

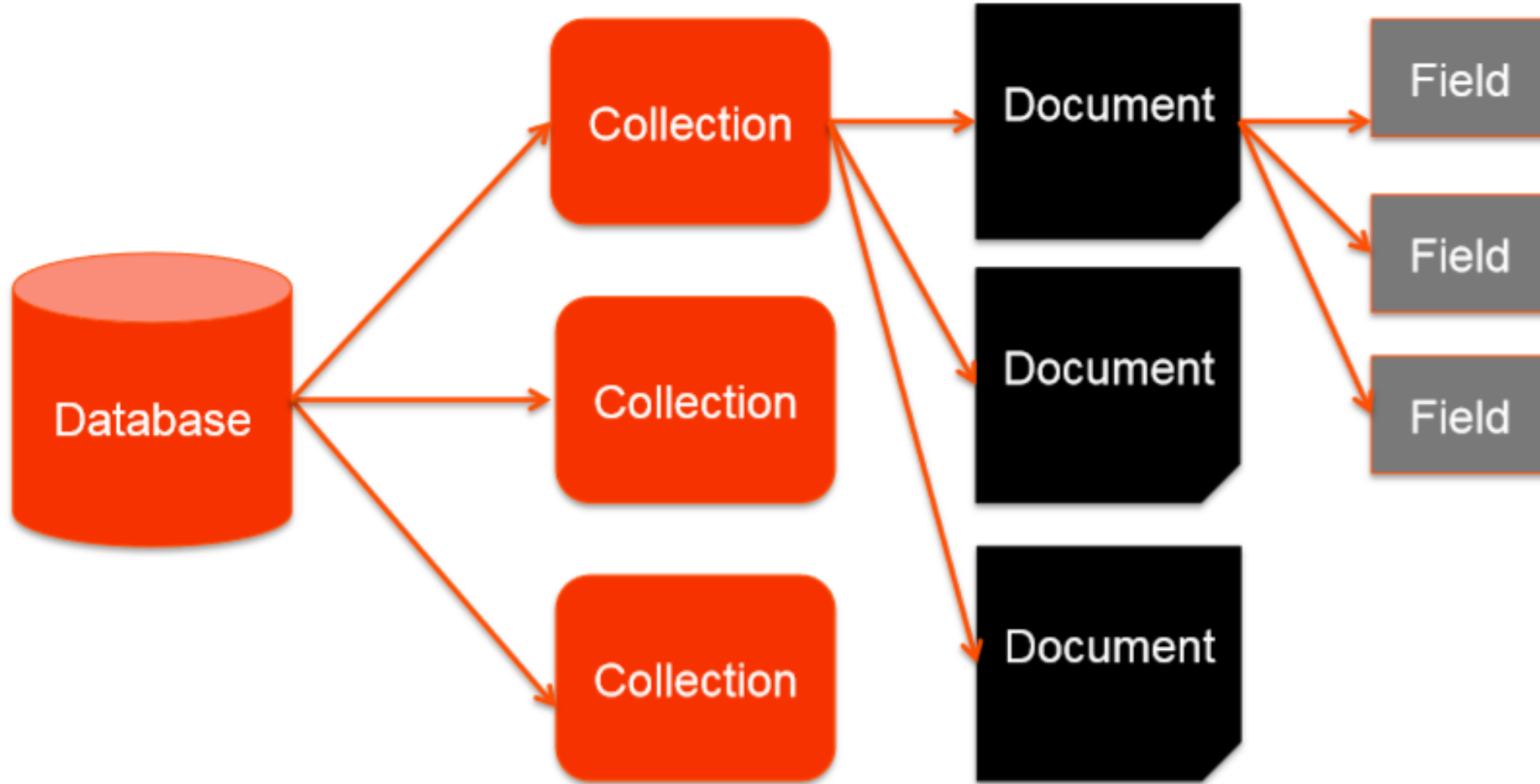
# Modèle Document

- **Principes**
- **MongoDB**

# Base de données documents

- Basé également sur le paradigme [clé, valeur], mais avec une valeur, de type document
- Document : structure arborescente contenant une liste de champs
- Champs est associé à une valeur qui peut elle même être une liste
- Documents principalement de type JSON ou XML
- Représentation d'un document :
  - Forme sérialisée (la plus courante) : contenu marqué par des balises ou par des accolades ouvrante/fermante.
  - Forme arborescente

