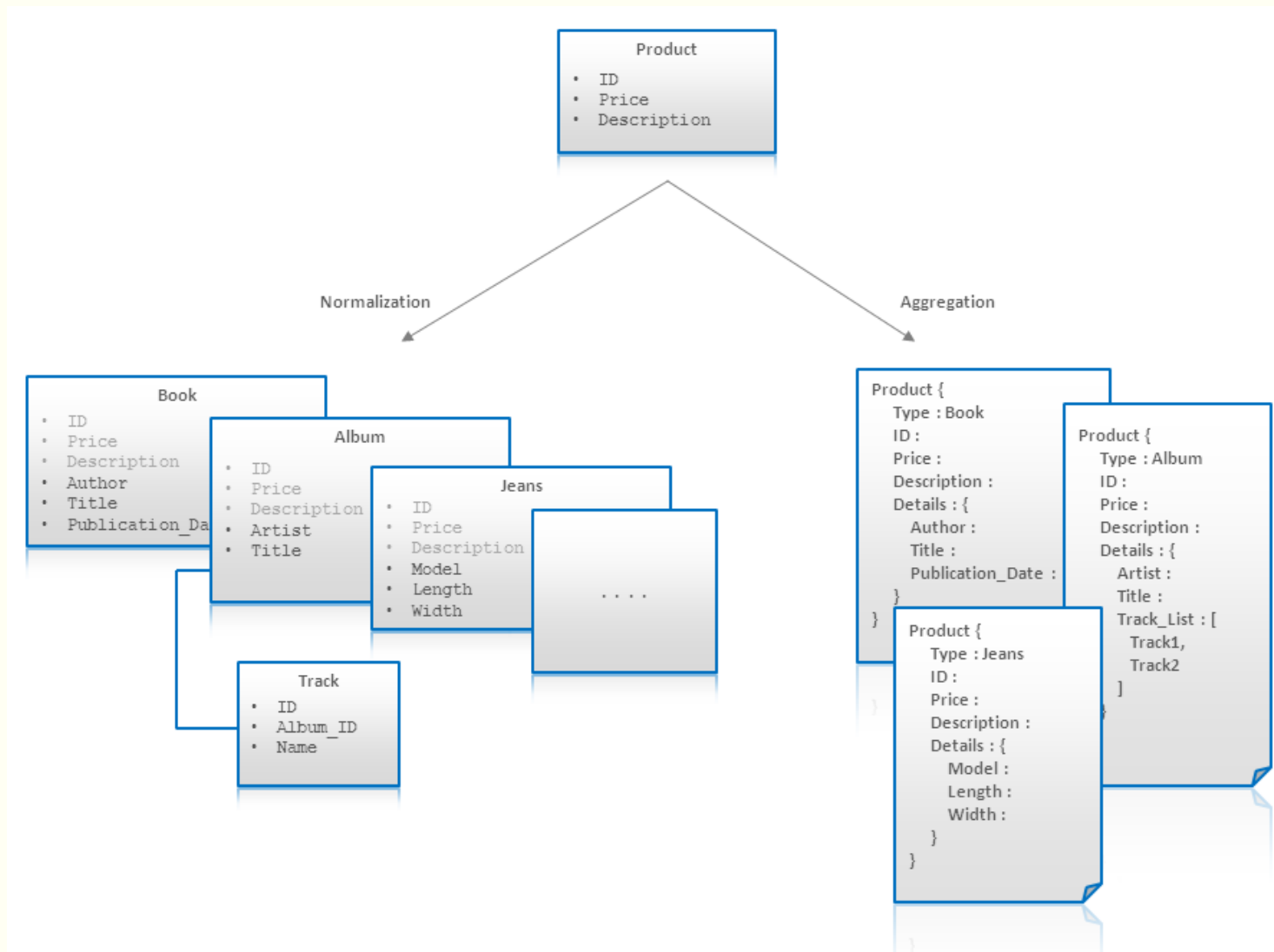
# Le monde Relationnel

---

## Caractéristiques:

- Données structurées (Tables/Schémas)
- Données Normalisées (Formes Normales)
- Standard (SQL)
- Transactionnel (A.C.I.D)
- Requête Complexe (Jointure)
- Des contraintes(Intégrité des données)
- ...

