Journées d'échanges CATIs CODEX – SICPA
Novembre 2015

MongoDb:
Un SGBD NoSql orienté « document »

NoSql

Principes

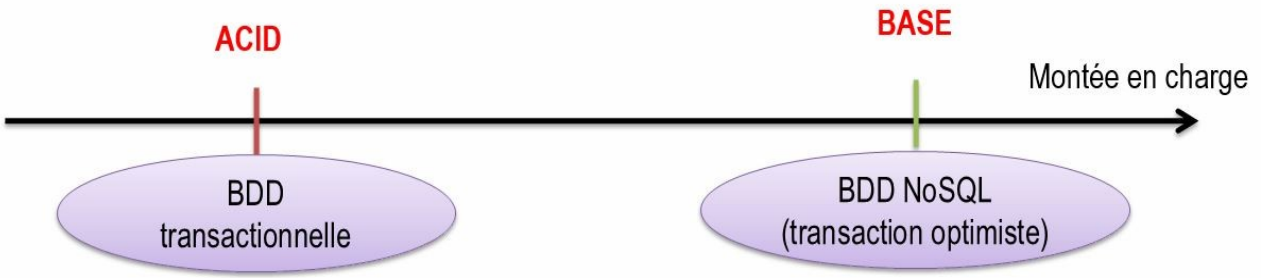
Google (Map Reduce, Big Table) et **Amazone** (Dynamo) pour faire face à la monté en charge liée au BigData

Les SGBD NoSql partagés ne peuvent satisfaire que 2 critères au plus

mongoDB

NoSql

Les transactions



- **A**tomicity : Une transaction est une instruction appliquée totalement ou rollbacké totalement
  Consistency : Toutes transaction effectuée doit garder des données cohérentes et respecter les contraintes de la base (index, typage…)
- **I**solation : Les transactions ne peuvent pas interférer les une avec les autres. Elles sont isolées.
- **D**urability : après l'exécution d'une transaction, ces effets seront permanents sur la base de données.

- **B**asically **A**vailable : les données sont disponibles selon le théorème CAP (haute disponibilité). La réponse ne sera pas forcement juste à 100% mais disponible.
- **S**oft state : la cohérence des données n'est pas géré par la base de données mais par les développeurs.
- **E**ventual consistency : La cohérence des données n'est pas garantie à un instant t mais les données convergeront plus tard et seront cohérentes.

NoSql

Les transactions

La différence entre Relationnel et NoSQL

- Relationnel est normalisé donc les transactions sont essentielles et très fréquentes. Utilisation intensive de l'ACID.
- Or l'ACID sur un cluster est **coûteux en temps**

Relationnel

NoSQL

- NoSQL est pensé pour limiter le nombre de relations entre les tables donc limiter le besoin de l'ACID. Donc plus performant.
- Il scale plus ou moins facilement sur un cluster en fonction du niveau de cohérence demandé

mongoDB

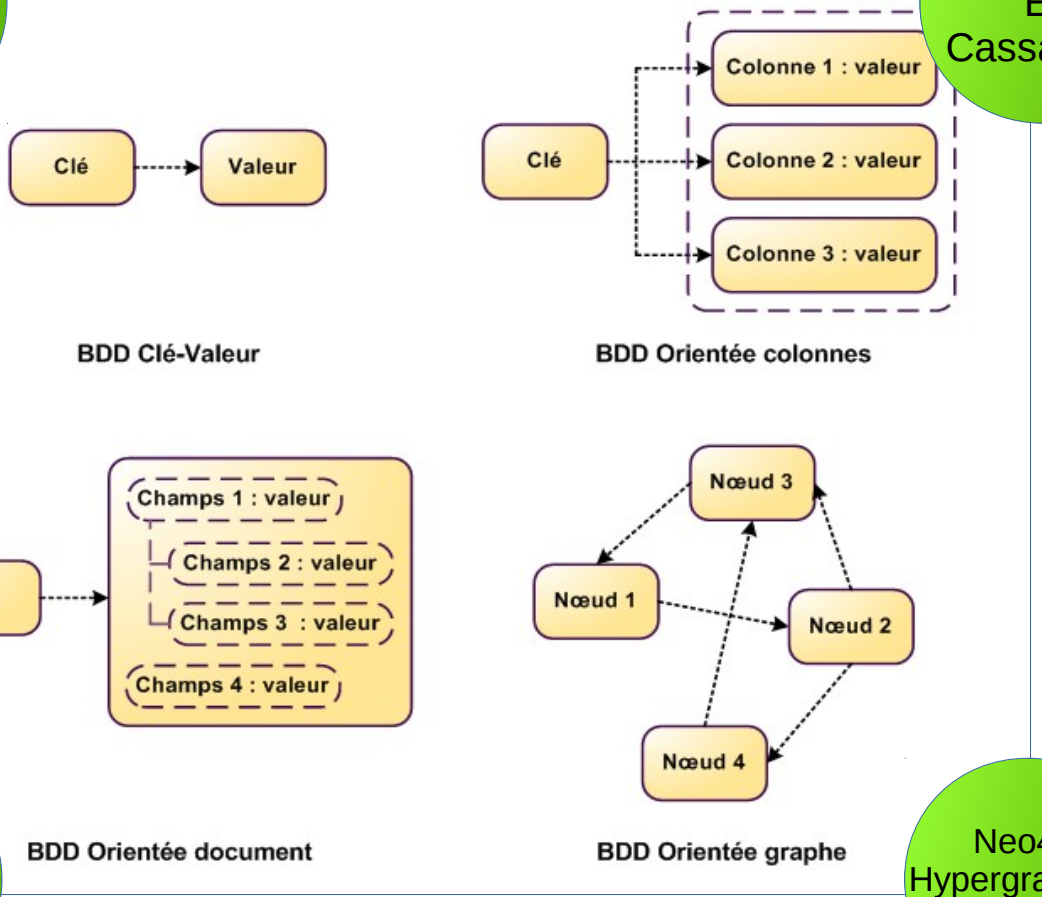
NoSql
Familles

Riak, Redis, Voldemort.