# Collection de documents

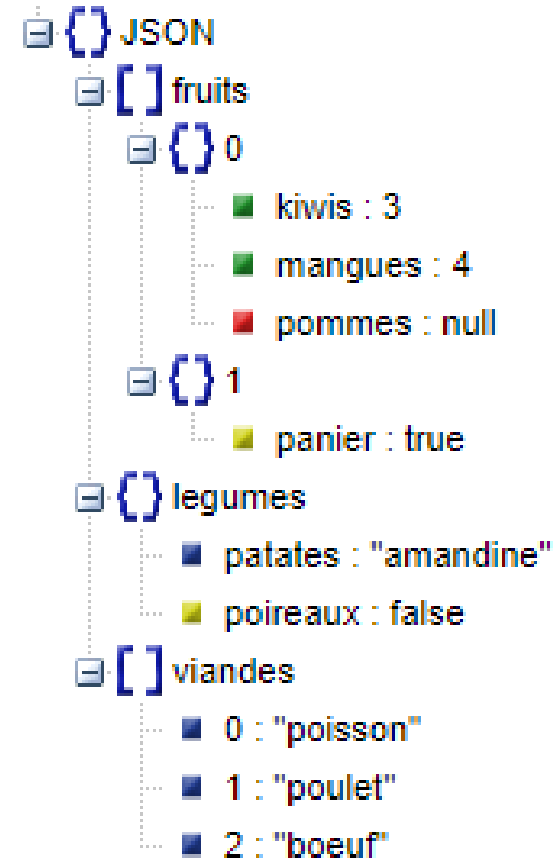


# Document JSON

## Forme sérialisée

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true }
  ],
  "legumes": {
    "patates": "amandine",
    "poireaux": false
  },
  "viandes": ["poisson", "poulet", "boeuf"]
}
```

## Forme arborescente



# Base de données documents : cas d'utilisation

- **Cas d'utilisation :**

- Systèmes de gestion de contenu (profils utilisateurs, avis et commentaires, blogs, etc.)
- Applications de e-commerce (stockage du catalogue, schéma flexible pour les produits et commandes)



























- **Non applicable aux:**

- Données interconnectées
- Données non-structurées

# Principaux moteurs orientés document

☐ include secondary database models

53 systems in ranking, February 2022

Rank			DBMS	Database Model	Score		
Feb 2022	Jan 2022	Feb 2021			Feb 2022	Jan 2022	Feb 2021
1.	1.	1.	MongoDB 	Document, Multi-model 	488.64	+0.07	+29.69
2.	2.	2.	Amazon DynamoDB 	Multi-model 	80.36	+0.50	+11.21
3.	3.	3.	Microsoft Azure Cosmos DB 	Multi-model 	39.95	-0.09	+8.29
4.	4.	4.	Couchbase 	Document, Multi-model 	30.07	+1.21	-0.59
5.	5.	5.	Firebase Realtime Database	Document	19.15	-0.21	+3.15
6.	6.	6.	CouchDB	Document, Multi-model 	17.45	+0.81	+1.84
7.	 8.	7.	MarkLogic	Multi-model 	9.46	+0.27	+0.00
8.	 7.	8.	Realm 	Document	9.42	-0.17	+0.27
9.	9.	9.	Google Cloud Firestore	Document	9.06	+0.14	+1.01
10.	 12.	 11.	ArangoDB 	Multi-model 	5.40	+0.67	+0.33
11.	 10.	 20.	Virtuoso 	Multi-model 	5.39	+0.02	+3.02
12.	 11.	12.	Google Cloud Datastore	Document	5.25	+0.41	+0.23
13.	13.	 10.	OrientDB	Multi-model 	5.03	+0.47	-0.10
14.	14.	 13.	Oracle NoSQL	Multi-model 	4.84	+0.43	+0.33

# MongoDB

- Base de données orientée documents (représentation structurée)
  - Open source développé par MongoDB Inc depuis 2007
  - Nom provenant de l'anglais "humongous" ("énorme")
  - Ecrit en C++
  - *Mongo Atlas* : mode hébergé dans le cloud sur la base d'une tarification horaire
  - Stockage des documents au format BSON (*Binary Serialized dOcument Notation* ou *Binary JSON*) – mais structure visible par les utilisateurs en JSON
-

# MongoDB : liens utiles

- Site officiel : <https://www.mongodb.com/fr>
- Documentation : : <https://docs.mongodb.com/manual/introduction/>
- Console en ligne dans la documentation :  
<https://docs.mongodb.com/manual/tutorial/insert-documents/>
- Tutoriels :
  - <https://docs.mongodb.com/manual/tutorial/>
  - <http://b3d.bdpedia.fr/docstruct.html#s4-mongodb-une-base-json>
  - <https://stph.scenari-community.org/contribs/nos/Mongo1/co/presentation.html>
  - [https://www.tutorialspoint.com/mongodb/mongodb\\_overview.htm](https://www.tutorialspoint.com/mongodb/mongodb_overview.htm)



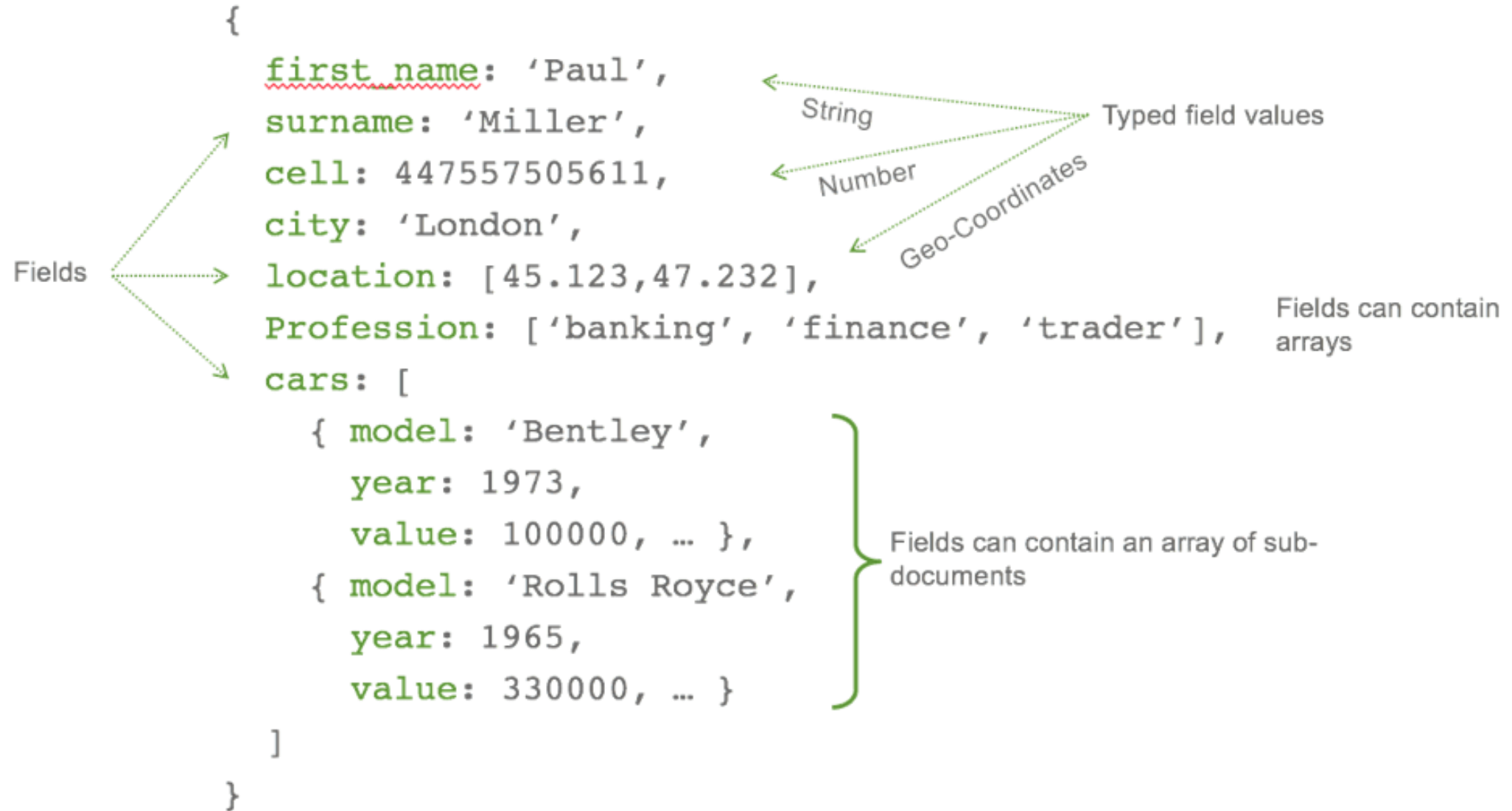
# MongoDB : modèle de données (1/2)

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key <code>_id</code> provided by mongodb itself)

# MongoDB : modèle de données (2/2)

- Document BSON = unité de stockage (~ une ligne dans une BDR)
- Pour l'utilisateur : structure visible en JSON
- Tout document appartient à une collection et a un champ appelé `_id` qui identifie le document dans la base de données
- Document : hiérarchie de paires clé-valeur, sans contrainte de présence ou de quantité
- Collection : ensemble de documents (~table en relationnel)

# MongoDB : exemple



# MongoDB : attribut identificateur `_id`

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

# MongoDB : relationnel vs document

## Relationnel

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	White	Dallas
3	Meagan	White	London
4	Edward	Daniels	Boston

Phone Number	Type	DNC	Customer ID
1-212-555-1212	home	T	0
1-212-555-1213	home	T	0
1-212-555-1214	cell	F	0
1-212-777-1212	home	T	1
1-212-777-1213	cell	(null)	1
1-212-888-1212	home	F	2



## MongoDB

```
{  customer_id : 1,
   name : {
     "f": "Mark",
     "l": "Smith"  },
   city : "San Francisco",
   phones: [ {
     number : "1-212-777-1212",
     dnc : true,
     type : "home"
   },
   {
     number : "1-212-777-1213",
     type : "cell"
   }
 ]}
```

# MongoDB : Relation implémentée par une imbrication de documents

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address": [
    {
      "building": "22 A, Indiana Apt",
      "pincode": 123456,
      "city": "Los Angeles",
      "state": "California"
    },
    {
      "building": "170 A, Acropolis Apt",
      "pincode": 456789,
      "city": "Chicago",
      "state": "Illinois"
    }
  ]
}
```

# MongoDB : Relation implémentée par l'utilisation de référence

```
{
  "_id":ObjectId("52ffc4a5d85242602e000000"),
  "building": "22 A, Indiana Apt",
  "pincode": 123456,
  "city": "Los Angeles",
  "state": "California"
}
```

```
{
  "_id":ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address_ids": [
    ObjectId("52ffc4a5d85242602e000000"),
    ObjectId("52ffc4a5d85242602e000001")
  ]
}
```