# Normalisation vs Agrégation



# RDBMS et Clusters

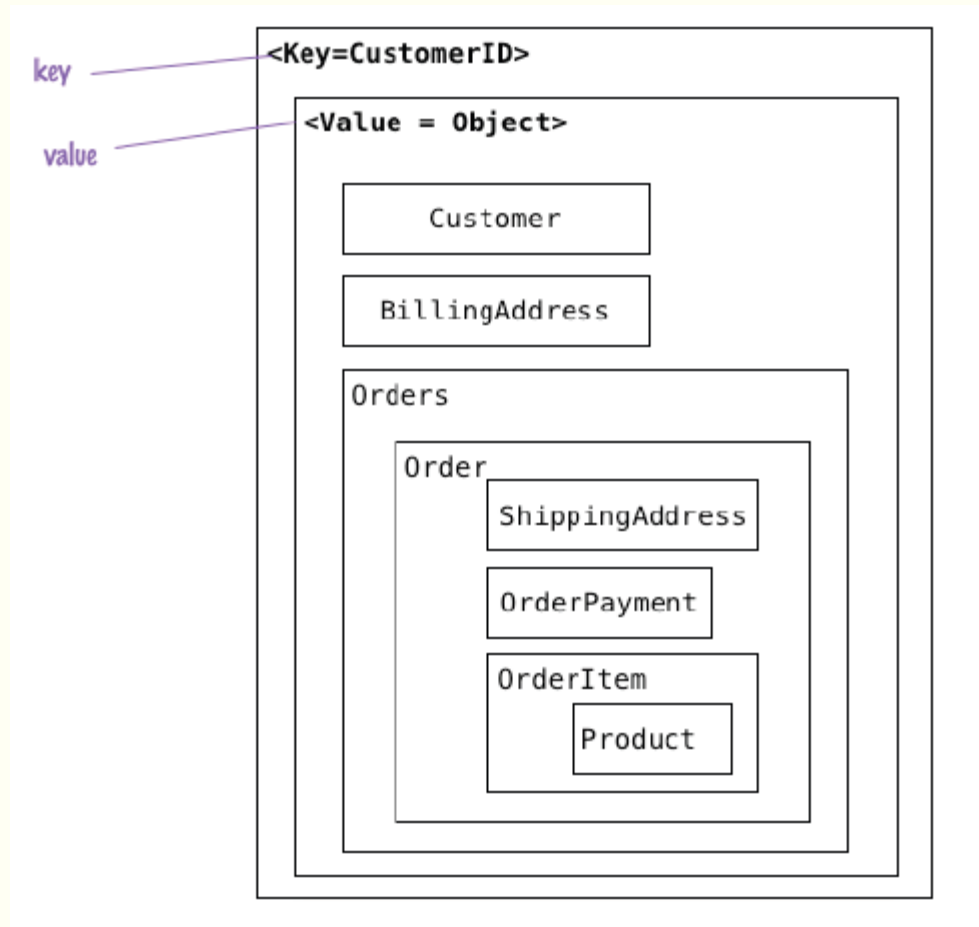
---





# 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"}
          }
        ]
      }
    ]
  }
}
```

# Big Data

---



# Les évolutions...

---

- **Nouvelles Données :**

- Web 2.0 : Facebook, Twitter, news, blogs,
- Flux : capteurs, GPS, ...

→ très gros volumes, données pas ou faiblement structurées

- **Nouveaux Traitements :**

- Moteurs de recherche , Extraction, analyse, ...
- Recommandation, filtrage collaboratif, ...

→ transformation, agrégation, indexation

- **Nouvelles Infrastructures :**

- Cluster, réseaux mobiles, microprocesseurs multi-coeurs, ...

→ distribution, parallélisation, redondance

# Le NoSQL

---

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

- Très haute disponibilité
- Grandes performances en lecture (et/ou) écriture
- Traitement de gros volumes de données.

# Le NoSQL

---

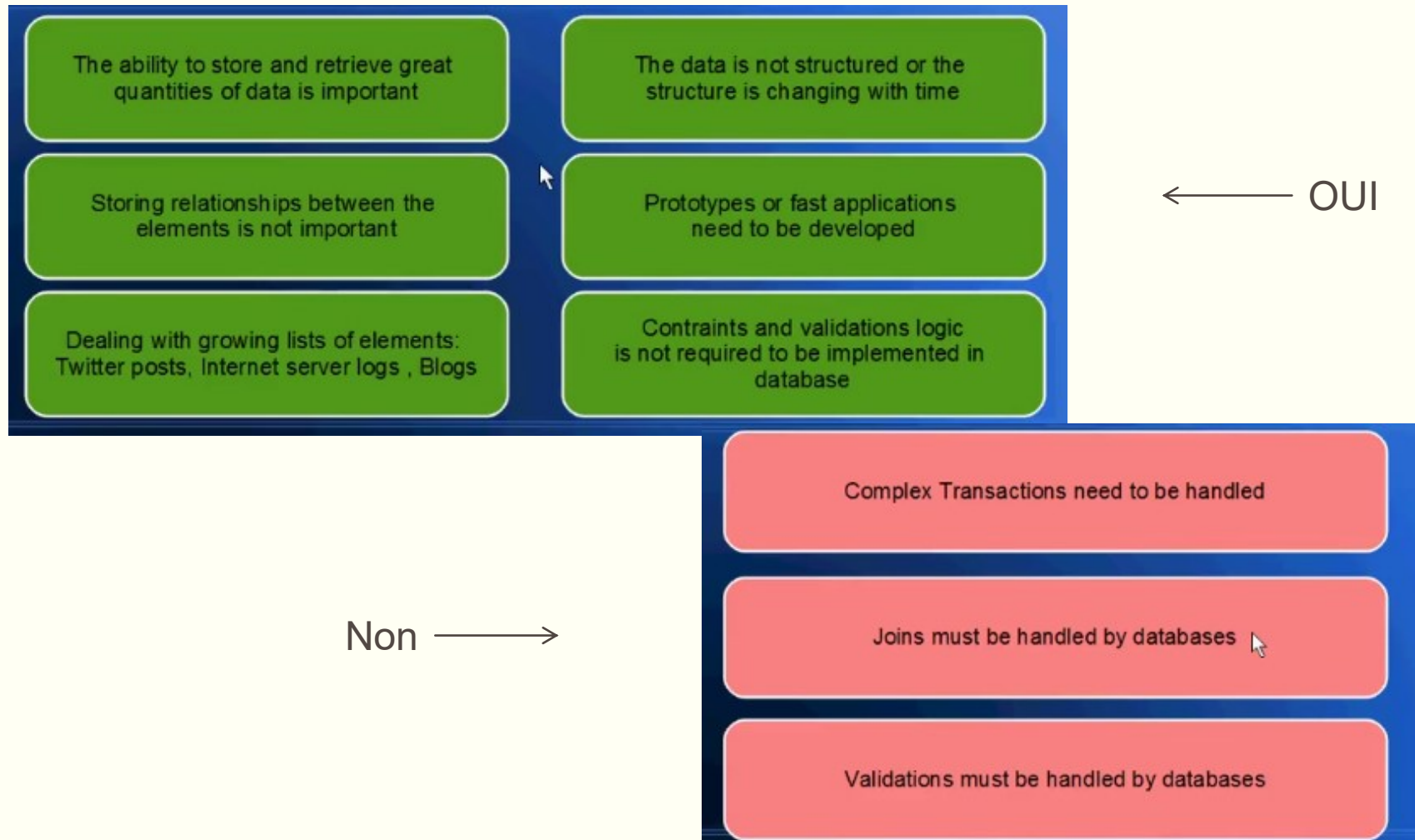
Un moteur NoSQL c'est...

- 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 compromis

# Le NoSQL

---

Alors, SQL versus NoSQL ?



# Théorème CAP

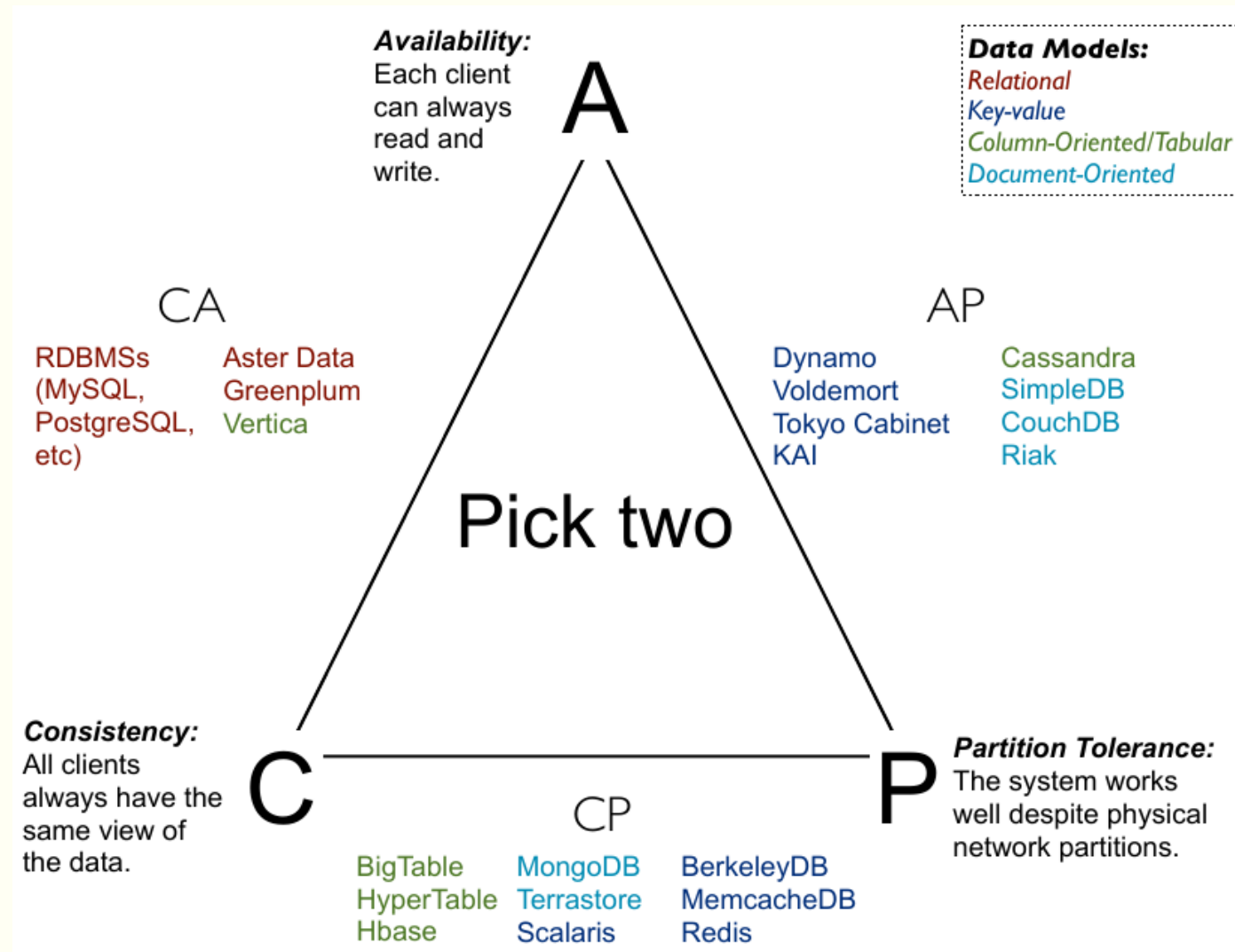
---

**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



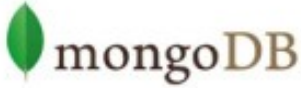













# Guide visuel des solutions NoSQL



# 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" ]  
}
```



← field: value  
← field: value  
← field: value  
← field: value

# MongoDB

---

- Une base de données orientée document signifie :
  - 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

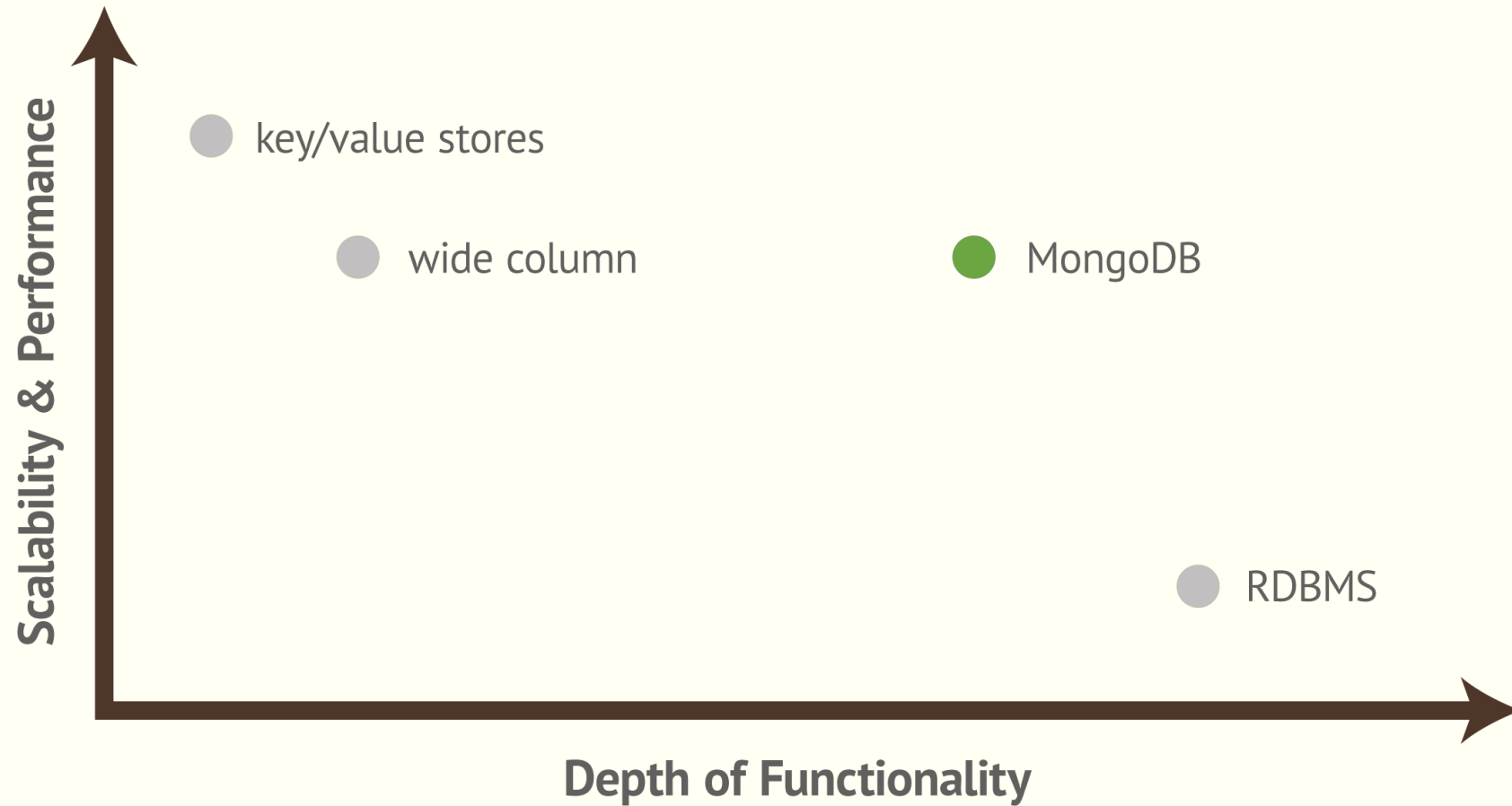
---

MongoDB est un SGBD :

- Orienté documents
- Open source
- Scalable : réplication / autosharding
- Flexible : pas de schéma de données, full-text index
- Ecrit 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
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

### CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	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

NO RELATION

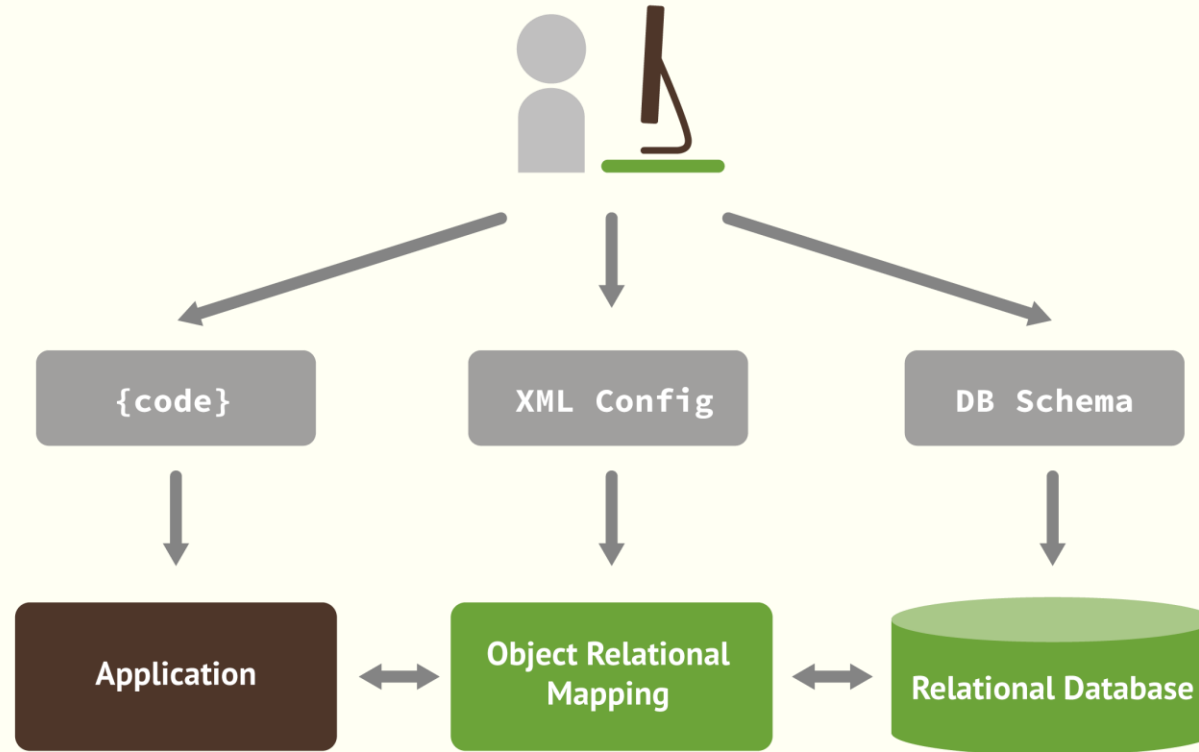
## Document - Collections

```
{ 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, ... }  
  ]  
}
```



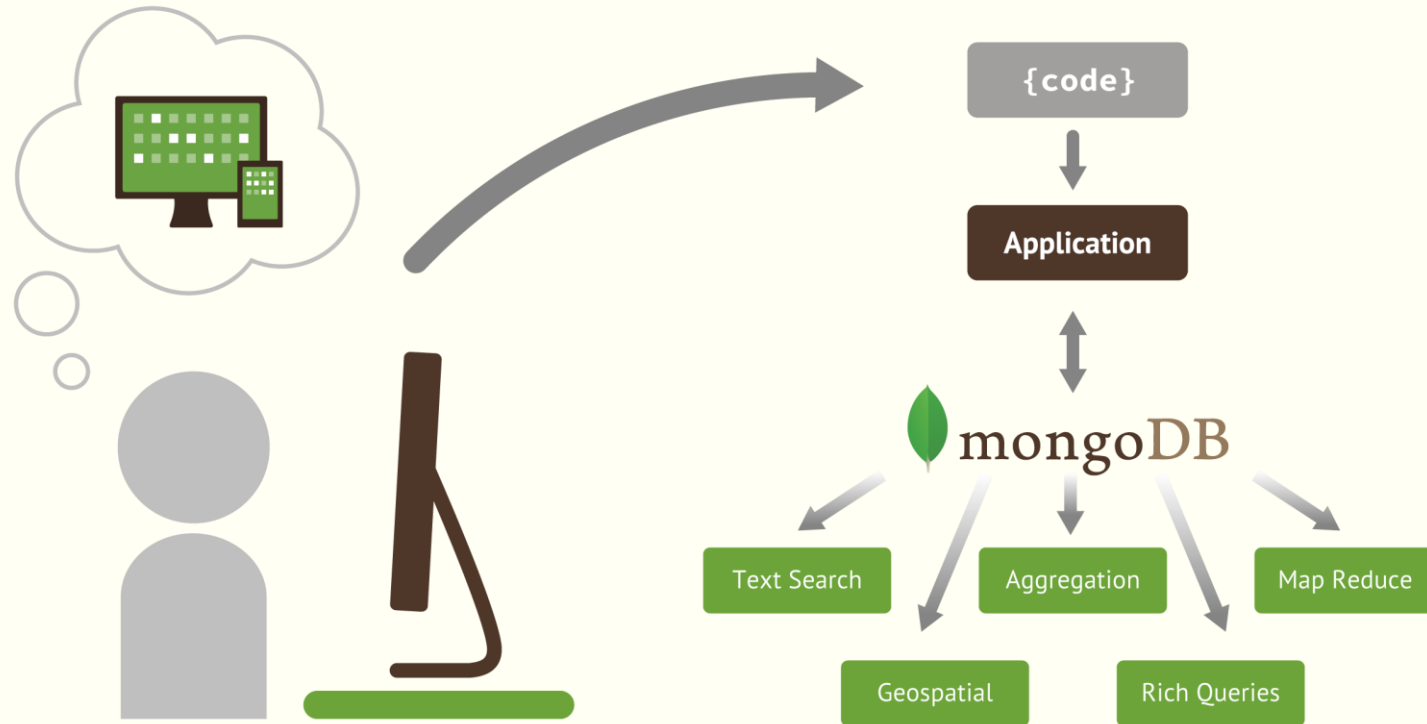
# Développement traditionnel

---



# Développement MongoDB

---



# Shell and Drivers

---

## Drivers

Drivers pour la majorité des langages de programmation et frameworks



Java



Ruby



JavaScript



Perl



Python



Haskell

---

## Shell

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

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

# Vocabulaire

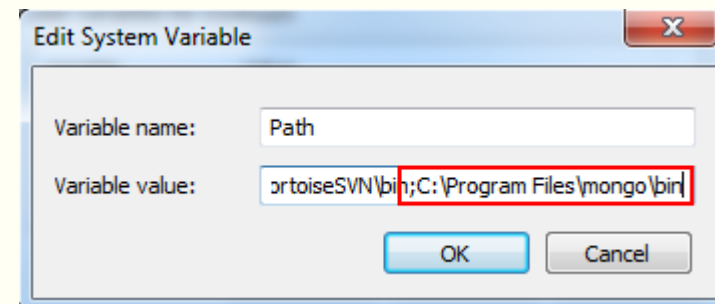
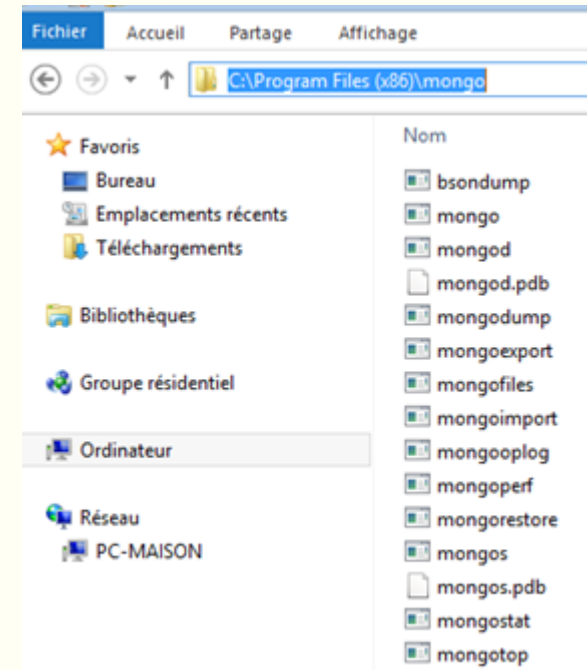
---

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

# Installation de MongoDB

---

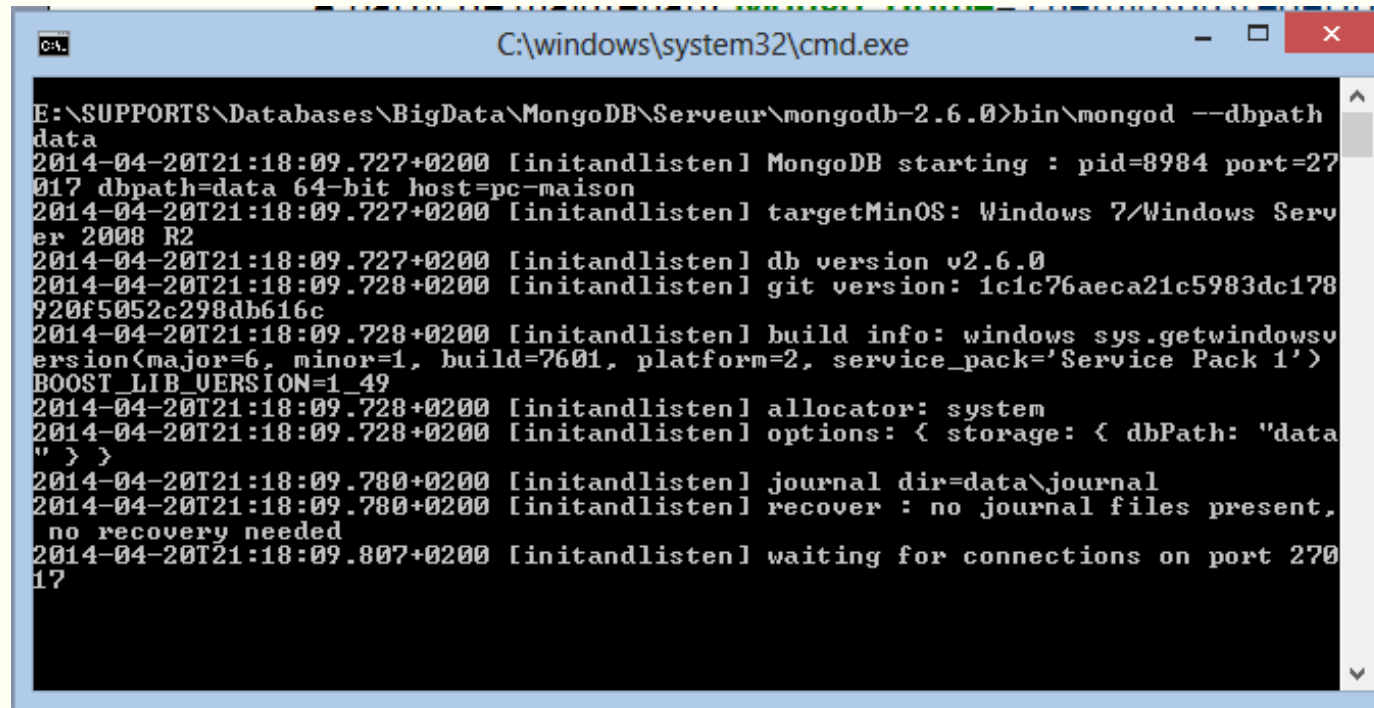
- Télécharger
  - <http://www.mongodb.org/downloads>
- Décompressez/Renommez
  - Déplacer dans par exemple:  
**C:\Programs Files\mongo**
- Création du dossier de données
  - Dans par exemple : **c:\data**
- Configuration des variables d'environnement
  - Path: **C:\Programs Files\mongo\bin**