HBase Et Cassandra

ElasticSearch Solr

Indexe Iversé (Lucene)



CouchDB Et MongoDB.

Neo4j, HypergraphDB, FlockDB

mongoDB

NoSql

Warning

1/ Il ne s'agit pas d'une technologie de remplacement des SGBDR. Les NoSQL **apportent simplement des options alternatives** quant au mode de représentation de données utilisé pour un projet.

2/ Les NoSQL sont des projets encore récents, l**'outillage est donc réduit** et les communautés ne sont pas comparables à celles des SGDBR courants, ce qui limitera les possibilités de réponses aux problèmes rencontrés. Ces produits sont encore **très loin de la maturité de MySQL ou PostgreSQL.**

3/ L'intérêt d'une base de données NoSQL pour un projet **ne dépend pas du volume de données qu'elle aura à manipuler. Le choix de son utilisation doit être basé sur la préférence d'un mode de représentation.**

4/ Il ne s'agit pas d'une solution miracle pour tout type de stockage de données. **La tentative de reproduire dans une base de données NoSQL une représentation ou un comportement habituellement offert par un SGBDR aboutira probablement à une solution peu efficace.**

NoSql
MongoDb

- Orienté document – **CAP**

- Open source (licence Agpl)

- Librairies / API : Java, Php, C#, C, Javascript

- Langage de requête : **JavaScript**

- Pensé pour supporter de **gros volumes de données**, il implémente donc une logique de scalabilité horizontale avec du sharding*. MongoDB permet aussi d'implémenter MapReduce*

- Le plus populaire et accessible en passant du relationnel au NoSql avec de très bonnes performances sur les grosses bases.

- l'un des rares SGBD NoSql qui soit codé avec un langage de performance : le **C++** (les autres sont plus souvent codés en Java ou Erlang

- Permet de faire de la recherche avancée comme de la **recherche géospatiale** ou **sur du texte** en définissant la langue, pour ignorer les « stop words » (« et », « ou », « le », … en français par exemple) ou pour faire du stemming (remonter un document contenant « vaccination », quand on fait une recherche sur « vaccin »).

MongoDb

Vocabulaire

Traditional Database	MongoDB
Relational	Document-Orientated
Server	Server
Database	Database
Table	Collection
Row	Document
Column	Attribute
SQL Query	BSON Query
Index	Index

MongoDb
Format BSON

JSON

Validation Json : https://jsonformatter.curiousconcept.com/

```
{id: 1,
 name: Joe,
 url: '...',
 stream:
 [{
   user: 2,
   title: 'today',
   body: 'go fly a kite',
   likes: [
    {user: 3},
    {user: 1},
   ],
 }]
}
```

mongoDB → BSON

http://bsonspec.org/

BSON { 01010100 11101011 10101110 01010101 }

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON can be compared to binary interchange formats, like Protocol Buffers. BSON is more "schema-less" than Protocol Buffers, which can give it an advantage in flexibility but also a slight disadvantage in space efficiency (BSON has overhead for field names within the serialized data).