# MongoDB : principes

- En général, regroupement de toutes les données relatives à un objet dans un même « document »
- ⇒ nombre de jointures réduit et donc impact positif sur les performances (toutes les données sont récupérées en une seule lecture)
- Document proche de la structure des objets dans les langages de programmation (facilite le développement)
  - Modèle des documents pouvant varier de structure dans une même collection
  - Schéma dynamique : possibilité d'ajouter de nouveaux champs sans affecter les autres documents
  - Schéma flexible mais conçu selon le type de requêtes



# MongoDB : opérations CRUD (insertion)

```
db.users.insertOne(  
  {  
    name: "sue",  
    age: 26,  
    status: "pending"  
  }  
)
```

← collection

← field: value

← field: value

← field: value } document

# MongoDB : Console et un tutoriel interactif

```
db.inventory.insertOne(  
  { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }  
)
```

copy

You can run the operation in the web shell below:

```
Click to connect  
Connecting...  
MongoDB shell version v3.6.0  
connecting to: mongodb://127.0.0.1:27017  
MongoDB server version: 3.6.0  
type "help" for help  
>>> db.inventory.insertOne(  
...   { item: "canvas", qty: 100, tags: ["cotton"], size: { h:  
28, w: 35.5, uom: "cm" } }  
... )  
{  
  "acknowledged" : true,  
  "insertedId" : ObjectId("5c66db7c32ab66414816fcd7")  
}  
>>>
```

Full  
Reset  
Clear

# MongoDB : opérations CRUD

## (initialisation ou non de l'attribut `_id`)

- Soit MongoDB crée le champ `_id` et lui affecte une valeur `ObjectId` unique

```
db.products.insert( { item: "card", qty: 15 } )
```

```
{ "_id" : ObjectId("5063114bd386d8fadbd6b004"), "item" : "card", "qty" : 15 }
```

- Soit on crée cet identificateur à l'insertion

```
db.products.insert( { _id: 10, item: "box", qty: 20 } )
```

```
{ "_id" : 10, "item" : "box", "qty" : 20 }
```

# MongoDB : opérations CRUD

## (mixage de la gestion des identificateurs)

```
db.products.insert(  
  [  
    { _id: 11, item: "pencil", qty: 50, type: "no.2" },  
    { item: "pen", qty: 20 },  
    { item: "eraser", qty: 25 }  
  ]  
)
```

```
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a0"), "item" : "pen", "qty" : 20 }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a1"), "item" : "eraser", "qty" : 25 }
```

# MongoDB : opérations CRUD (**find**)

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

← collection  
← query criteria  
← projection  
← cursor modifier

# MongoDB : traduction de `find` en SQL (1/2)

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

```
SELECT * FROM inventory
```

```
db.inventory.find( { status: "D" } )
```

```
SELECT * FROM inventory WHERE status = "D"
```

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

```
SELECT * FROM inventory WHERE status in ("A", "D")
```

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

# MongoDB : MongoDB : traduction de `find` en SQL (2/2)

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

```
SELECT _id, item, status from inventory WHERE status = "A"
```

```
db.inventory.find( { status: "A" }, { item: 1, status: 1, _id: 0 } )
```

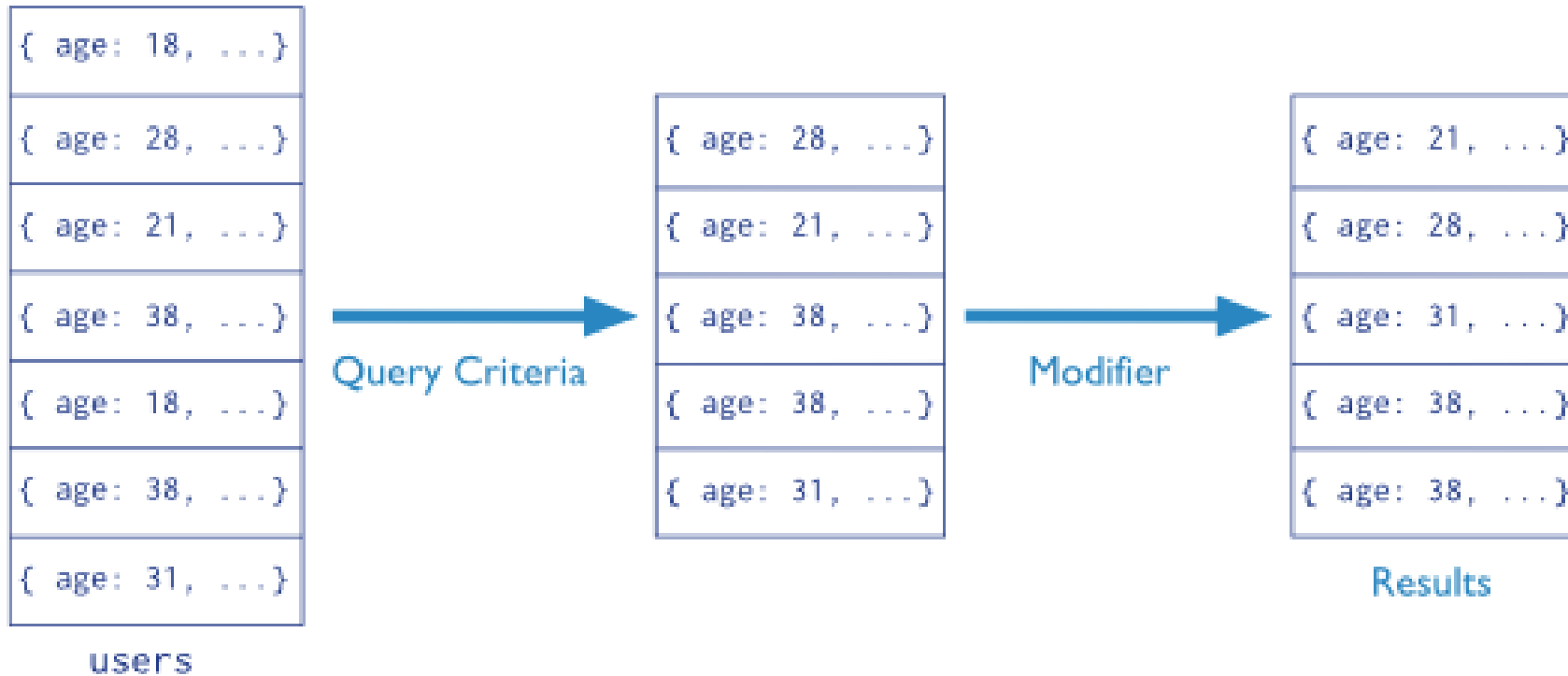
```
SELECT item, status from inventory WHERE status = "A"
```