# Démarrez une instance MongoDB

---

- Ouvrir un shell (ligne de commande DOS) et tapez:  
`mongod --dbpath c:\data`



```
C:\windows\system32\cmd.exe

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: 1c1c76aeca21c5983dc178
920f5052c298db616c
2014-04-20T21:18:09.728+0200 [initandlisten] build info: windows sys.getwindowsv
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**

session en cours

La BDD de la



# Commandes de base

---

Syntaxe

**db.dropDatabase()**  
database

Supprimer une

Syntaxe

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

Syntaxe

**db.nomCollection.drop()**  
Supprimer une collection

Syntaxe

**Crtl+C**  
Quitter



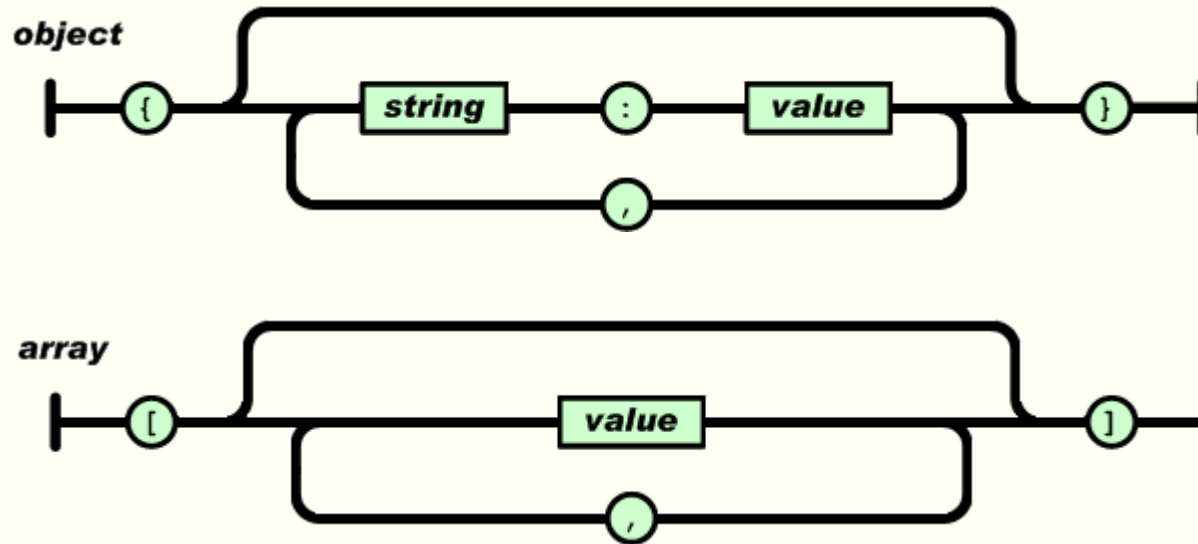
# MODULE 2

Opération CRUD

# Document JSON-Format

---

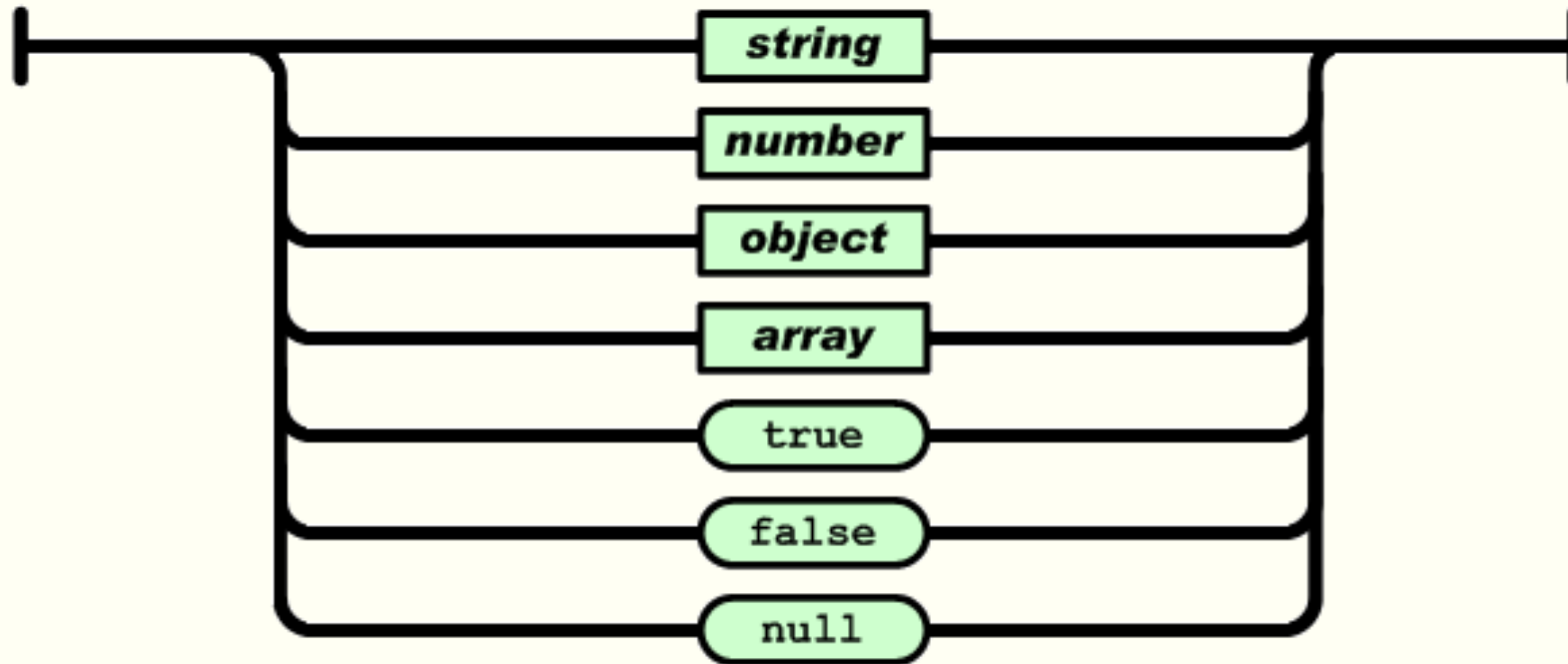
- Official Site: <http://json.org/>
- Un document **json** est de la forme: **{ }** (object) ou **[ ]** (tableau).



# 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:

	<u>MongoDB</u>	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'existait 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
}                  } document
)
```

```
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 } },
  { $set: { status: "A" } },
  { multi: true }
)
```

← collection  
← update criteria  
← update action  
← update option

```
UPDATE users
SET status = 'A'
WHERE age > 18
```

← table  
← update action  
← 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 !)
- **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" }  
)
```

← collection  
← remove criteria

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

← table  
← delete criteria

# Lecture de document avec **find**

---

## Syntaxe

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

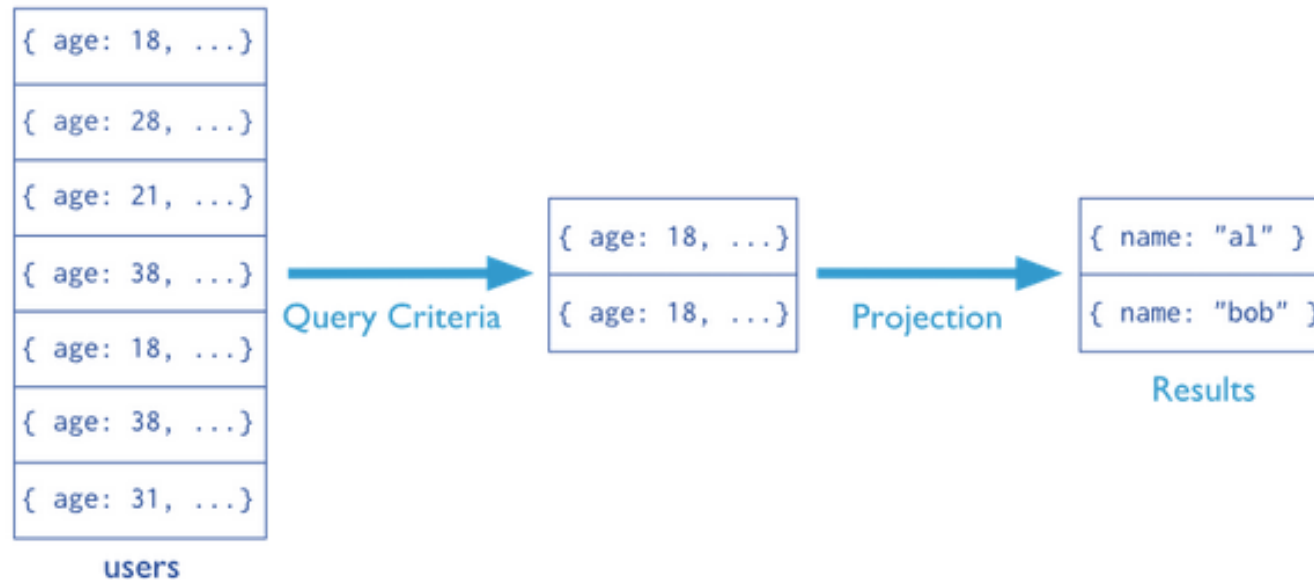
- **sans paramètre**, tous les documents sont renvoyés
- **criteria** est un tableau clefs / valeurs spécifiant des opérateurs sur les champs des documents recherchés
- **projection** est un tableau permettant de limiter les champs que l'on souhaite consulter dans les documents recherchés (cette option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



# 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' ]
}
```





