MongoDB

3IF Lab Support

Előd EGYED-ZSIGMOND

Plan

- Introduction
- Data modeling
- Practice
- CRUD queries
- Aggregates
- Indexes
- Replication
- Sharding/ Partitioning
- Conclusion

Introduction

MongoDB, one of the most popular "NoSQL" DBMS

- It is a "document based" NoSQL system (with CouchDB, ElasticSearch, ...)
- relies on a semi-structured data model of (JSON encoding);
- Schema less (flexibility);
- An original (and specific) query language;
- No (or very little) of transactional support.

Built from the beginning as a scalable and distributed system

- Distribution by partitioning (sharding) ;
- Fault tolerance through replication.

Why MongoDB

- A MongoDB server manages **data bases.**
- A database contains **collections**.
- The collections have **documents**.
- Each document has a unique identifier generated by MongoDB, **_id** field.

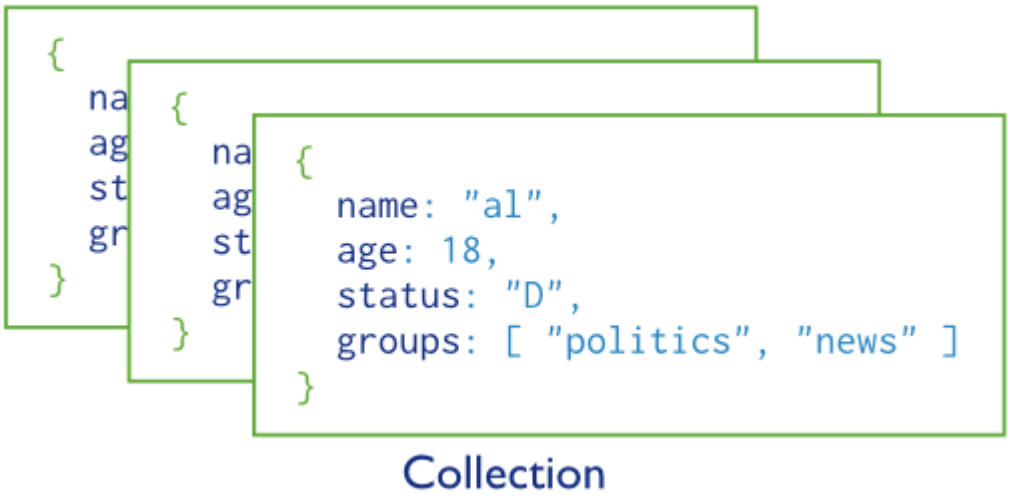
Document oriented

- The documents (lines) of the same kind are stored in collections (tables)
- A document is a tree, made of keys and values
- A value can be:
  - Scalar (int, long, string, date, binary, bool, etc.)
  - Array
  - A nested document

Vocabulary

- Document

- Collection



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

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

Replication

Sharding/ Partitioning

Conclusion

How to model

- complex document
  - nested documents
  - the value objects are part of the document
  - Sometimes greedy (denormalisation)
  - (current) limit: 4MB per document
- normalization
- mixed approach

JSON: Modeling

Relational

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	Rom

no relation

Car:

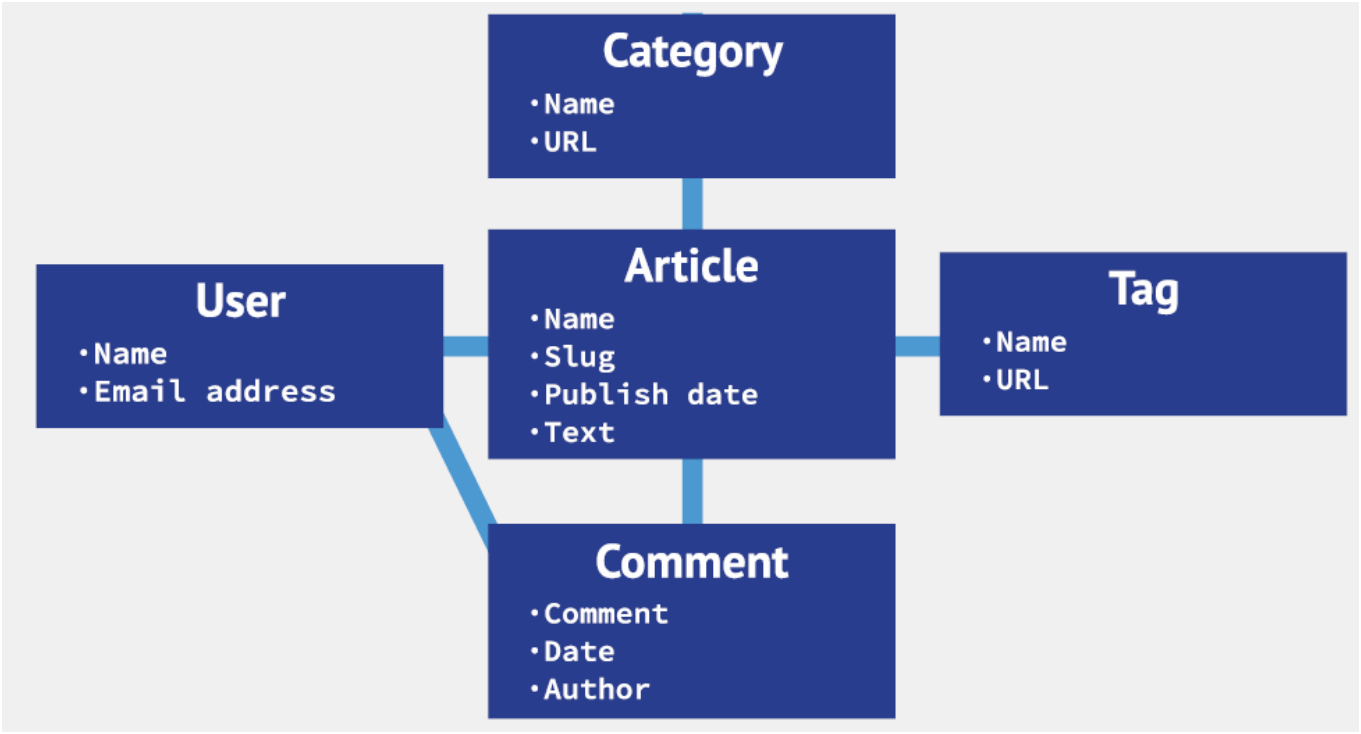
Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB Document