**Attention : Pas d'erreur en cas de faute de frappe sur des noms d'attributs**

---

# MongoDB : exemple de tri

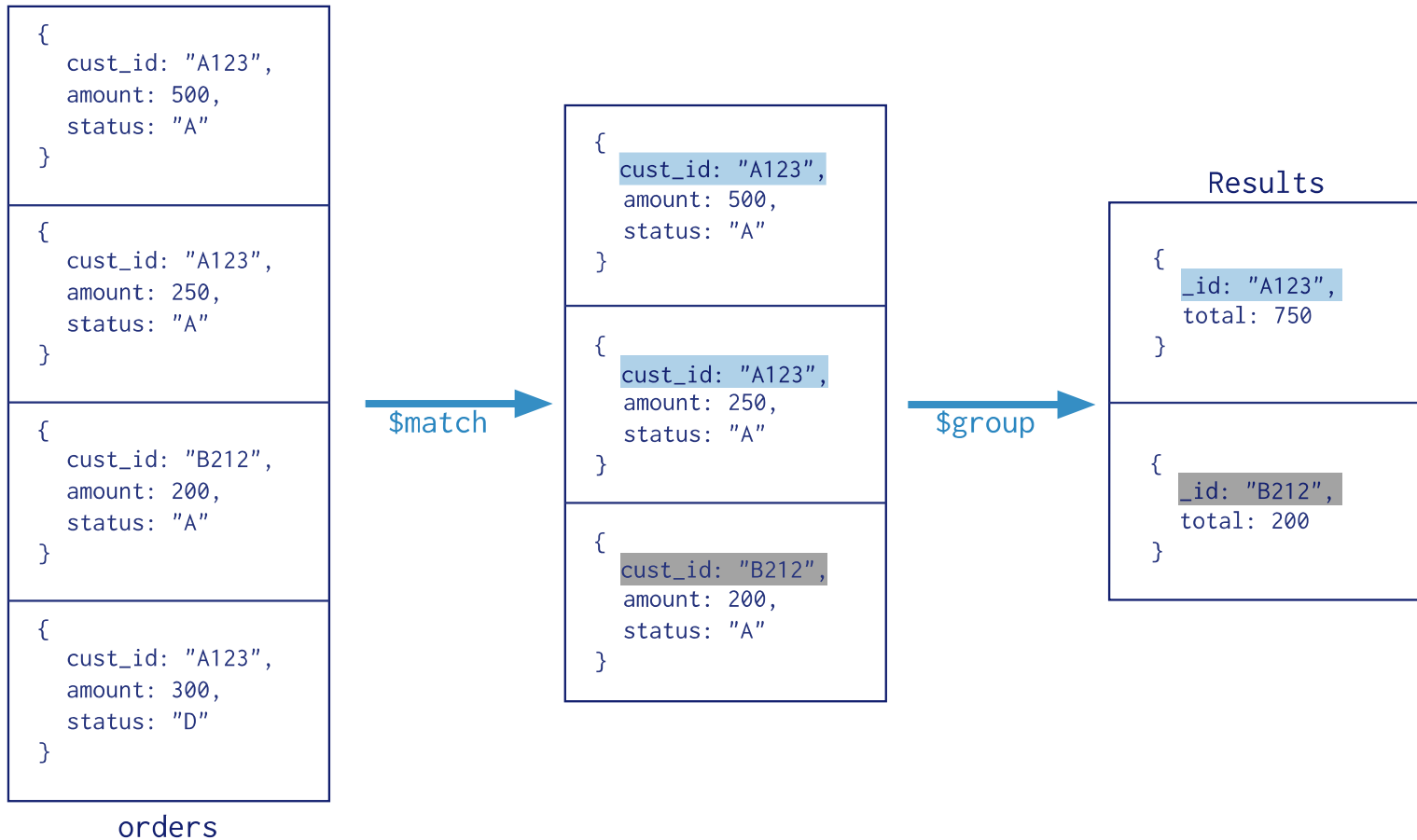
Collection                      Query Criteria                      Modifier  
`db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )`





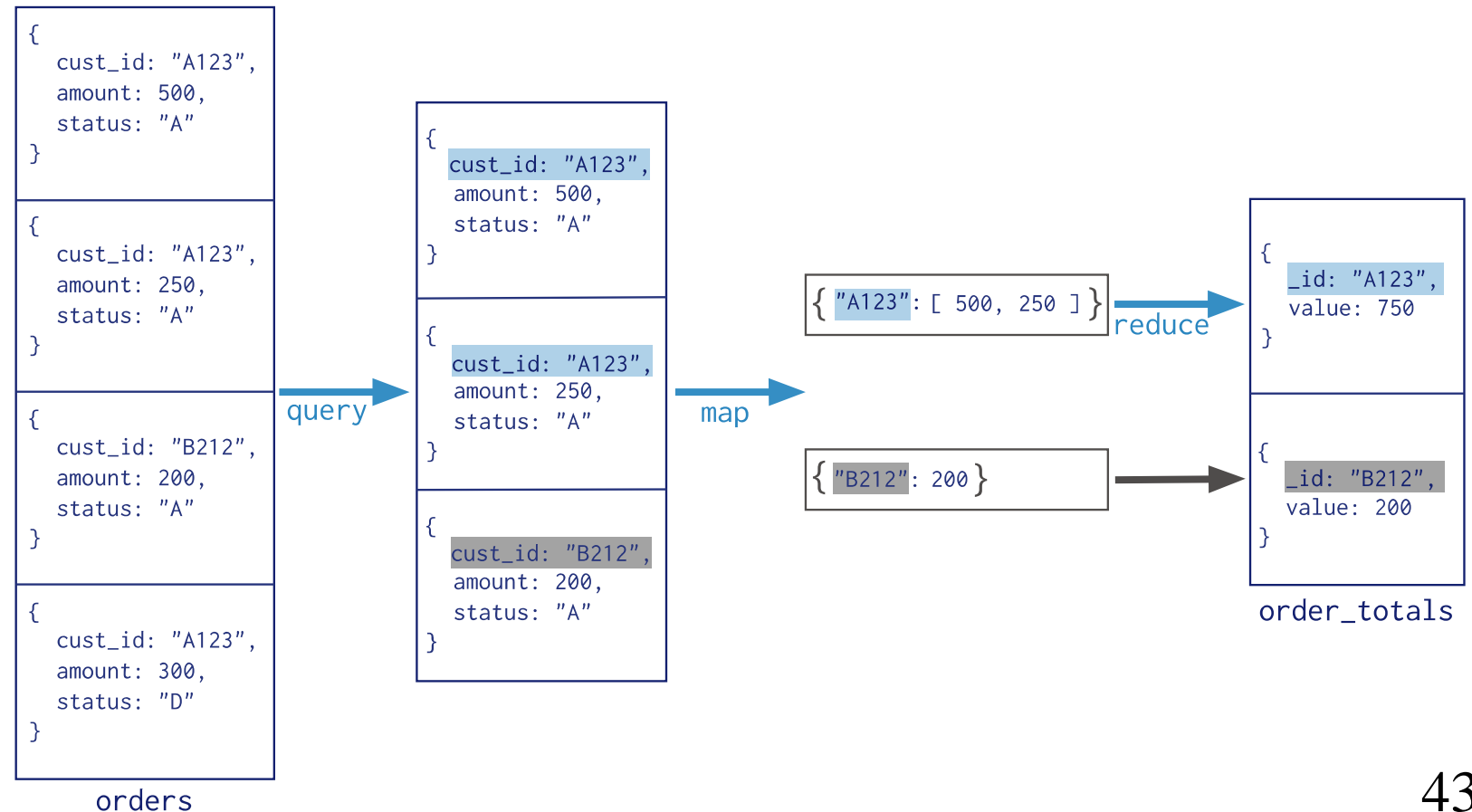
# MongoDB : agrégation

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



# MongoDB : agrégation en utilisant *Map Reduce*

Collection  
↓  
db.orders.mapReduce(  
    map     →   function() { emit( this.cust\_id, this.amount ); },  
    reduce  →   function(key, values) { return Array.sum( values ) },  
    query   →   {  
    output  →    query: { status: "A" },  
              out: "order\_totals"  
    }  
)



# MongoDB : opérations CRUD (update)

```
db.users.updateMany(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } }  
)
```

← collection  
← update filter  
← update action

---

# MongoDB : opérations CRUD (delete)

```
db.users.deleteMany(  
  { status: "reject" }  
)
```

← collection  
← delete filter

# MongoDB : possibilités de requêtage

- Possibilité de faire des requêtes d'agrégation et d'utiliser *map-reduce*
- Recherche textuelle
- Gestion de données spatiales
  - Données modélisées en GeoJSON (<http://geojson.org/>)

```
location: {  
  type: "Point",  
  coordinates: [-73.856077, 40.848447]  
}
```

- Requêtes spatiales (cf. <https://docs.mongodb.com/manual/geospatial-queries/>)

# MongoDB : atomicité

- Ecriture d'un document = opération atomique
  - Ecriture de plusieurs documents est atomique pour chacun des documents, mais pas pour l'ensemble
  - Ecritures atomiques  $\Rightarrow$  impact sur le design des documents et des collections
  - Nécessité de s'assurer de maintenir la cohérence des données
-