# MODULE 3

Les requêtes

# 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 repertoire `/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" : "boulangier" }
```

```
> 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 «.»

```
> db.users.findOne()
{
  "_id" : ObjectId("535be5b1a8f64ced71fb8b81"),
  "name" : "richard",
  "email" : {
    "work" : "richard@yahoo.fr",
    "personnal" : "richard@gmail.com"
  }
}
```

[illegible]

# 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 })
```



# 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

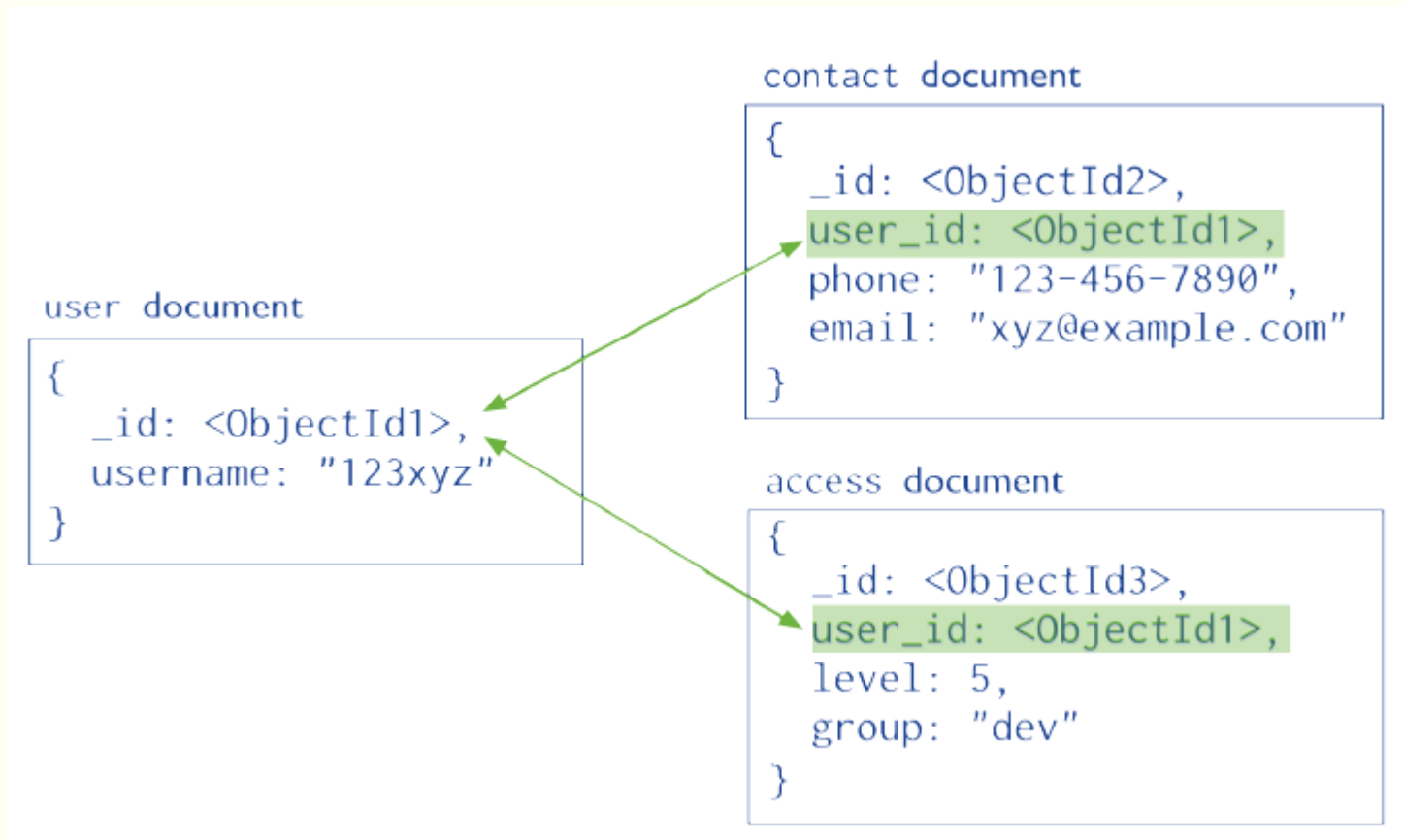
---

- Embedding
- Linking (par reference)



# Linking

---



# Embedding & Linking

---

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

Embedded sub-document

Embedded sub-document



# MODULE 5

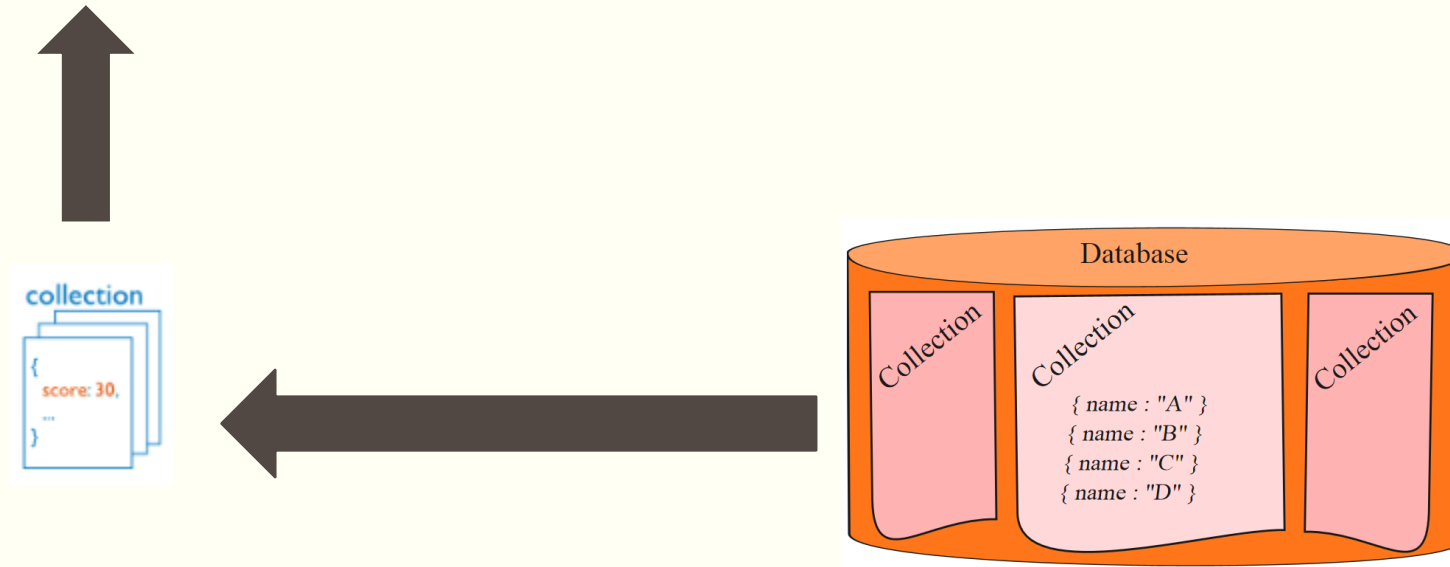
Performance

# 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.

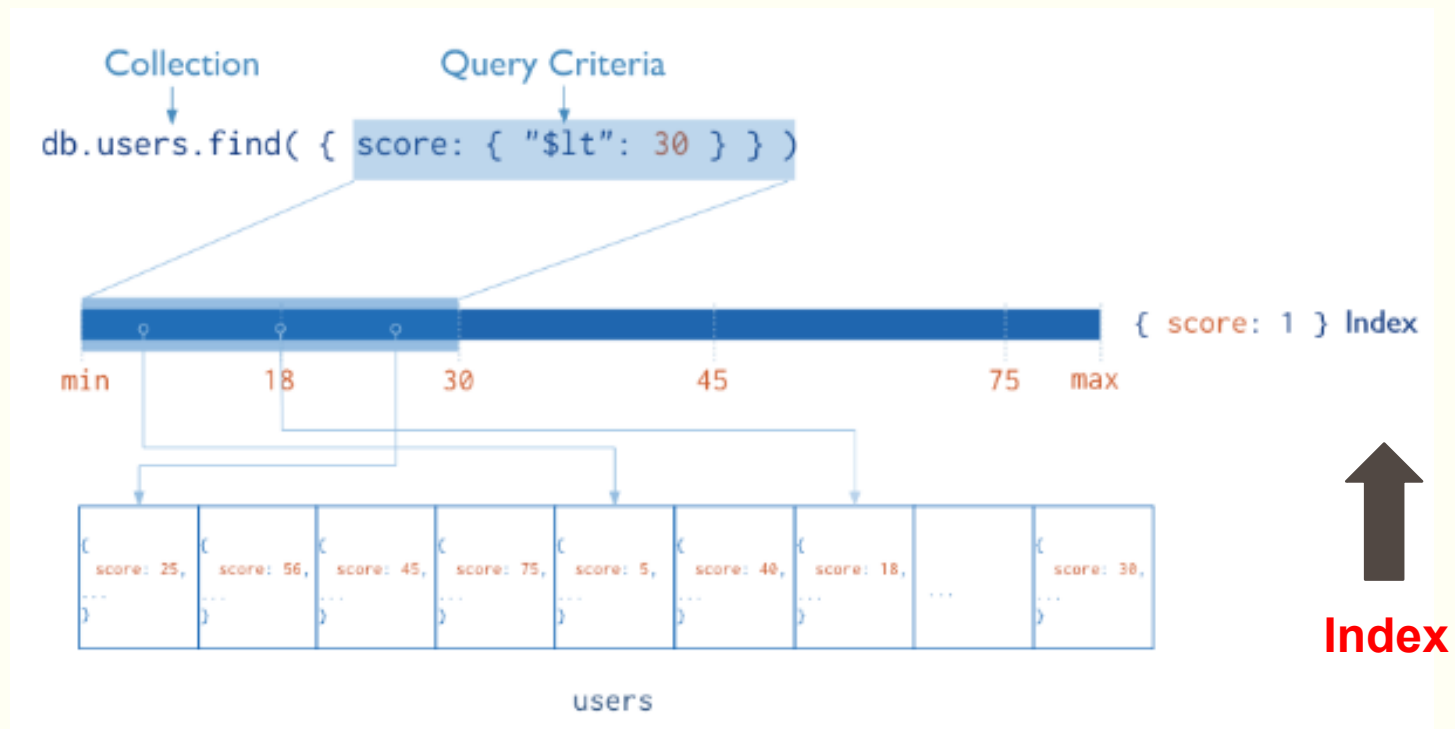


Diagram of a query that uses an index to select

# Index in MongoDB

---

## ► 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})`

- Override MongoDB's default index selection

# Index Types

---

- Single Field Indexes (Index champ unique)
- Compound Field Indexes (Index à 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})
```