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
   { model: 'Bentley',
     year: 1973,
     value: 100000, … },
   { model: 'Rolls Royce',
     year: 1965,
     value: 330000, … }
  ]
}
```

Mat Keep: https://www.slideshare.net/matkeep/migrating-from-relational-databases-to-mongodb

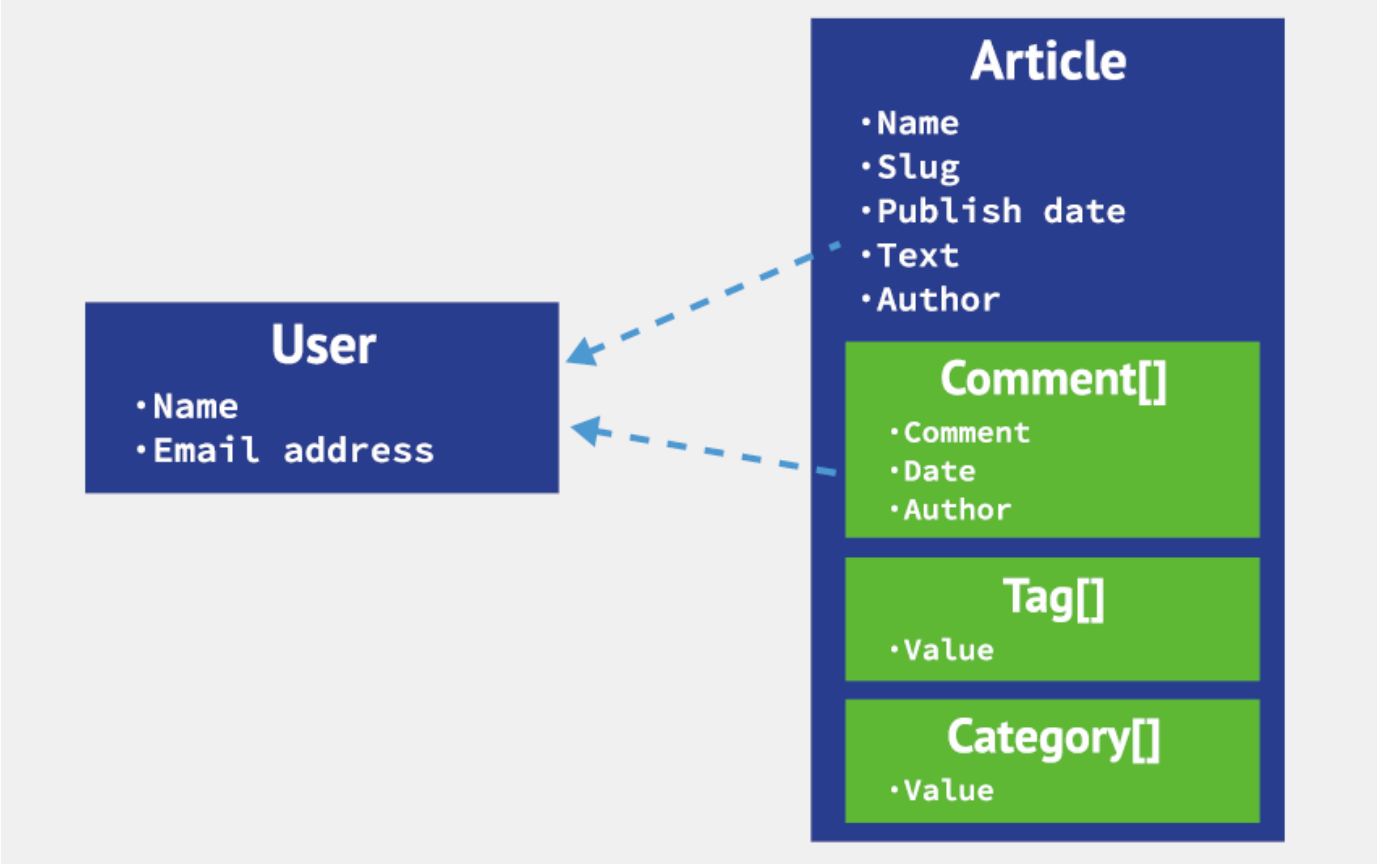
JSON: Modeling



RDBMS: Join 5 tables

Mat Keep: https://www.slideshare.net/matkeep/migrating-from-relational-databases-to-mongodb

JSON: Modeling



MongoDB: Denormalized to 2 BSON documents

Mat Keep: https://www.slideshare.net/matkeep/migrating-from-relational-databases-to-mongodb

Document Model Benefits

- Rich data model, natural data representation
  - Embed related data in sub-documents & arrays
  - Support indexes and rich queries against any element
- Data aggregated to a single structure (pre-JOINed)
  - Programming becomes simple
  - Performance can be delivered at scale
- Dynamic schema
  - Data models can evolve easily
  - Adapt to changes quickly: agile methodology

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

Replication

Sharding/ Partitioning

Conclusion

Convenient

MongoDB comes with a shell: bin/mongo

Start server with: bin/mongod

Some arguments:

`--dbpath` <path>: Data Storage path

`--port` <port>: Port the server listens

`--replSet` <Name>: Enter the server in a replicas cluster

Convenient

MongoDB comes with a shell: bin/mongosh

Start server with: bin/mongod

Some arguments:

`--dbpath` <path>: Data Storage path

`--port` <port>: Port the server listens

`--replSet` <Name>: Enter the server in a replicas cluster

Shell

- View the current database:
  - `db`
- See list of existing databases:
  - `show dbs`
- Select / create a database:
  - `use <name>`
- View Collections:
  - `show collections`
- Create collection (required)
  - `db.createCollection`

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

Replication

Sharding/ Partitioning

Conclusion

Queries in MongoDB

- JSON queries
- We find the equivalent of SQL projections
- Some default aggregations

... Map/Reduce for the others

SQL to Aggregation Mapping

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM()	$sum
COUNT()	$sum $sortByCount
join	$lookup

Mapping Chart:
http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/

CRUD

Modification:

- Insert
- Update
- Remove

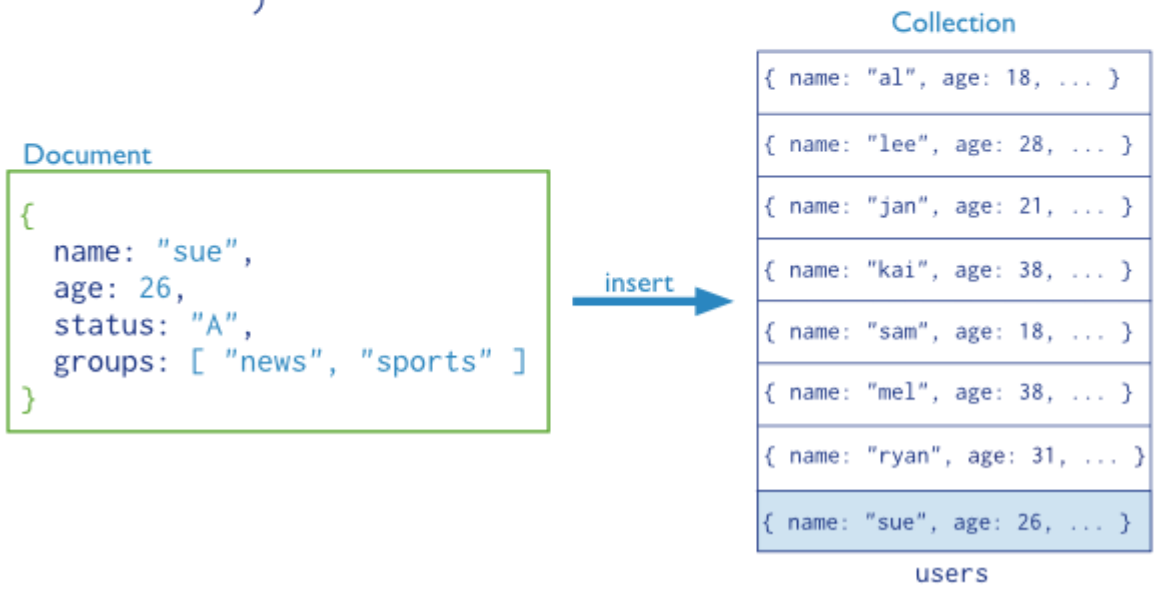
The operations are applied to a collection

http://docs.mongodb.org/manual/

CRUD

Insertion

http://docs.mongodb.org/manual/



The "_id" field is automatically added / alternative to save

CRUD

Update