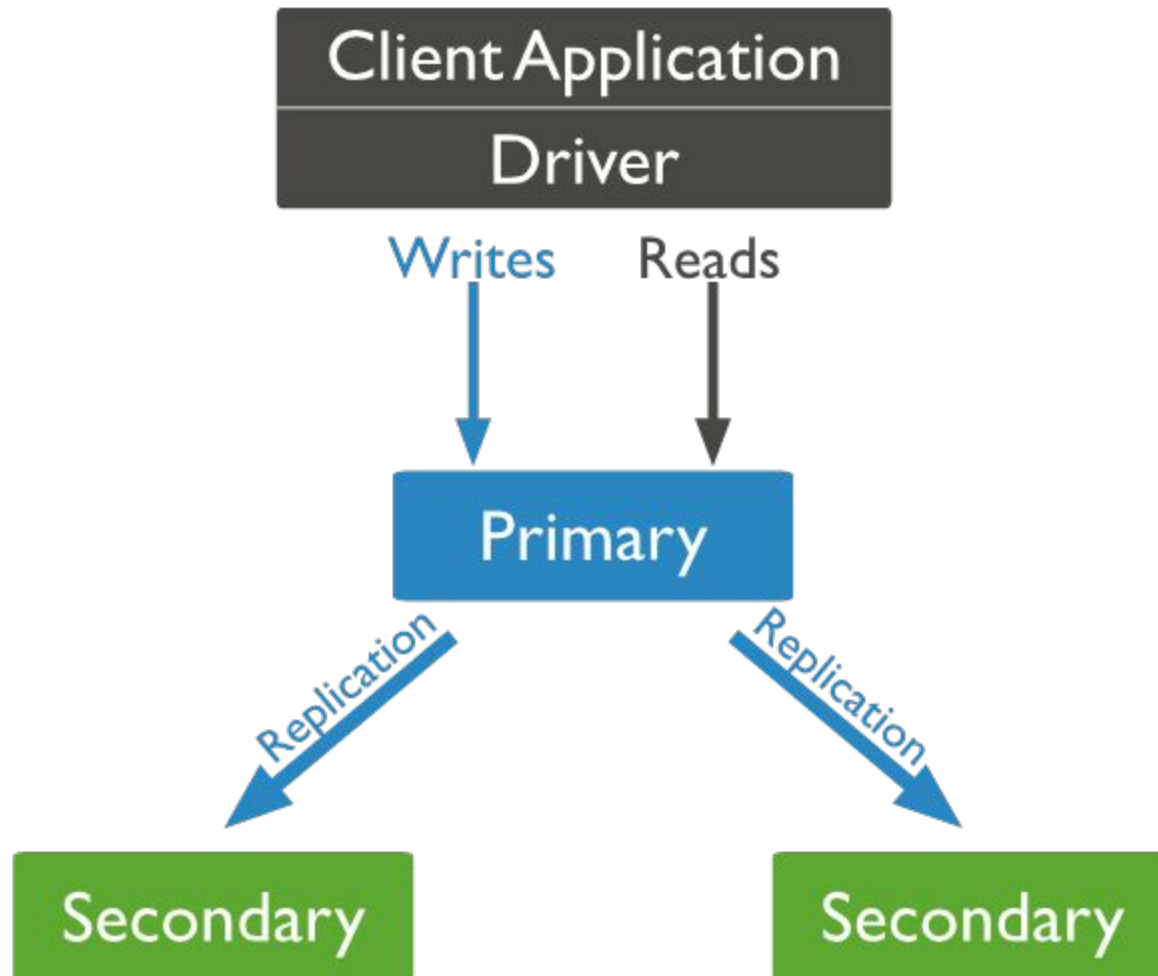
# Réplication des données

---

- **Replicat Set** : groupe de processus **mongod** qui fournit la redondance et la haute disponibilité
- Types de membres dans un replicat set:
  - Primaire**: Reçoit toutes les opérations d'écriture
  - Secondaire**: réplication des opérations du primaire pour maintenir des répliques identiques à l'original
  - Arbitre**: ne conserve pas de copie de la donnée, mais joue un rôle dans les élections qui sélectionnent un membre primaire, si le primaire actuel est indisponible

# Réplication des données

---





# Réplication des données

---

## ■ **Primaire :**

- Seul membre qui reçoit les opérations d'écriture
- MongoDB applique les opérations d'écriture sur le primaire, puis les enregistre dans son log (oplog)
- Les membres secondaires dupliquent ce log et appliquent les opérations sur leurs données
- Tous les membres d'un replicat set peuvent accepter une opération de lecture
- Par défaut, une application dirige ces opérations vers le primaire
- Un replicat set a au plus un primaire
- Si le primaire tombe en panne, une élection a lieu pour en choisir un autre

# Réplication des données

---

## ■ **Secondaire :**

- Maintiennent une copie des données du primaire
- Pour répliquer les données, un secondaire applique les opérations du oplog du primaire sur ses propres données, de manière asynchrone
- Un replicat set peut avoir un ou plusieurs secondaires
- Même si les clients ne peuvent pas écrire des données sur les secondaires, ils peuvent en lire
- Un secondaire peut devenir un primaire, suite à une élection
- Il est possible de configurer un secondaire :
  - L'empêcher d'être élu pour devenir primaire, lui permettant de rester toujours comme un backup
  - Empêcher les applications de lire à partir de lui, lui permettant ainsi de se consacrer à certaines applications nécessitant un trafic séparé