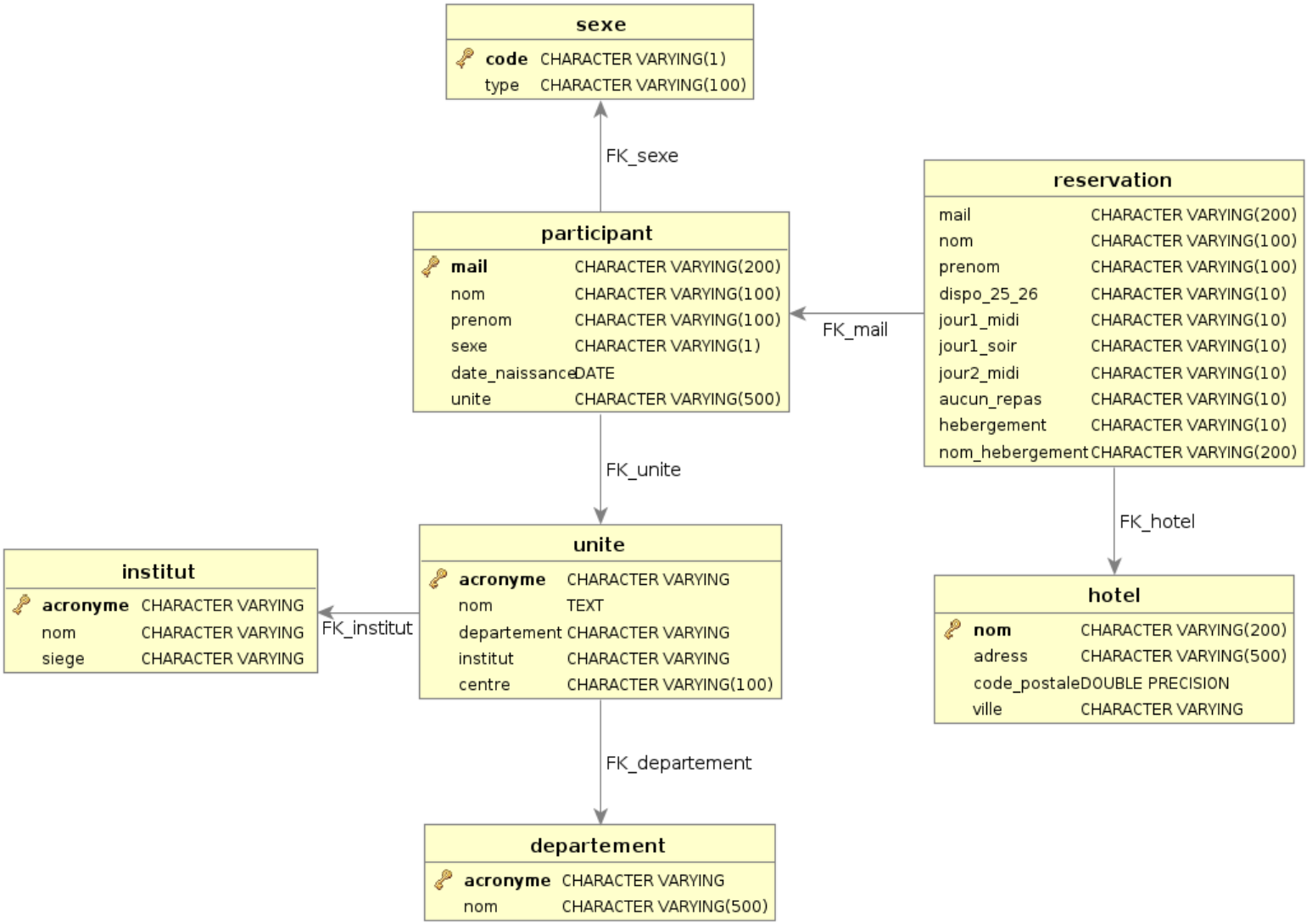
BSON was designed to have the following three characteristics:

1. **Lightweight**
   Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.
2. **Traversable**
   BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.
3. **Efficient**
   Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