```
db.collection.update(
    <Query>          Same constraints as find
    <Update>         $set, $unset...
    {
      upsert<boolean>
      Multi <boolean>
      writeConcern: <Document>
    }
)
```

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

Default update concerns a single document

http://docs.mongodb.org/manual/

CRUD

Update

```
db.movies.update({ "year":{$lt: 2000}}, {$set: {old: true}})
```

```
db.movies.find({old: true})
```

```
db.movies.updateMany({ "year":{$lt: 2000}}, {$set: {old:
true}})
```

(since MongoDB 3.2)

```
db.movies.update({ "year":{$lt: 2000}}, {$unset : {old: true}})
```

```
db.movies.update({ "year":{$lt: 2000}}, {$set: {old: true}},
{multi: true})
```

CRUD

Update, update a table

- We will add a table to a document

```
db.produits.insert({Counter: 100001, tab: [ 'a', 'b', 'c']})
```

- You can view it with the following command:

```
db.produits.find({Counter: 100001})
```

- Now to add an item to the table, use the operand **$push**:

```
db.produits.update({Counter: 100001}, {$push: {tab: 'd'}})
```

To add several items at once, there is $**pushAll** to add a whole table.

CRUD

Update a table

- The **$pop** operand will remove the last item:

```
db.produits.update({Counter: 100001} {$pop: {tab: 1}})
```

- The array has lost its last element "d".
- To remove the first element:

```
db.produits.update({Counter: 100001} {$pop: {tab: -1}})
```

- Analogously with the sort method, with -1 elements are deleted in the other direction.
- With the operand **$addToSet** an item is added without duplication