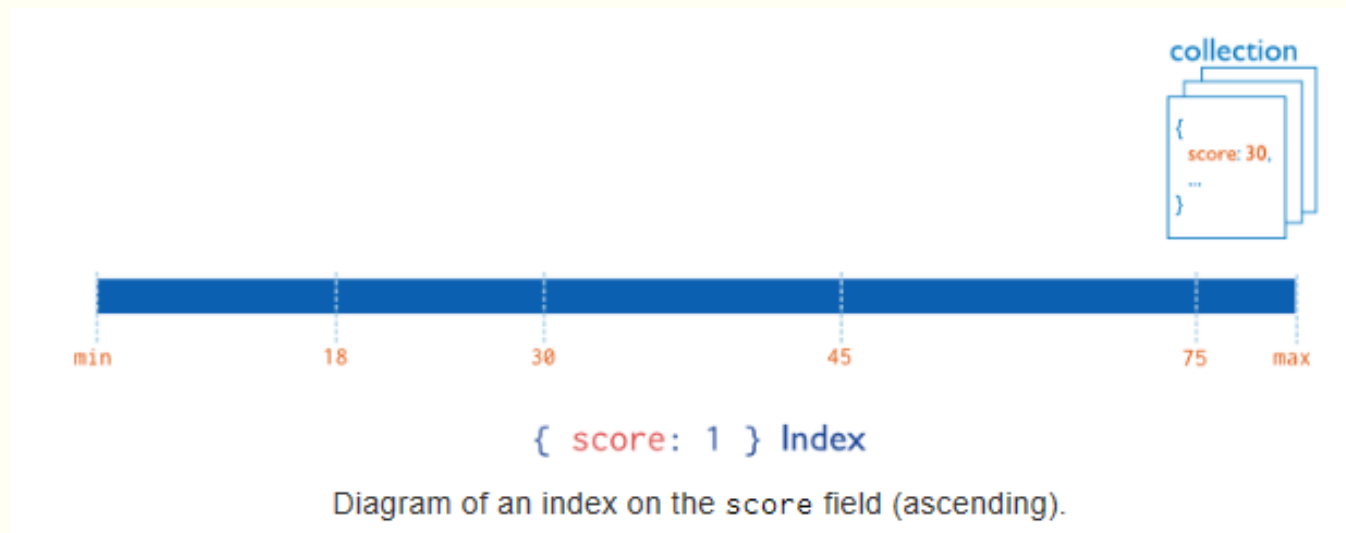


# Single Field Indexes

---

- Single Field Indexes

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



# Single Field Indexes

---

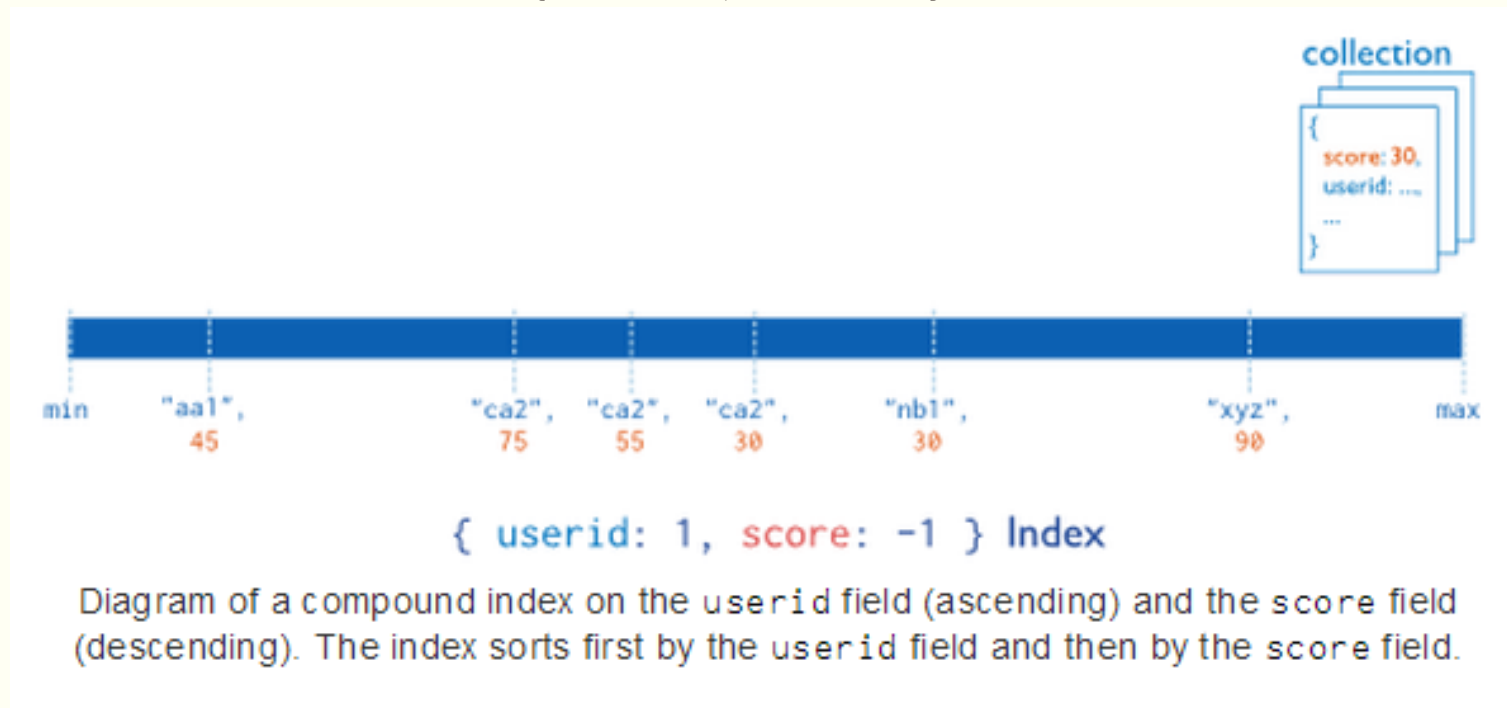
- `db.collection.ensureIndex({student_id:1})`
- `1`=ascending,
- `-1`=descending
  - important for sorting not so much for searching
- Sorting can also use a reverse index if the sort criteria are exactly the reverse of an (simple or compound) index

# Compound Field Indexes

---

- **Compound Field Indexes**

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



**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})`



# MODULE 6

Agrégation Framework

# Framework d'agrégation

---

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

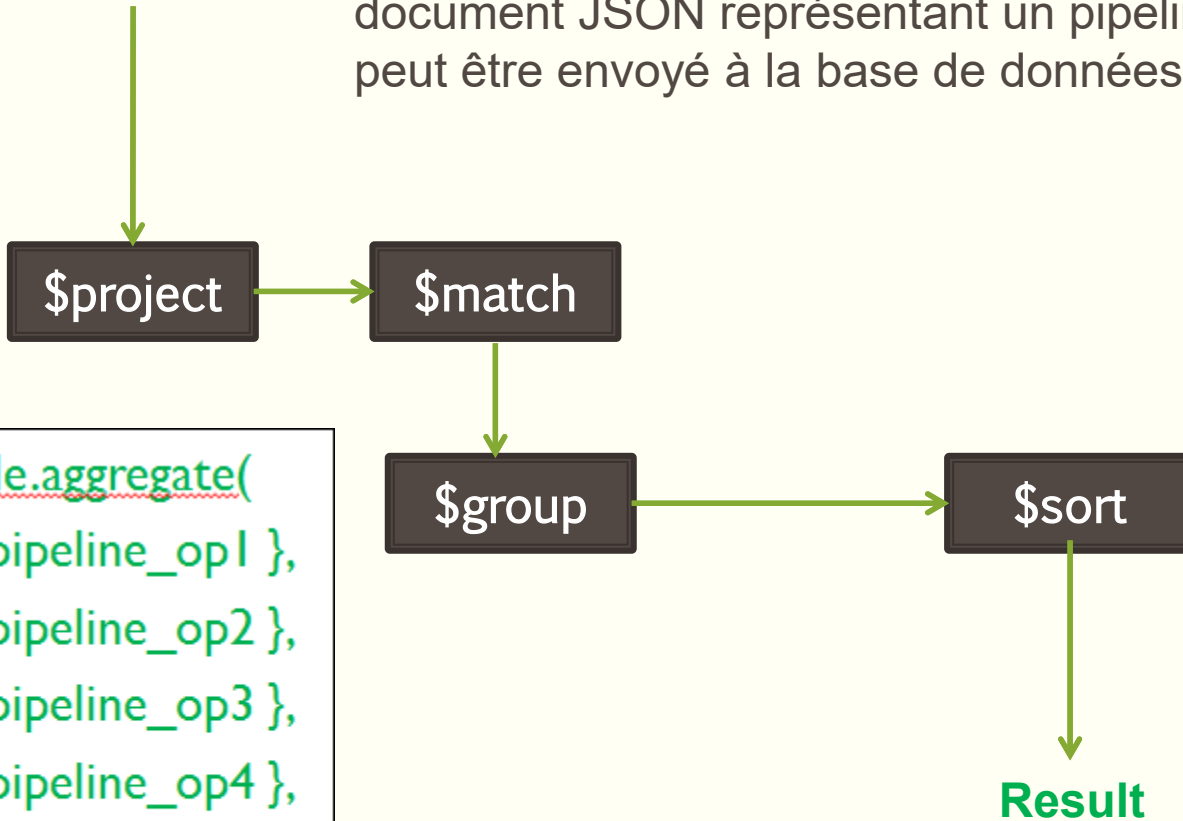
Avec les nouveaux opérateurs, les développeurs pourront trier et agréger par groupes les données sur lesquelles portent les requêtes et leur appliquer diverses opérations.