specification | implementations | FAQ | discussion

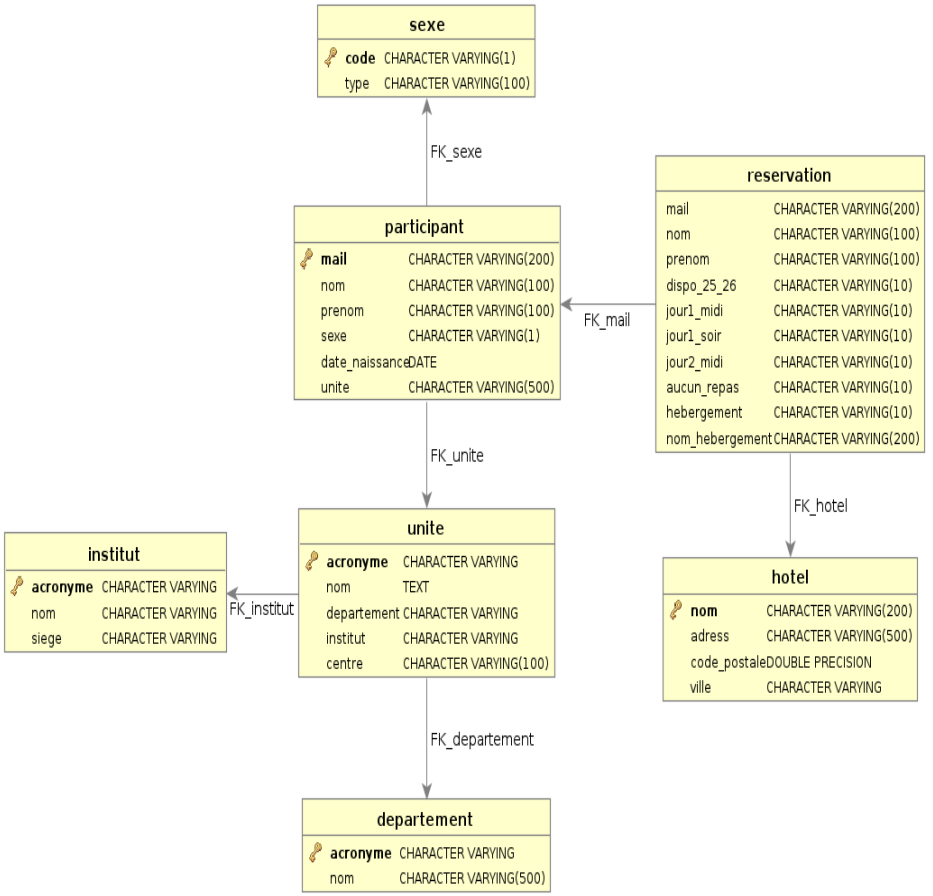
creative commons

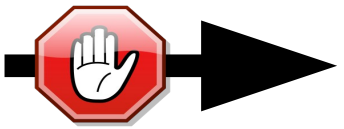
MongoDb
Conception



MongoDb

Conception structurée





Collection **participant**

Collection **reservation**

Collection **institut**

Collection **sexe**

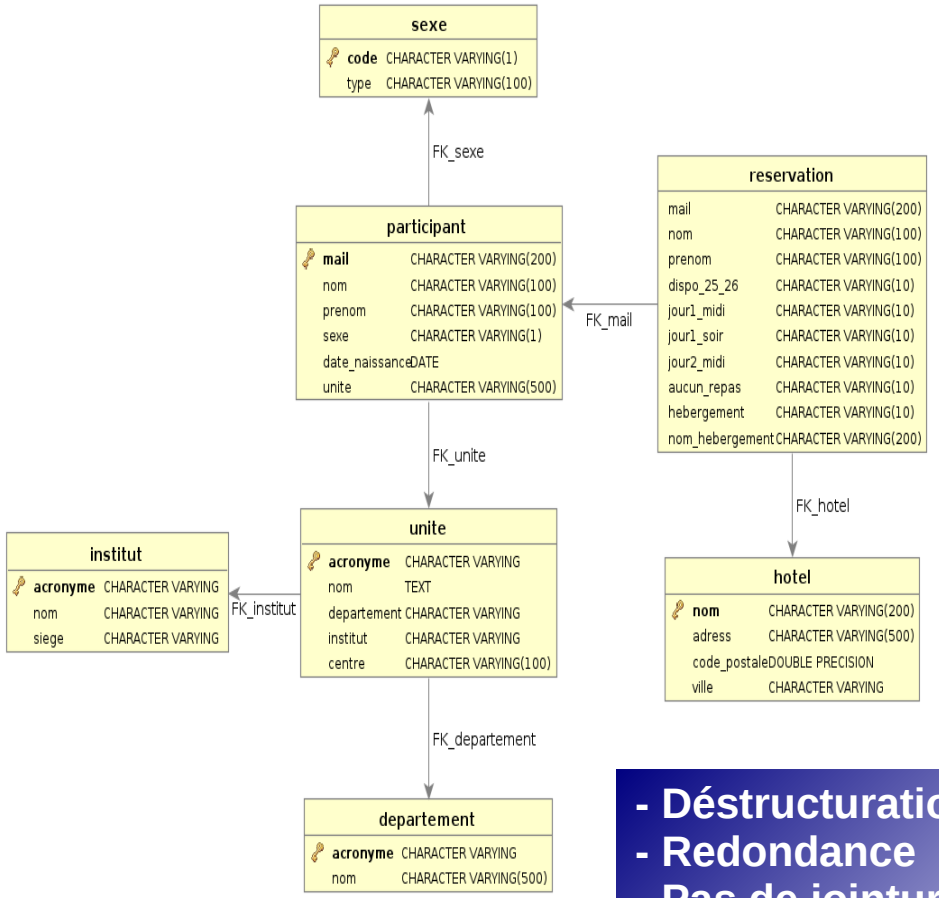
Collection **hotel**

Collection **departement**

Collection **unité**

MongoDb

Conception déstructurée



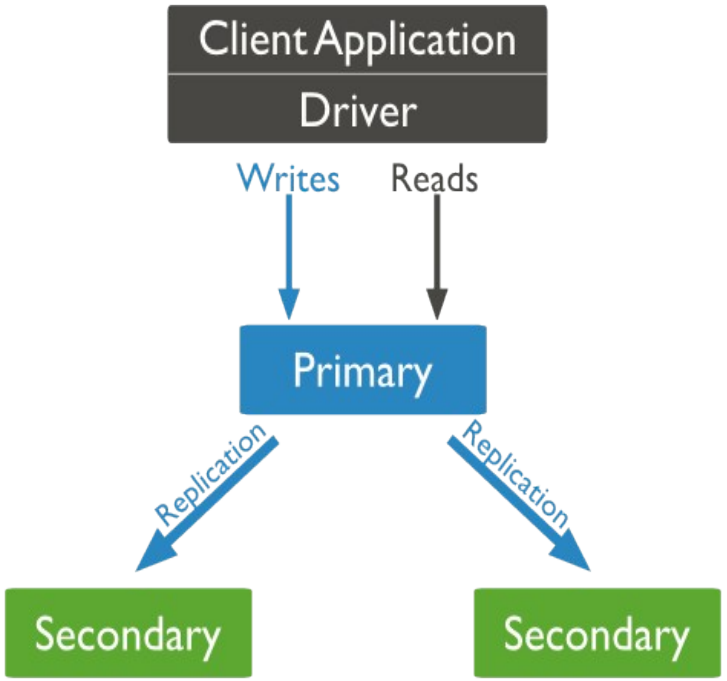
- Déstructuration du schéma SQL
- Redondance
- Pas de jointure