```
db.produits.update({Counter: 100001} {$addToSet : {Tab:
  'b'}})
```

- The "b" value was present in the table so it is not added.

CRUD

Changes, isolation

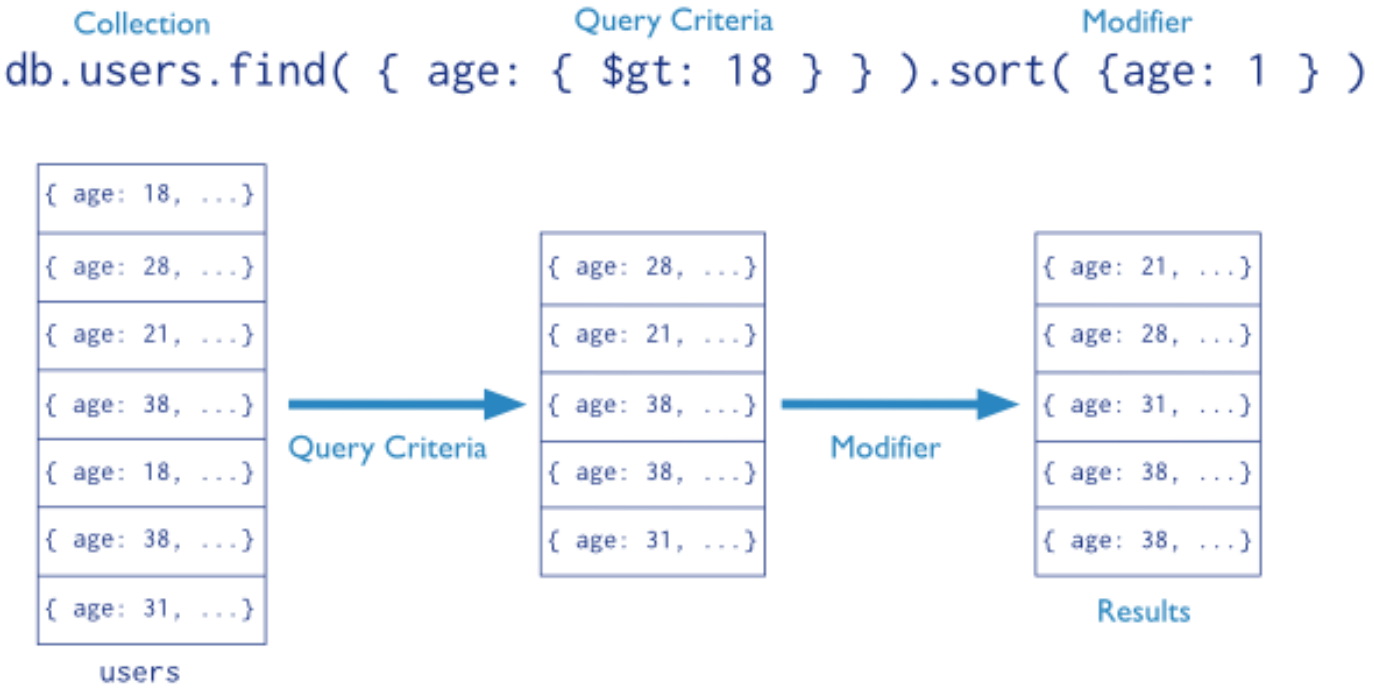
```
db.films.update(
  { old: true, $isolated : 1},
  {$inc: {count: 1}},
  {multi: true }
)

db.products.remove(
     { qty: {$gt: 20}, $isolated: 1})
```

$isolated does not work on distributed collections

CRUD

queries



http://docs.mongodb.org/manual/

CRUD

queries

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

in SQL

```
SELECT _id, name, address   ← projection
FROM   users                ← table
WHERE  age > 18             ← select criteria
LIMIT  5                    ← cursor modifier
```

```
db.movies.find({ "country": {$ne: "USA"}}, {_id: 0, title: 1
}).limit(5)
```

http://docs.mongodb.org/manual/

operators

$eq	**=**	**values which are equal to a specified value.**
$gt	>	values that are greater than a specified value.
$gte	> =	values that are greater than or equal to a specified value.
$lt	<	values which are less than a specified value.
$lte	<=	values that are less than or equal to a specified value.
$ne	!=	all values that are not equal to a specified value.
$in	∈	one of the values specified in a table.
$nin	∉	none of the values specified in the table.
$or		
$and		
$not		
$nor		
...		

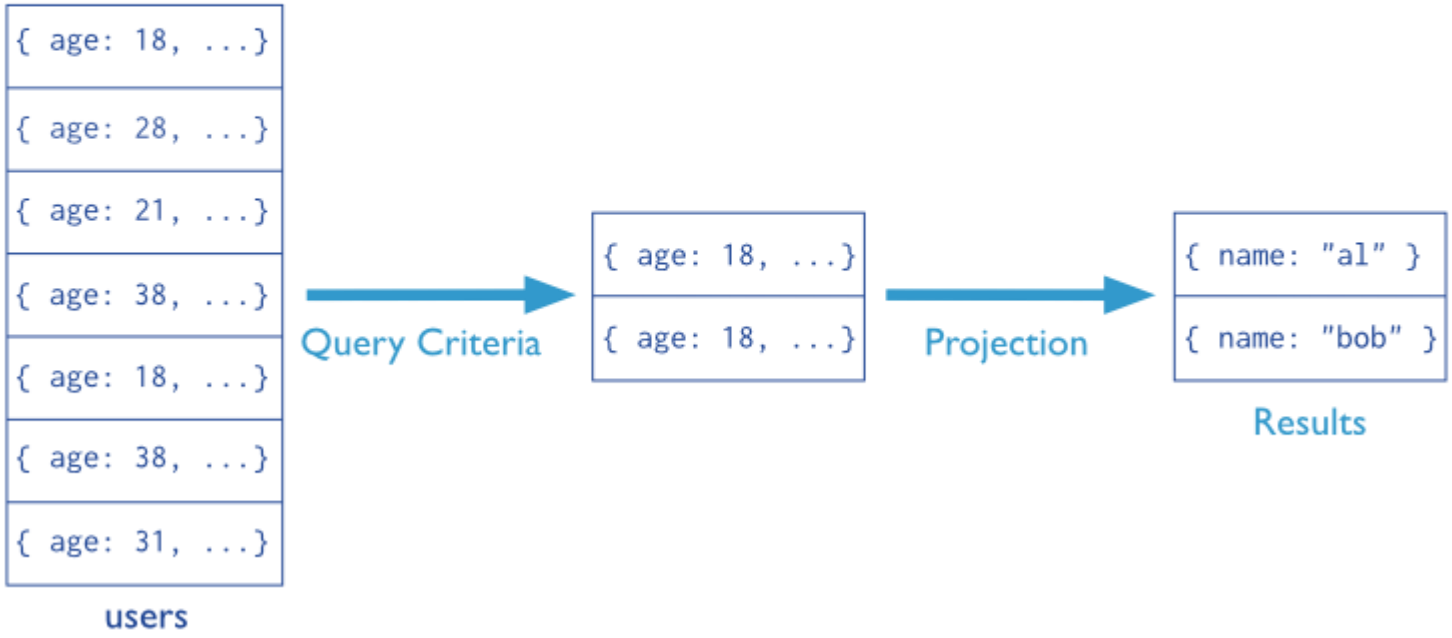
http://docs.mongodb.org/manual/reference/operator/query/

CRUD

projection

Collection Query Criteria Projection

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




```
db.movies.find({ "country":{$ne: "USA"}}, {_id: 0, title: 1})
```

CRUD Examples

Exclude the year:

```
db.films.find({ "country":"USA"}, {year: 0})
```

Return the title and genre and _id

```
db.films.find({ "country":"USA"}, {title1, type: 1})
```

Return movies with a given role

```
db.films.find({ "actors.role":"William Munny"})
```

Return the title and genre

```
db.films.find({ "country":" USA"}, {title: 1, type: 1, _id: 0})
```

Sort Results

```
db.films.find({ "year":{$gt: 2000}}).sort({year: -1})
```

CRUD

Examples

```
db.movies.find({$and : [{"year": 2003}, {genre: "romance"}] },{"title":1,"genre":1});
```

```
db.movies.find({$or : [{"year": 2003}, {genre: "romance"}] },{"title":1,"genre":1});
```

CRUD

sliders

```
db.collection.find()
db.films.find({ "country": "FR"})
```

Course one by one results

```
var myCursor = db.films.find( { country: 'FR' } );

while (myCursor.hasNext()) {
   print(tojson(myCursor.next()));
}
```

```
var myCursor = db.films.find( { country: 'FR' } );

myCursor.forEach(printjson);
```

CRUD

Cursors, methods

- *limit(n)*: To retrieve only first n results
- *sort (…)*: To sort the results
- *skip (n)*: To skip n results

For example, for the penultimate film in the name, you could do:

```
var cur = db.films.find();
cur.sort({title: -1}).limit(1).skip (2);
```

CRUD

Cursors

Transformation into an array

```
var myCursor = db.films.find( { country: 'FR' } );
var documentArray = myCursor.toArray();
var myDocument = documentArray[1];

print(tojson(myDocument));
```

Analyze query performance

```
db.films.find({Country: 'FR'}).explain();
db.films.createIndex({country: 1})
```

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

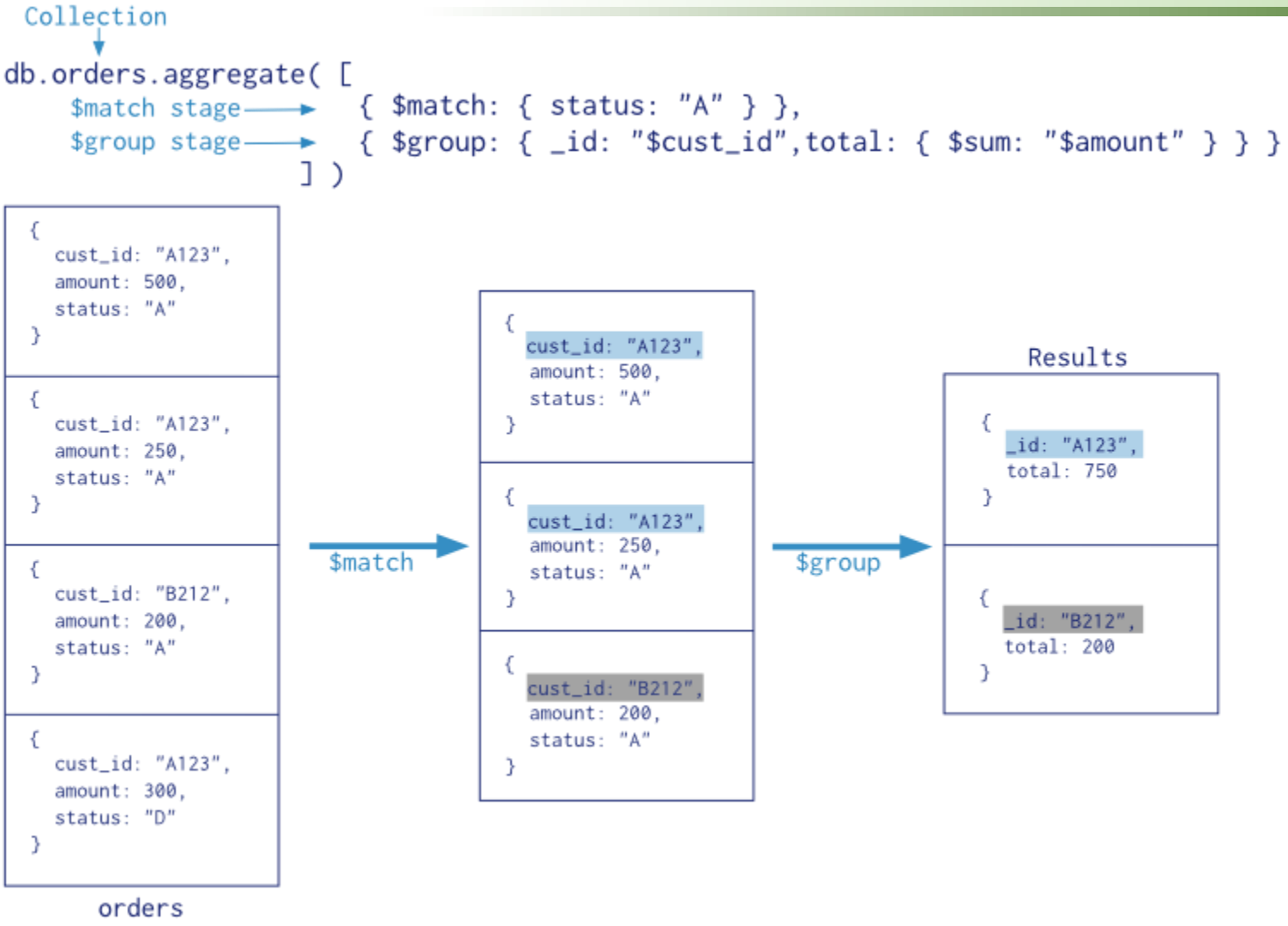
Indexes

Replication

Sharding/ Partitioning

Conclusion

aggregates



https://docs.mongodb.org

aggregates

Nb films by genre

```
db.movies.aggregate([
  {$group:{_id:"$genre",count:{$sum:1}}}
])
```

Number of American films by genre

```
db.movies.aggregate([
{$match:{country:"USA"}},
{$group:{_id:"$genre",count:{$sum:1}}}
])
```

Date of first horror movie