# Réplication des données

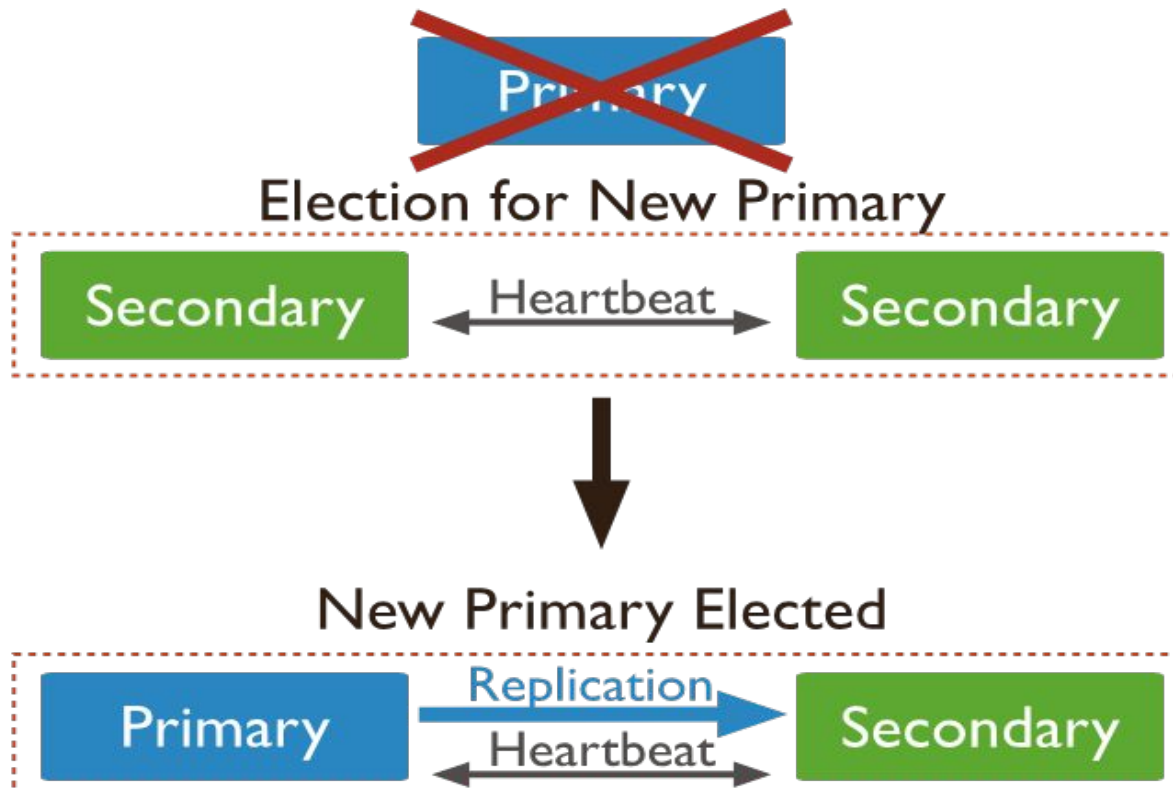
---

## ■ Élections :

- Ont lieu à la création d'un replicat set, ou bien quand un primaire devient indisponible
- Processus qui prend du temps, et qui rend le replicat set en readonly
- Chaque membre a une priorité déterminant son éligibilité à devenir primaire
- L'élection choisit le membre ayant la plus haute priorité comme primaire
- Par défaut, tous les membres ont la même priorité
- Il est possible de modifier la priorité d'un membre ou groupe, selon leur position géographique par exemple.
- Chaque membre a la possibilité de voter pour un seul autre membre
- Le membre recevant le plus grand nombre de votes devient primaire

# Réplication des données

- **Élections :**



# Distribution des données

---

**Le sharding** : une méthode de distribution des données sur plusieurs machines pour assurer la scalabilité de l'architecture :

- MongoDB utilise le sharding pour prendre en charge les déploiements avec de très grands ensembles de données et des opérations à haut débit.
- MongoDB prend en charge la **scalabilité horizontale** grâce au sharding.
- Division des données et leur distribution entre plusieurs serveurs ou shards
- Chaque shard est une base indépendante, et mis ensemble, forment une seule base de données logique
- Réduction du nombre d'opérations que chaque machine gère
- Chaque machine gère les données qui y sont stockées
- Réduction de la quantité de données que le serveur a besoin de stocker
- Plus le cluster grandit, moins un serveur contient de données

# Distribution des données

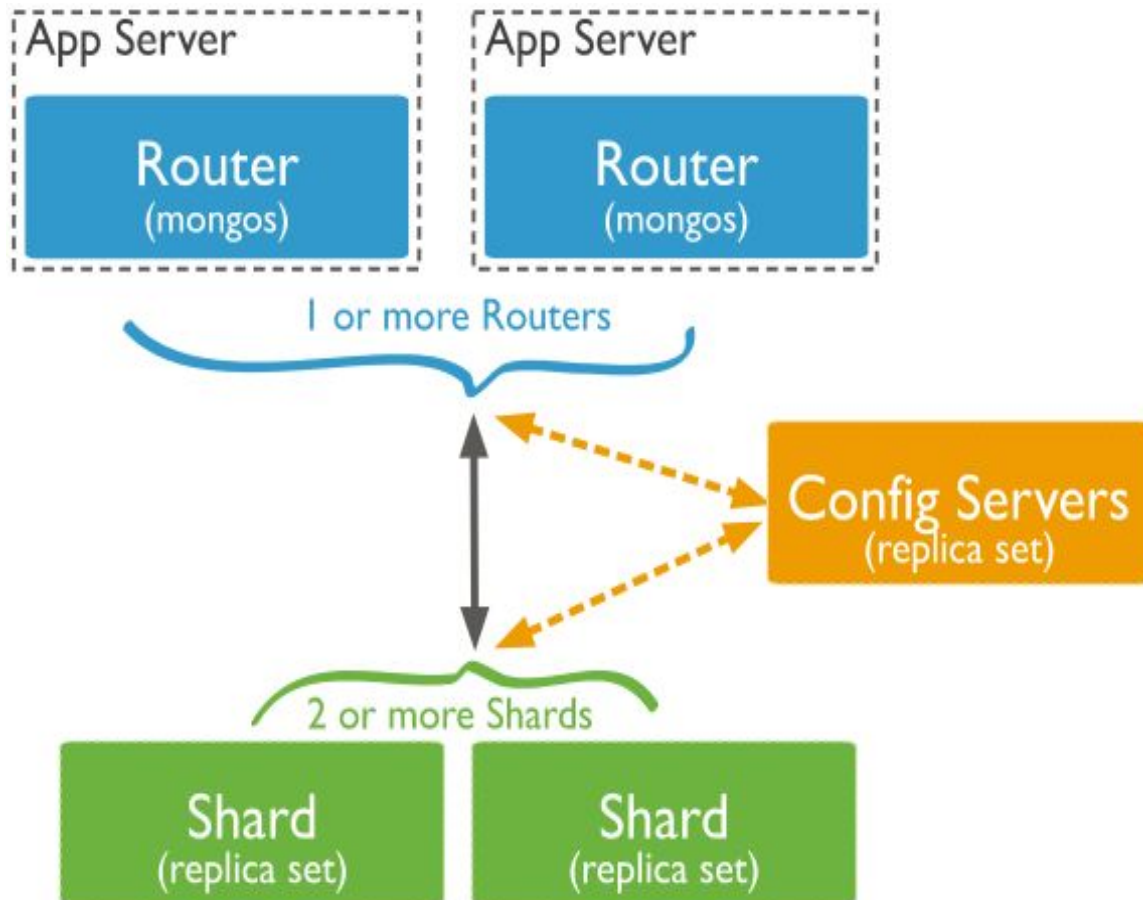
---

- Le sharding MongoDB consiste à créer un **Cluster** d'instances MongoDB composée d'au moins trois serveurs:
  - **Shard:**
    - déployés en tant *Replicat set*
    - Stockent les données
    - Les données sont distribuées et répliquées sur les shards
  - **Query Routers :**
    - Instances mongos
    - Interfaçage avec les applications clientes
    - Redirige les opérations vers le shard approprié et retourne le résultat au client
    - Plusieurs Routers pour la répartition des tâches.
  - **ConfigServers :** stockent les métadonnées et les paramètres de configuration pour l'ensemble du cluster.

# Distribution des données

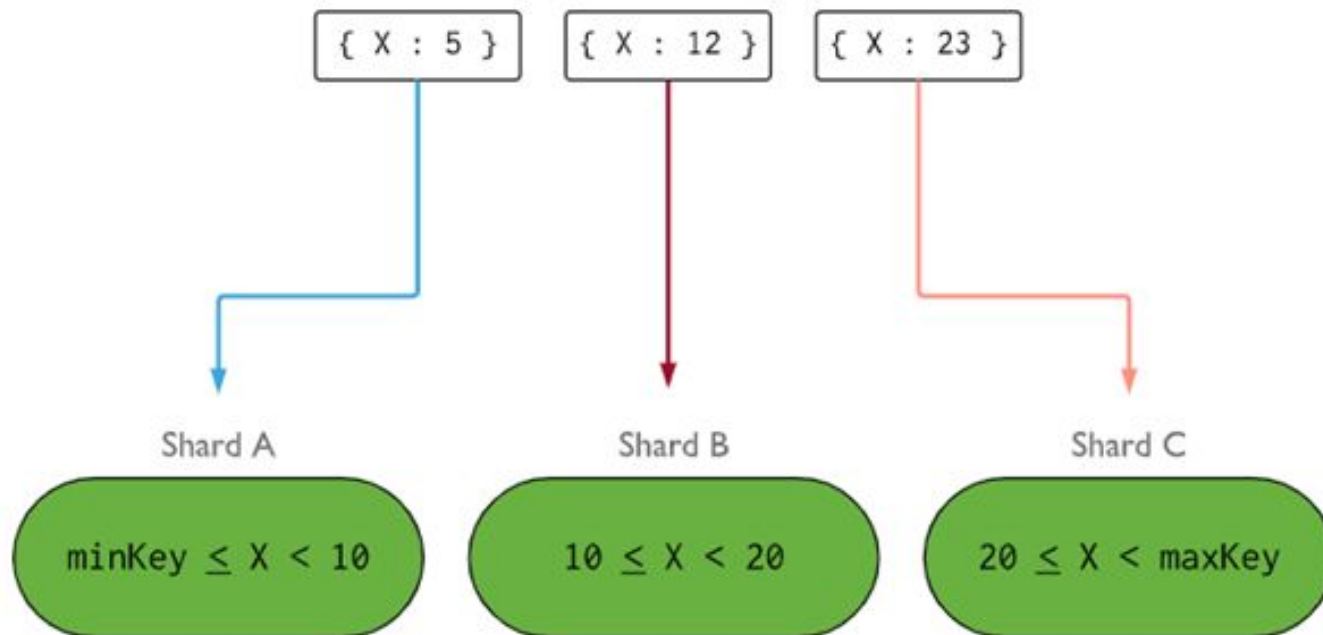
- L'architecture de distribution d'un cluster est schématisée dans la figure ci-dessous :