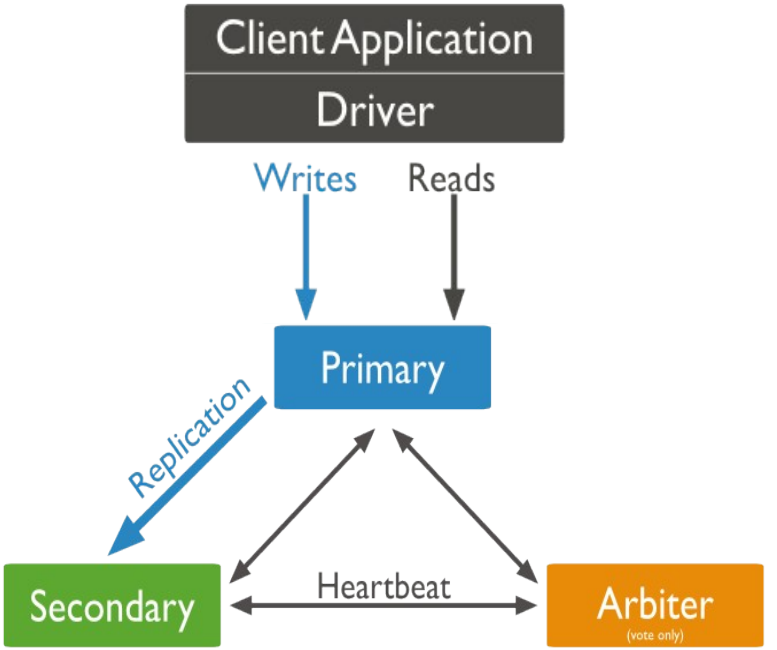
Collection réservation

```
{id : 0155fd51sdfçsdf545sdf,
participants : [{
        mail : vincent.negre@sgr.fr,
        nom : negre,
        prenom  : vincent,
        sexe : m,
        date_naissance : 1978/02/04,
        unité : [{
                nom : LEPSE
                code : 0759
                Département : EA
                }],
        }],
réservation : [{
        repas: negre,
        prenom  : vincent,
        sexe : m,
        date_naissance : 1978/02/04
        repas: [{
                repas_midi1 : 0,
                repas_midi2 : 0,
                repas_soir : 0,
                aucun_repas : N
        }],
        hotel: [{
                nom : Apartcity,
                code_postale: 11100
                ville : narbonne
        }],
}]
}
```

MongoDb

Architecture : Réplicat Set

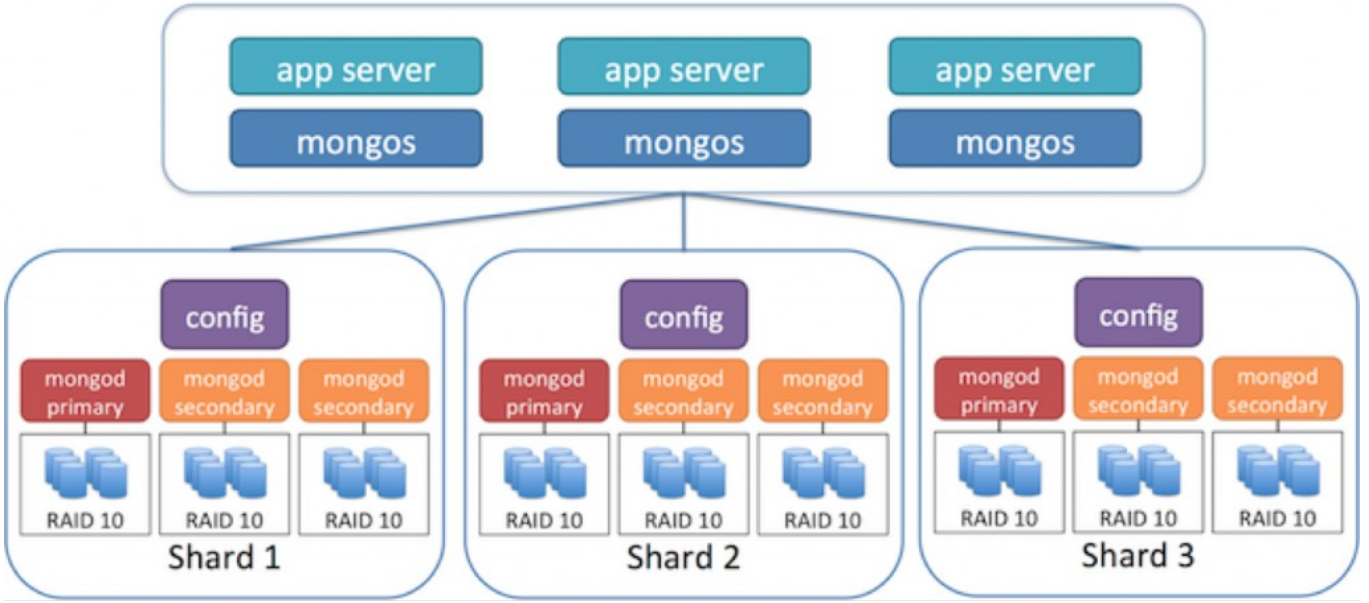




- Les serveurs sont séparés physiquement
- Le serveur arbitre n'est pas un réplicat
- N'empêche pas un dump de la base
- Basse volumetrie

MongoDb

Architecture : Sharding



- Les serveurs sont séparés physiquement
- Les routeurs ne sont pas des réplicats
- Haute volumetrie (>200Go)
- Forte connaissance en administration

MongoDb

Format BSON

Type	Number*	Notes
Double	1	
String	2	
Object	3	
Array	4	
Binary data	5	
Undefined	6	Deprecated.
Object id	7	
Boolean	8	
Date	9	
Null 1	0	
Regular Expression	11	
JavaScript	13	
Symbol	14	
JavaScript (with scope)	15	
32-bit integer	16	
Timestamp	17	
64-bit integer	18	

* : Each data type has a corresponding number that can be used with the **$type** operator to query documents by BSON type.