# Pipeline

---

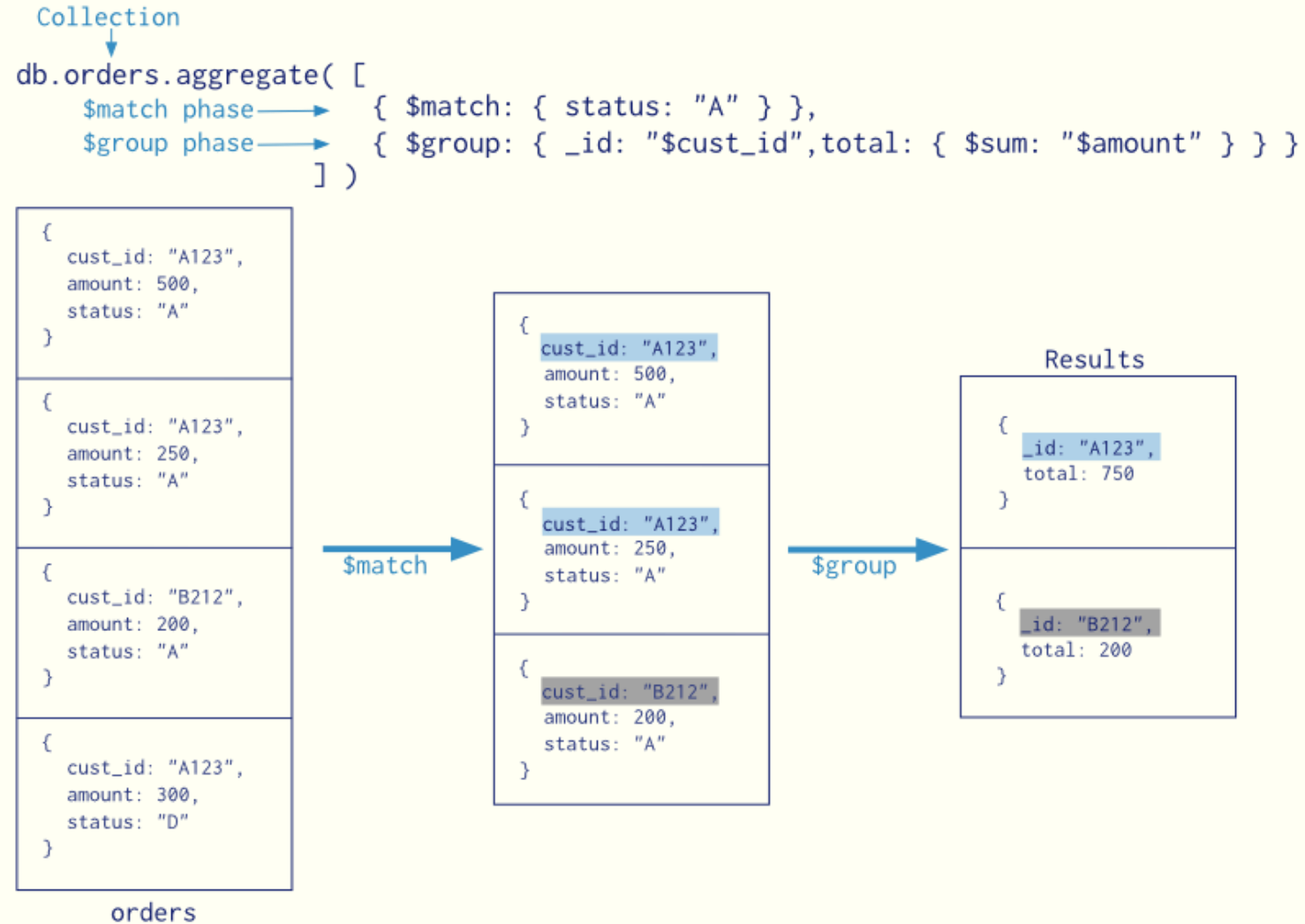
**Collection**

Pour effectuer une requête, le client construit un document JSON représentant un pipeline qui peut être envoyé à la base de données.



```
db.article.aggregate(  
  { $pipeline_op1 },  
  { $pipeline_op2 },  
  { $pipeline_op3 },  
  { $pipeline_op4 },  
  ...  
);
```

# Example I





## 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

# \$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([
  {$project:
    {
      _id:0,
      'maker': {$toLowerCase:"$manufacturer"},
      'details': {'category': "$category",
                  'price' : {"$multiply":["$price",10]}
                },
      'item':'$name'
    }
  }
])
```