```
db.movies.aggregate([
{$match:{genre:"Horreur"}},
{$group:{_id:"$genre",debut:{$min:"$year"}}}
])
```

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

Replication

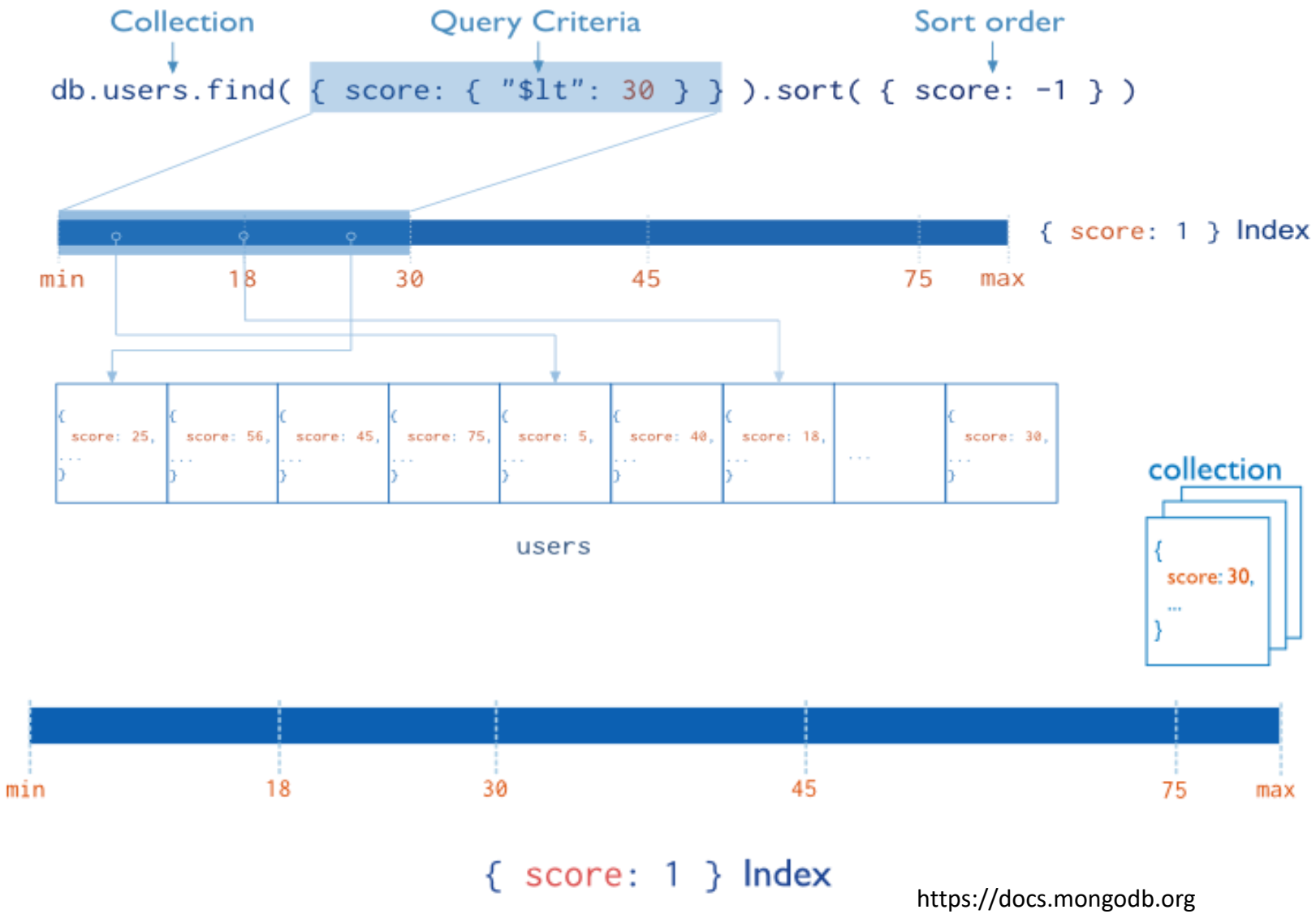
Sharding/ Partitioning

Conclusion

indexes

- Very similar to the RDBMS, indexation in MongoDB is made on one or more fields.
- Improves search performance.
- The indexes are stored in the collections.
- Provides overload for write operations.
- The internal operation is very similar to that found in the current DBMS.

indexes



https://docs.mongodb.org

indexes

Types of indexes

- single field
- composed field
- multi key
- geo spatial
- text
- Hashed

indexes

- Consider using indexes effectively.
- A query, a bit field has no interest in being indexed
- Although we speak of NoSQL the indexes are similar to those in the RDBMS.

Loading Documents

- unitary insert:
  - Example: coll.insert({x:1,y:2,z:"test"})
  - Command to avoid for large inserts via a client (> 10 000 documents)
- Bulk insert:

  -Example:

```
var Documents = [];
        for(var i = 1; i < 10 000; i++) {
documents.push({x:i,y:i, z:"test"})};
            db.coll.insert(documents);
```

Inserting massive documents

- Program to load a BSON file containing hundreds of GB or more in a MongoDB base: *mongorestore*
- Mongorestore: Method far more effective to insert data (other methods: (bulk) Insert, mongoimport)
- Backup / Restore tool of a MongoDB base (Sharded) with *mongodump*

other features

- geographic queries

- Javascript running on the server

  db.eval (~ Stored procedures)
  map/Reduce (Aggregations)
  db.system.js (functions)

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

Replication

Sharding/ Partitioning

Conclusion

Replication

What brings replication?

- Redundancy
- Simplify tasks (backups, ...)
- Increase reading ability

- A replica set is a cluster of MongoDB instances.
- master / slave strategy
- There must ALWAYS be a single master.

Replication

- The master to the slave replication is asynchronous.

- Synchronous: Blocking / Expensive / Strong consistency
- Asynchronous: No blocking / Data refreshing mandatory.

Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

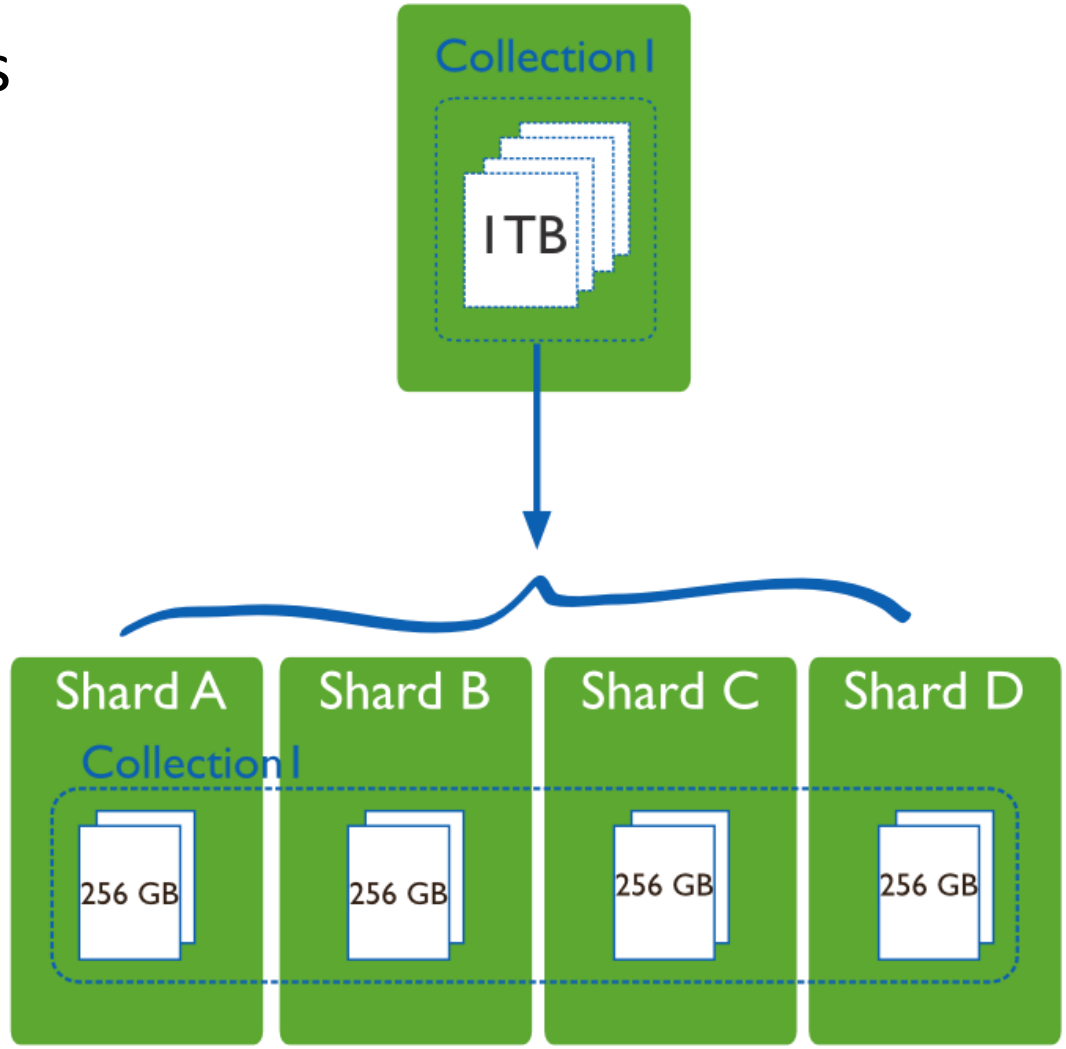
Replication

Sharding/ Partitioning

Conclusion

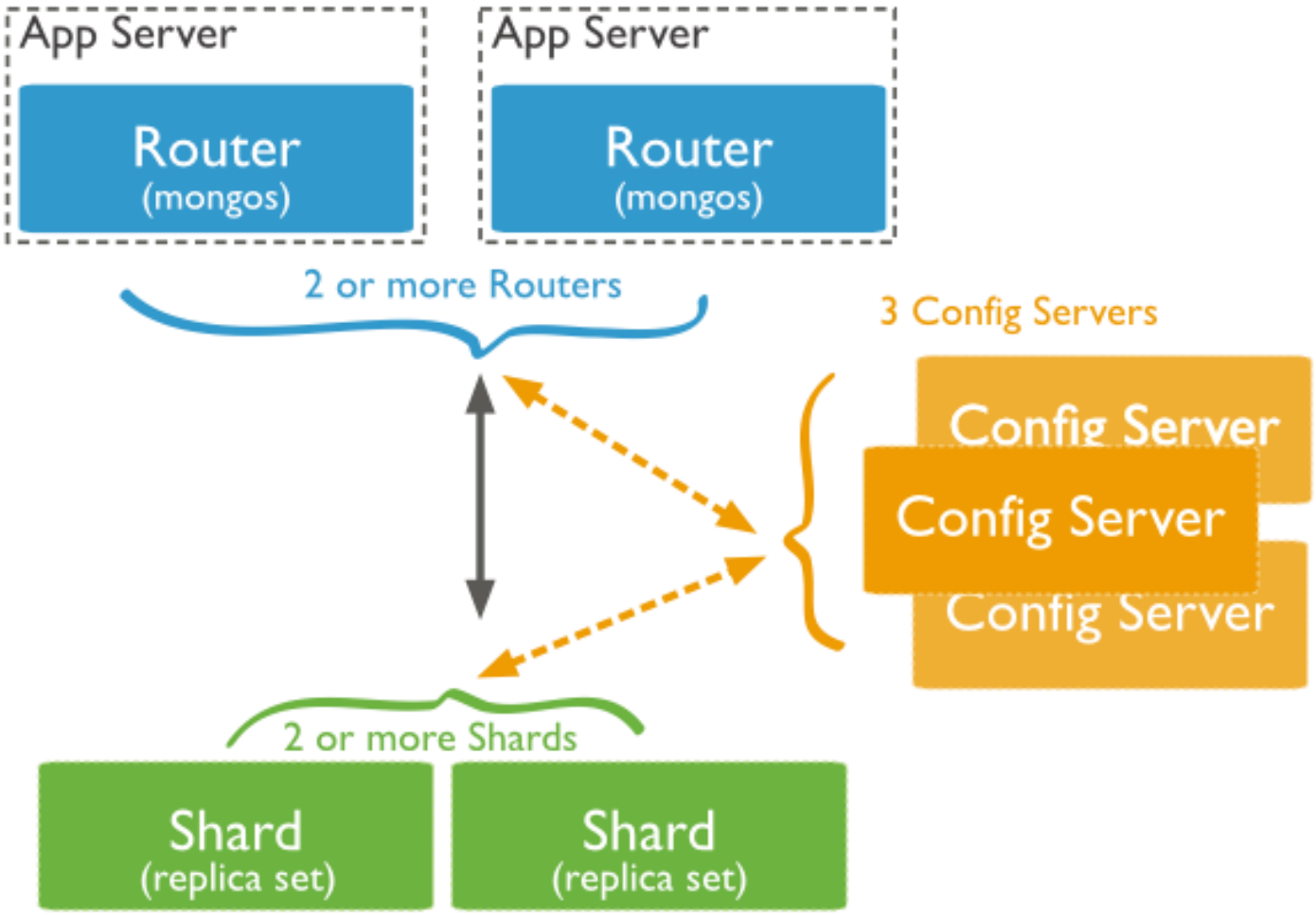
sharding

- The sharding enables automatic data partitioning



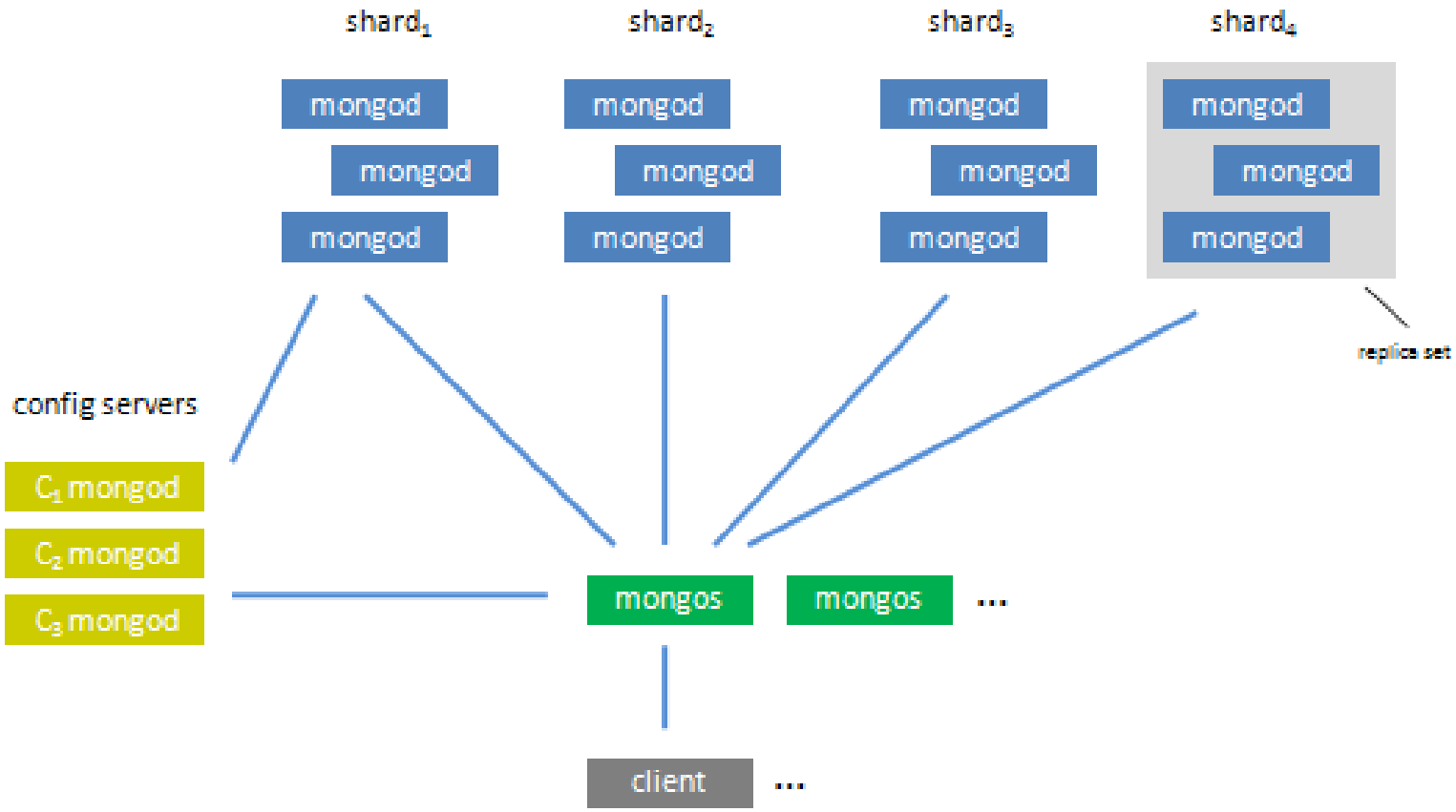
https://docs.mongodb.org

sharding



https://docs.mongodb.org

sharding and replication



Plan

Introduction

Data modeling

Practice

CRUD queries

Aggregates

Indexes

Replication

Sharding/ Partitioning

Conclusion

More Information

Resource	Location
MongoDB Downloads	mongodb.com/download
Free Online Training	education.mongodb.com
Webinars and Events	mongodb.com/events
White Papers	mongodb.com/white-papers
Case Studies	mongodb.com/customers
Presentations	mongodb.com/presentations
Documentation	docs.mongodb.org

References

http://nosql.developpez.com/

http: //news.humancoders.com/t/nosql/

https://docs.mongodb.com/manual/reference/sql-comparison/

MongoDB

Official site : www.mongodb.org

A client: https://studio3t.com/