MongoDb

Premiers pas

- installation : http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/

- lancer le moteur de base de données : > sudo service mongod start
  > *waiting for connections on port 27017*

- lancer l'intrepréteur de commandes : > mongo

- créer / utiliser une base de données : > use pancakedb

MongoDb

CRUD : « select »

SQL SELECT Statements	MongoDB find() Statements
`SELECT *` `FROM users`	`db.users.find()`

`SELECT id,`
`user_id,`
`status`
`FROM users`
	`db.users.find(`
	`{ },`
	`{ user_id: 1, status: 1 }`
	`)`

`SELECT user_id, status`
`FROM users`
	`db.users.find(`
	`{ },`
	`{ user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *`
`FROM users`
`WHERE status = "A"`
	`db.users.find(`
	`{ status: "A" }`
	`)`

`SELECT user_id, status`
`FROM users`
`WHERE status = "A"`
	`db.users.find(`
	`{ status: "A" },`
	`{ user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *`
`FROM users`
`WHERE status != "A"`
	`db.users.find(`
	`{ status: { $ne: "A" } }`
	`)`

`SELECT *` `FROM users`	`db.users.find(` `{ status: "A",`

MongoDb

CRUD : «insert»

SQL INSERT Statements	MongoDB insert() Statements

`INSERT INTO users(user_id,`
`age,`
`status)`
`VALUES ("bcd001",`
`45,`
`"A")`
	`db.users.insert(`
	`{ user_id: "bcd001", age: 45, status: "A" }`
	`)`

MongoDb

CRUD : «update»

SQL Update Statements	MongoDB update() Statements

`UPDATE users`
`SET status = "C"`
`WHERE age > 25`
	`db.users.update(`
	`{ age: { $gt: 25 } },`
	`{ $set: { status: "C" } },`
	`{ multi: true }`
	`)`

`UPDATE users`
`SET age = age + 3`
`WHERE status = "A"`
	`db.users.update(`
	`{ status: "A" } ,`
	`{ $inc: { age: 3 } },`
	`{ multi: true }`
	`)`

MongoDb

CRUD : «delete»

SQL Delete Statements	MongoDB remove() Statements
`DELETE FROM users` `WHERE status = "D"`	`db.users.remove( { status: "D" } )`
`DELETE FROM users`	`db.users.remove({})`

MongoDb

Les opérateurs

Comparison

For comparison of different BSON type values, see the specified BSON comparison order.

Name	Description
`$gt`	Matches values that are greater than the value specified in the query.
`$gte`	Matches values that are greater than or equal to the value specified in the query.
`$in`	Matches any of the values that exist in an array specified in the query.
`$lt`	Matches values that are less than the value specified in the query.
`$lte`	Matches values that are less than or equal to the value specified in the query.
`$ne`	Matches all values that are not equal to the value specified in the query.
`$nin`	Matches values that **do not** exist in an array specified to the query.

Logical

Name	Description
`$and`	Joins query clauses with a logical `AND` returns all documents that match the conditions of both clauses.
`$nor`	Joins query clauses with a logical `NOR` returns all documents that fail to match both clauses.
`$not`	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
`$or`	Joins query clauses with a logical `OR` returns all documents that match the conditions of either clause.

MongoDb

Les opérateurs

Element

Name	Description
`$exists`	Matches documents that have the specified field.
`$type`	Selects documents if a field is of the specified type.

Evaluation

Name	Description
`$mod`	Performs a modulo operation on the value of a field and selects documents with a specified result.
`$regex`	Selects documents where values match a specified regular expression.
`$text`	Performs text search.
`$where`	Matches documents that satisfy a JavaScript expression.

MongoDb

Les opérateurs

Geospatial

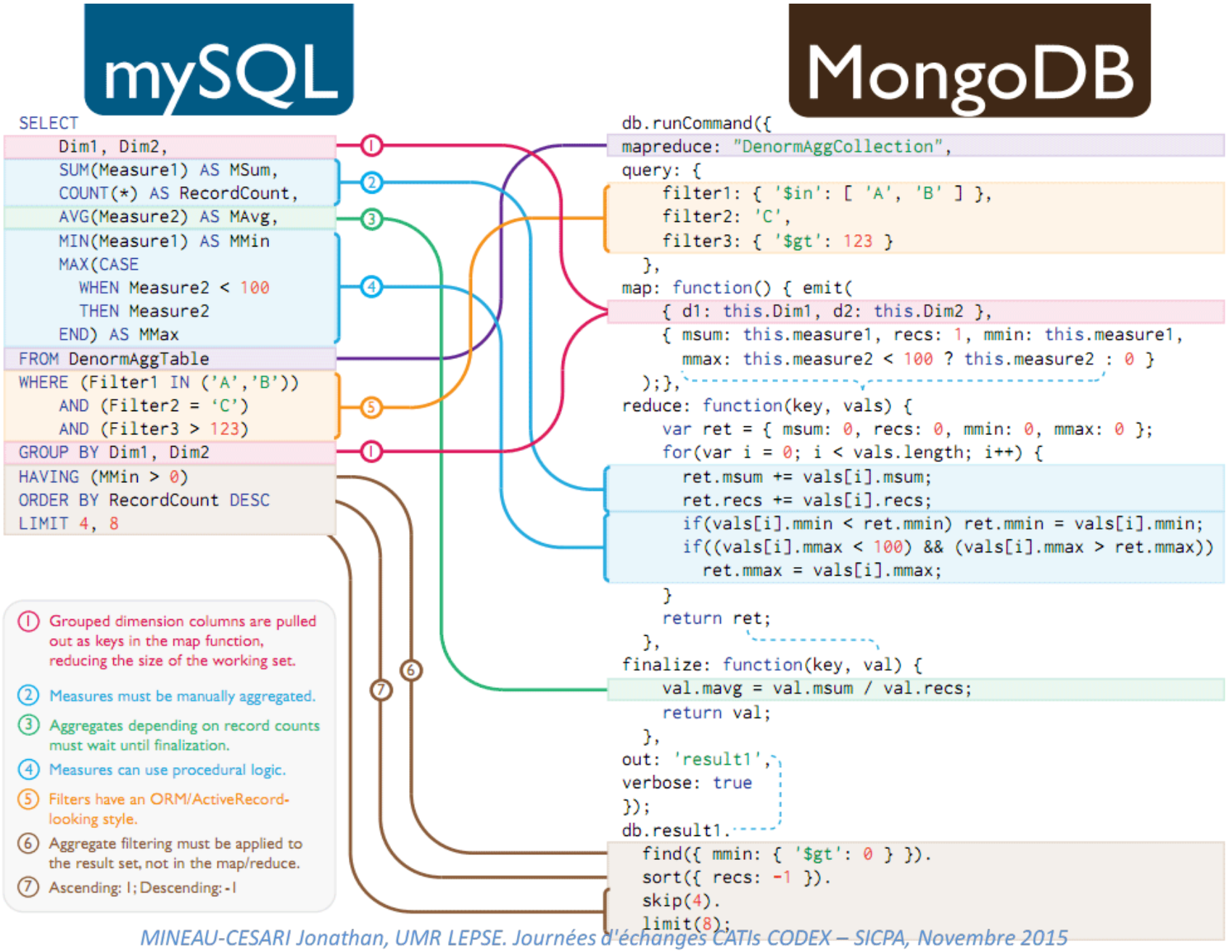
Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The 2dsphere index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The 2dsphere and 2d indexes support `$geoWithin`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The 2dsphere and 2d indexes support `$nearSphere`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The 2dsphere and 2d indexes support `$near`.