```
{ 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

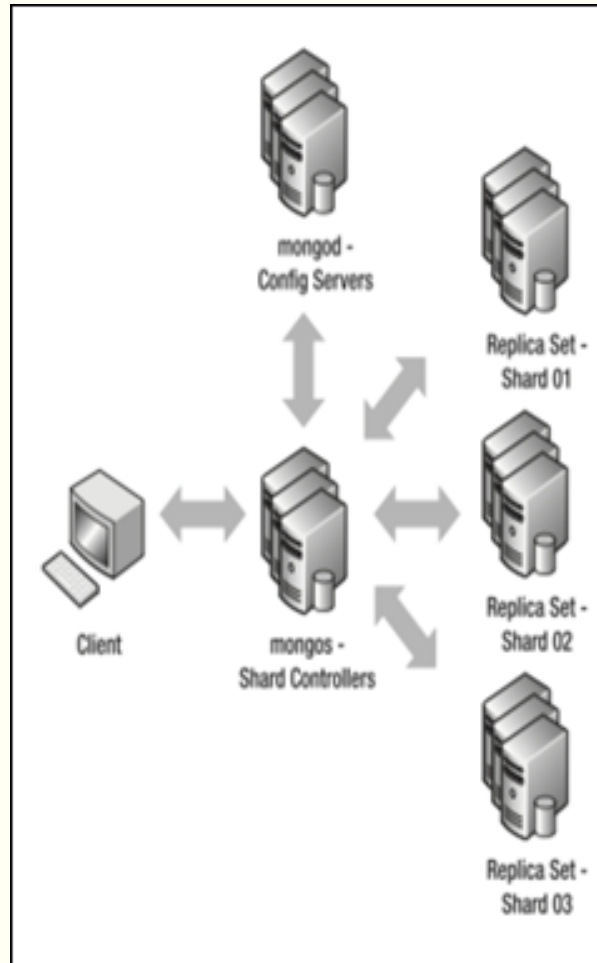


# MODULE 7

Administration

# Replication & Sharding

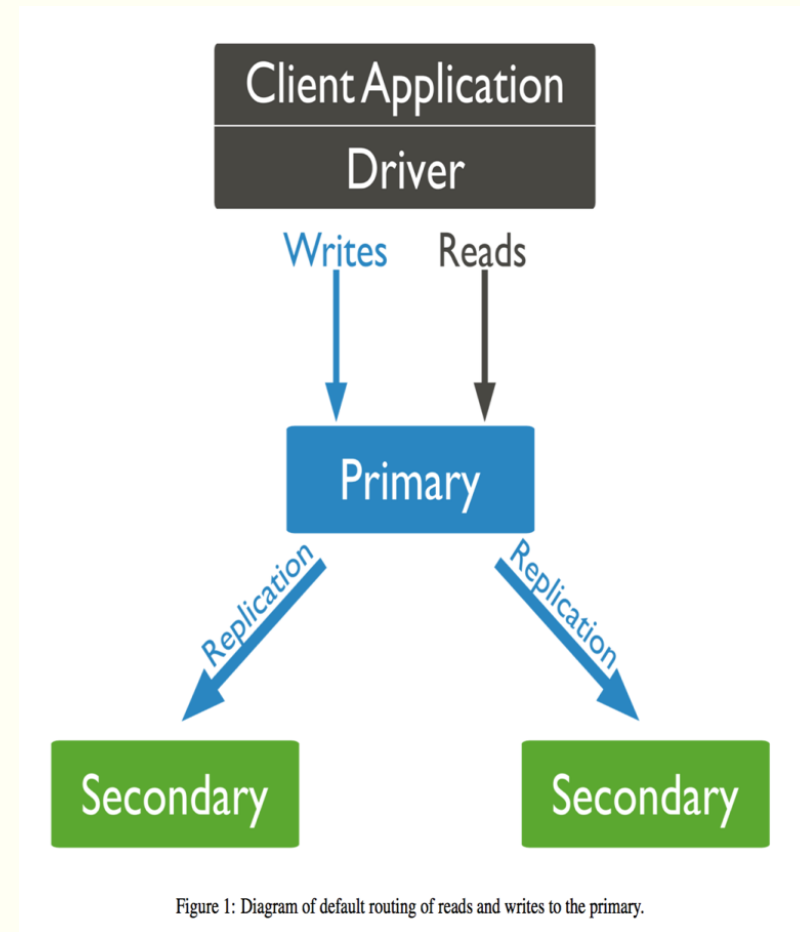
---



# MongoDB réplication

---

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



# 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

---

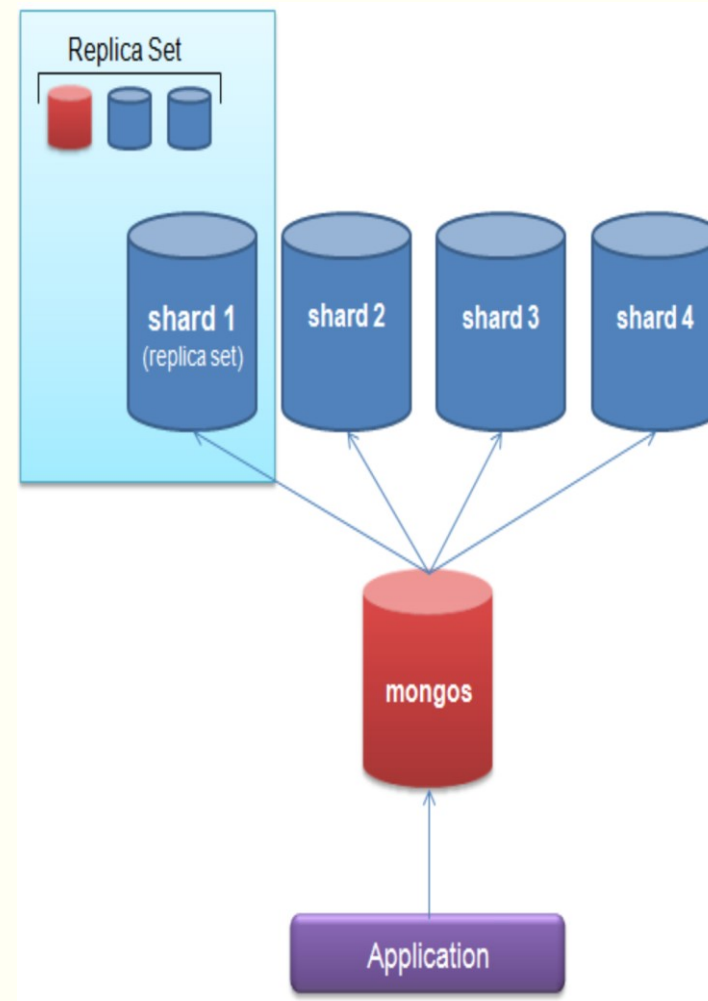
- La réplication du maître vers les esclaves est asynchrone.
- Quels sont les avantages et inconvénients ?
  - Synchrones : Bloquant / Coûteux / Forte cohérence
  - Asynchrone : Non bloquant / Rafraîchissement des données obligatoires.



# 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

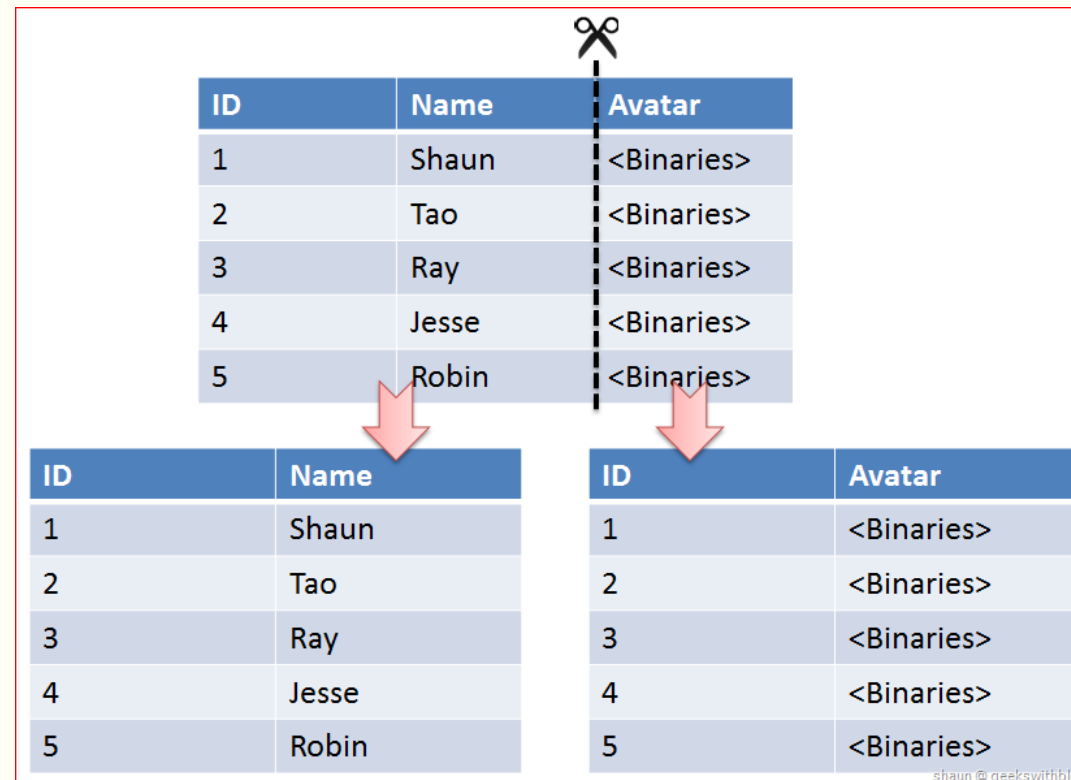
---

- 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

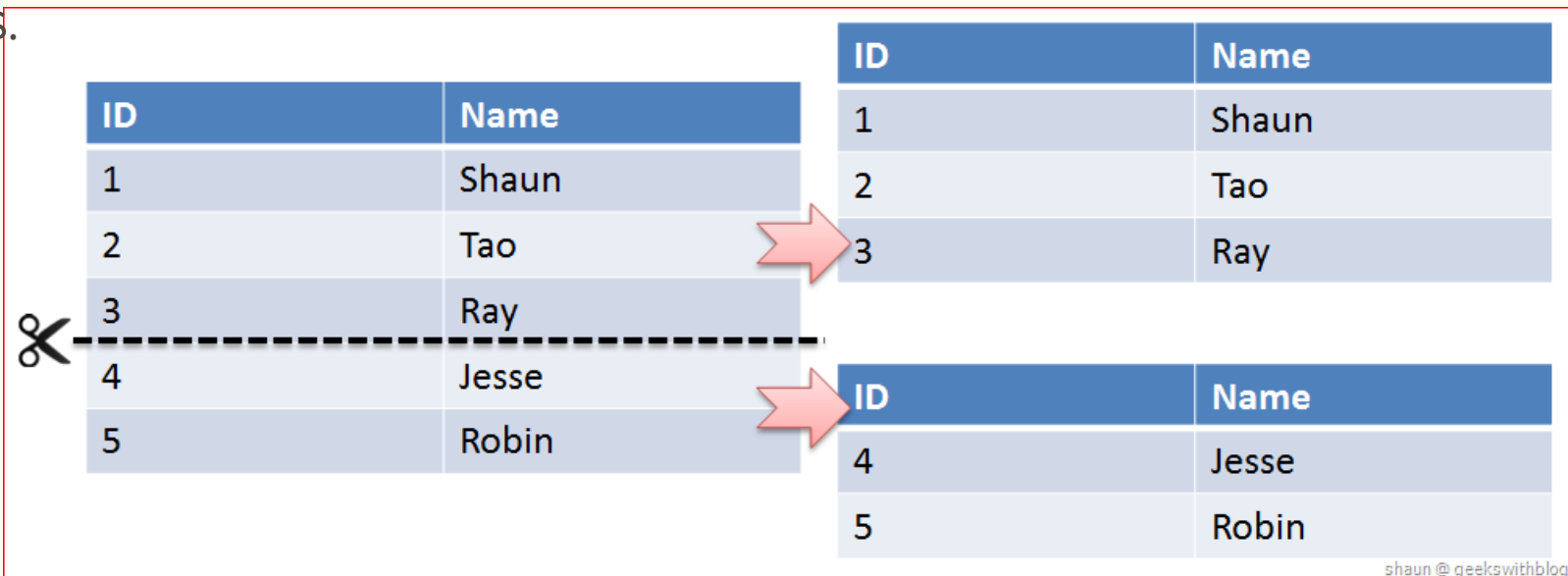
---

- 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

---

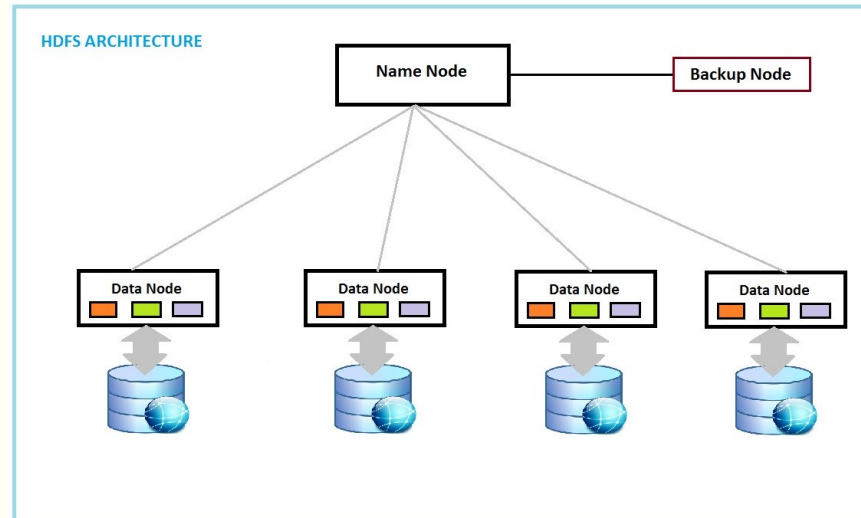
- Les grands du web ont, sur la base de ce choix de scalabilité horizontale, développé des solutions spécifiques (NoSQL) répondant à ces enjeux et ayant les caractéristiques suivantes :
  - une implémentation à partir de machines de grande série
    - Basée sur l'allocation de ressources : HDFS
  - une 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...





# 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  
        MongoDBDatabase db = mongoClient.getDatabase("test");  
        System.out.println("Connect to database successfully ");  
  
        MongoClientCollection<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);

// critere 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);
```