- 1) L'application cliente communique avec les routeurs (mongos) au sujet de la requête à exécuter.
- 2) L'instance mongos consulte les serveurs de configuration pour vérifier quel Shard contient l'ensemble de données requis et envoyer la requête à ce Shard
- 3) Le résultat de la requête sera renvoyé à l'application.



# Fragment de données

- **fragment de données (Chunk):** sont des sous-ensembles de documents. MongoDB sépare les documents d'une collection en fragments (chunks) qui sont distribués sur l'ensemble des shards du cluster.

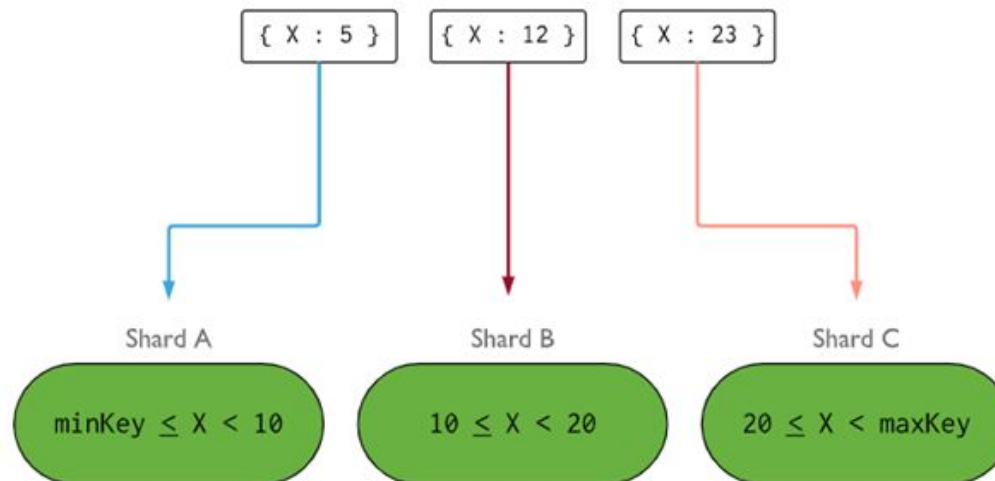




# Clé de partitionnement

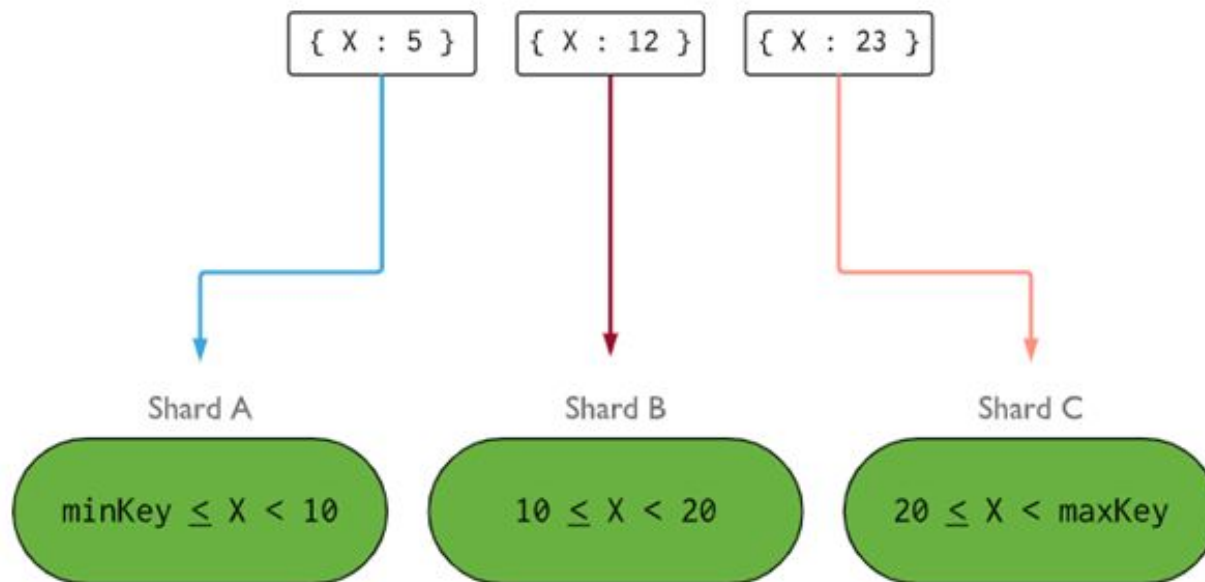
**Clé de partitionnement/shard key:** Un partitionnement s'effectue toujours en fonction d'une *clé*. MongoDB utilise une clé de partitionnement pour répartir les documents de la collection entre les shards.

- La clé de partitionnement est constituée d'un ou de plusieurs champs des documents.
- Idéalement, on choisira comme clé de partitionnement l'identifiant unique des documents