Array

Name	Description
`$all`	Matches arrays that contain all elements specified in the query.
`$elemMatch`	Selects documents if element in the array field matches all the specified `$elemMatch` conditions.
`$size`	Selects documents if the array field is a specified size.

MongoDb

Requête



MongoDb

Insertion des données

- En ligne de commande avec l'interpreteur Mongo avec la commande insert :

  > **db.acteurs.insert({nom:"Johansson", prenom:"Scarlett"})**

- En ligne de commande avec l'interpreteur Mongo avec JS :

  >

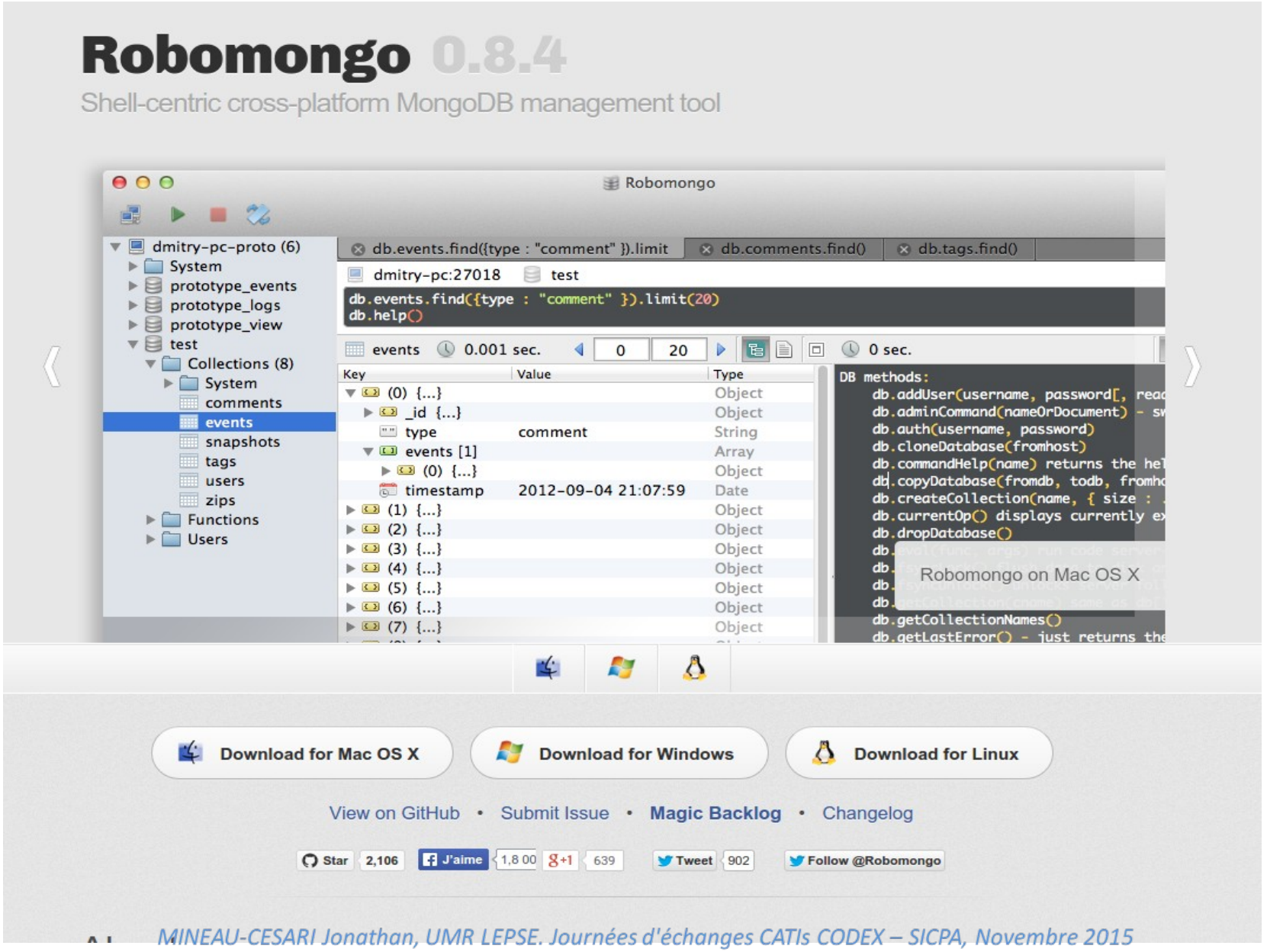
```
for(i=1; i<=100000; i++){
   var dixmill = i%10000;
   var mill = i%1000;
   var cent = i%100;
   db.produits.insert({
      compteur:i,
      dixmill:dixmill,
      mill:mill, cent:cent
      }
   )
}
```

- En ligne de commande Shell avec fichier CSV :

  > **mongoimport --db maBd --collection maCollection --type csv --file monFichier --headerline**

MongoDb

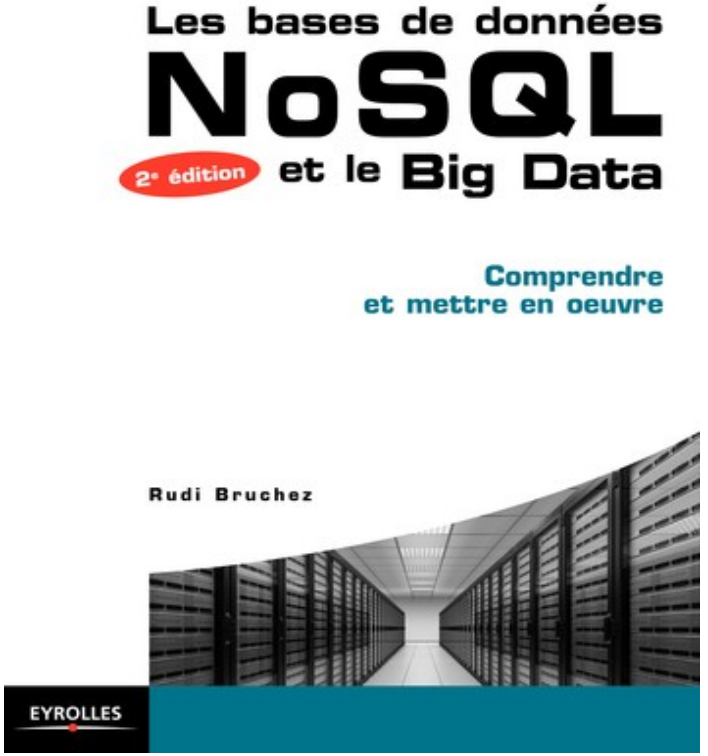
Client Mongo



mongoDB

NoSql

Bibliographie



mongoDB

NoSql
Bibliographie

Répartition de la charge dans Mongo DB

Que vous soyez développeur d'une application avec un fort volume de données, ou que vous souhaitiez vous lancer dans l'aventure d'une telle application, vous êtes certainement très intéressés par le NoSQL, mais vous avez également quelques craintes et c'est parfaitement légitime. La possibilité de rajouter des serveurs pour augmenter la puissance de la base est séduisante, mais le coût de la ferme de serveurs peut être rédhibitoire pour une application en début de vie.

Cet article explique :

- Dans ce qui suit, nous présentons une architecture minimale MongoDB et une stratégie simple pour garantir la sécurité de vos données et la performance de votre application.

Ce qu'il faut savoir :

- Cet article s'adresse aux développeurs, aux architectes et aux managers désireux d'en savoir plus sur une méthode de déploiement de MongoDB qui a fait ses preuves. Les principaux concepts d'architecture de MongoDB sont présentés en introduction pour les néophytes.

Présentation de MongoDB

MongoDB est un système de gestion de base de données NoSQL Open Source orienté document. Une base MongoDB est composée d'un ensemble de collections contenant des documents. Les collections sont à MongoDB ce que les *tables* représentent dans une base de données relationnelle classique et les documents correspondent aux *enregistrements*. Ce système permet de manipuler des données structurées et non-typées. Aucun schéma n'est imposé, les documents étant stockés au format BSON (JSON Binaire). Les documents d'une même collection pourront être structurellement différents les uns des autres et document pourra contenir lui-même d'autres documents. MongoDB est un système de gestion de base de données très souple et, de ce fait, très performant : les données habituellement obtenues par le biais de *jointures* sur plusieurs tables peuvent être directement embarquées dans un même document comme si les jointures avait déjà été exécutées.

Dans cet article, nous allons décrire les grands principes de Mongo DB et insister particulièrement sur trois d'entre eux :

- L'indexation, qui permet d'effectuer des requêtes rapides sur les collections
- Le sharding, qui permet de répartir une collection sur plusieurs serveurs
- Les Replica Set, qui dupliquent les données pour en garantir la disponibilité et l'intégrité

Contrairement à une base de données relationnelle avec un modèle normalisé, qui permet d'éviter les redondances et les dépendances incohérentes entre les tables,

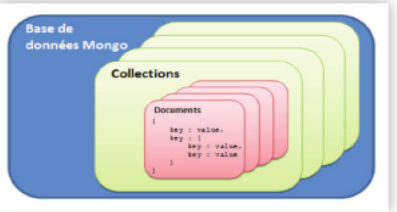


Figure 1. *Base de données Mango*

MongoDb

Démo !