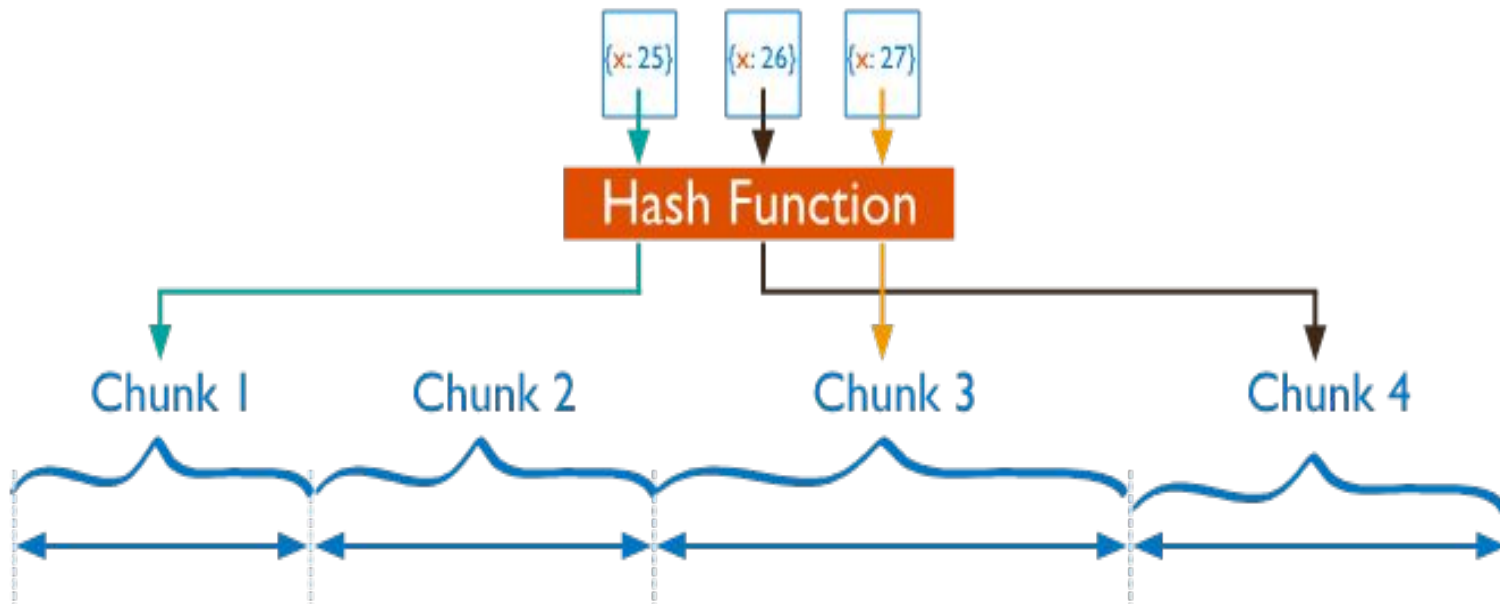
# Stratégie de Sharding

- **Ranged Sharding:** Le domaine de la clé est partitionné en intervalles semi-ouverts. Chaque intervalle est associé à un fragment stocké sur un shard.



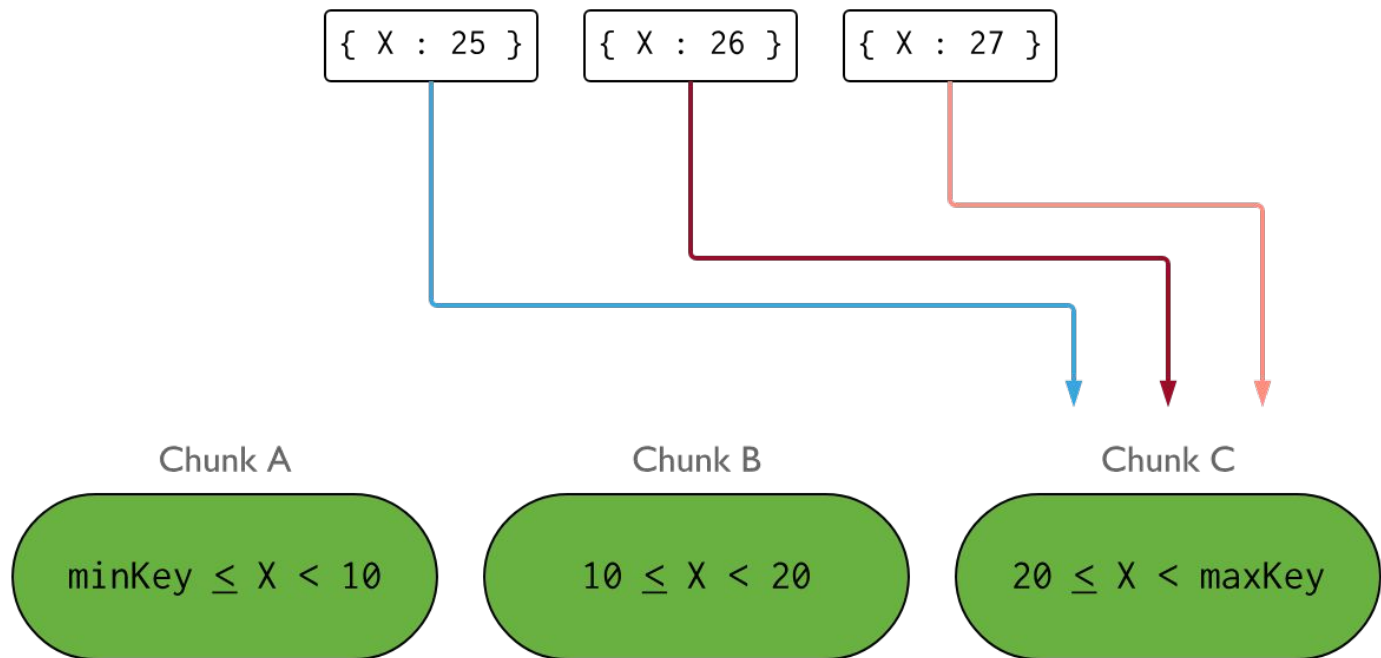
# Stratégie de Sharding

- **Hashed Sharding:** une fonction de hachage est appliquée à la clé de partitionnement qui détermine le fragment d'affectation.



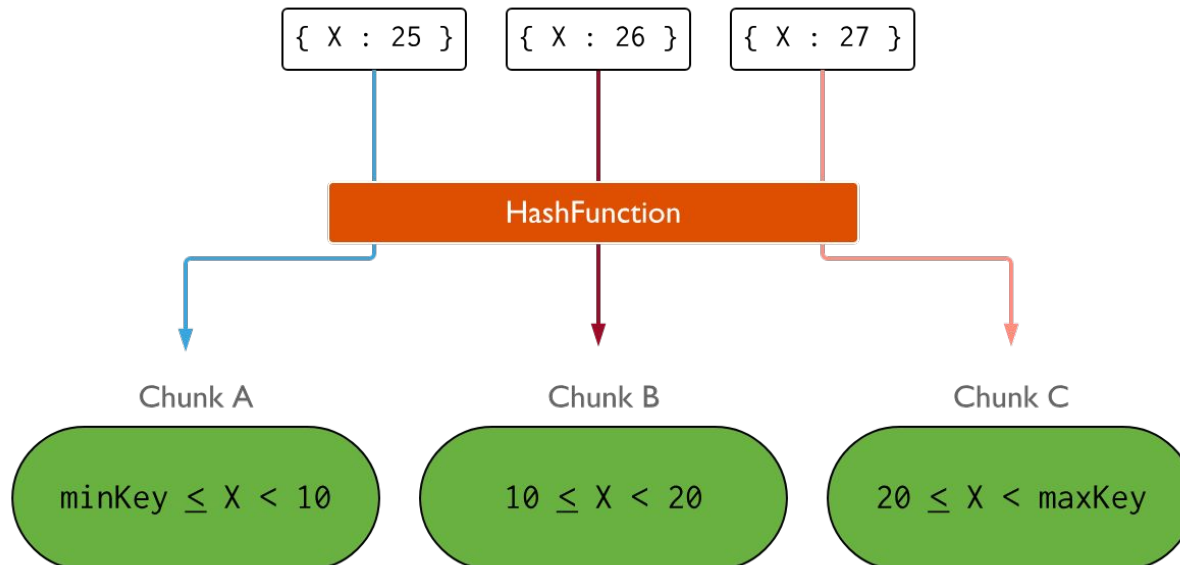
# Ranged sharding Vs Hashed Sharding

- **Hashed Sharding:** Étant donné une collection utilisant une valeur croissante de **X** comme clé de partitionnement, la distribution des insertions entrantes en utilisant ranged sharding est similaire à ce qui suit:



# Ranged sharding Vs Hashed Sharding

- **Hashed Sharding:** Étant donné que la valeur de X est toujours croissante, le fragment dont la limite supérieure est **MaxKey** reçoit la majorité des écritures entrantes. Cela limite les opérations d'insertion à l'unique shard contenant ce fragment, ce qui réduit l'avantage des écritures distribuées dans un cluster de serveurs.
- En utilisant un index haché sur X, la distribution des insertions est similaire à ce qui suit :




# MongoDB : conclusion

 **Flexibilité dans le stockage des données**

 **Montée en charge**

 **Requêtes**

 **Pas de jointure**

 **Transactions (en amélioration)**

 **Gestion de la mémoire**

---