Data Management

Prof. Alejandro Zunino

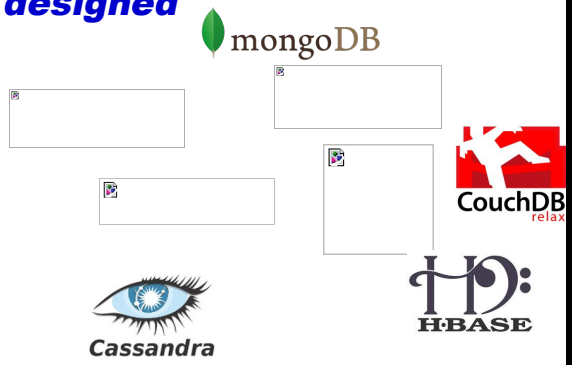
Prof. Alfredo Teyseyre

UNICEN

Contents

NoSQL - non SQl or non relational

A variety of storage solutions were designed

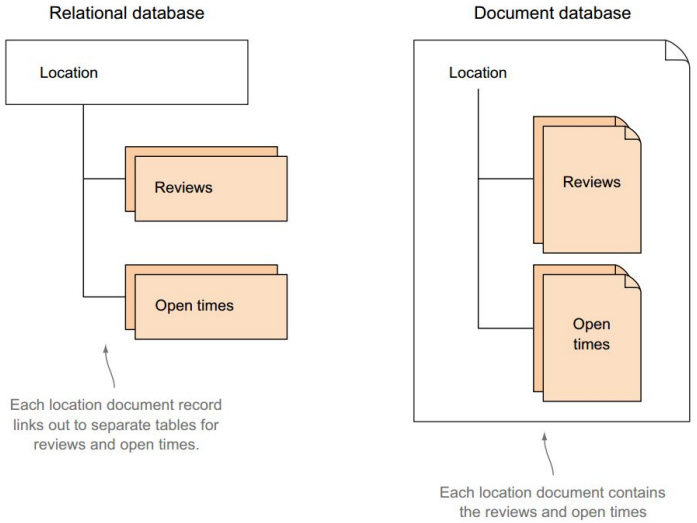


- Key-value
- Graph database
- Document-oriented
- Column family

Document storage solutions that were designed for better availability, simple querying, and horizontal scaling

Document Storage

Storing your data as holistic documents will allow faster read operations since your application won't have to rebuild the objects with every read



Document Storage

Although there are a few document-oriented databases, none are as popular as MongoDB

- Faster read operations
- Model changes
  - Schemaless
    - different properties for different objects
  - e-commerce application: furniture
    - large number of empty columns (relational)

Features

- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce

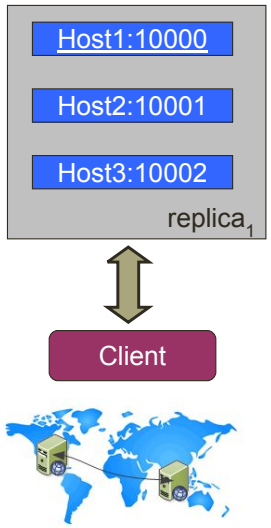
Agile

Scalable

Replica Sets

To provide data redundancy and improved availability, MongoDB uses an architecture called replica set

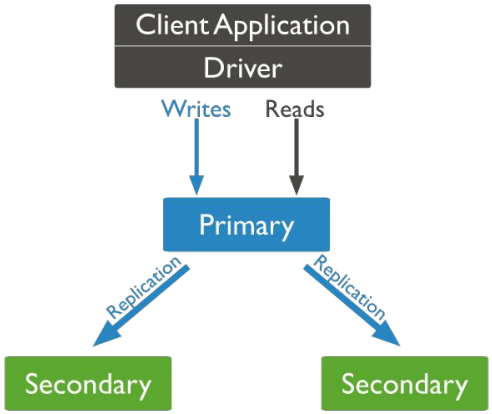
- Redundancy and Fail over
- Zero downtime for upgrades and maintenance
- Master-slave replication
  - Strong Consistency
  - Delayed Consistency
- Geospatial features



Replica Sets

To provide data redundancy and improved availability, MongoDB uses an architecture called replica set

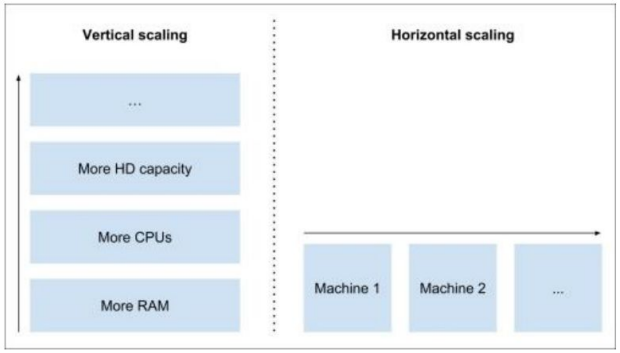
- One service is used as the primary and the other services are called secondaries
- All of the set instances support read operations
- Only the primary instance is in charge of write operations
  - When a write operation occurs, the primary will inform the secondaries about the changes
- Replica set is its automatic failover



Scaling

Scaling is a common problem with a growing web application

- Vertical scaling
  - Easy
  - More Expensive
  - Limited
  - Cloud hosting limit
- Horizontal scaling
  - Cheaper
  - Better scalability
  - More Complicated

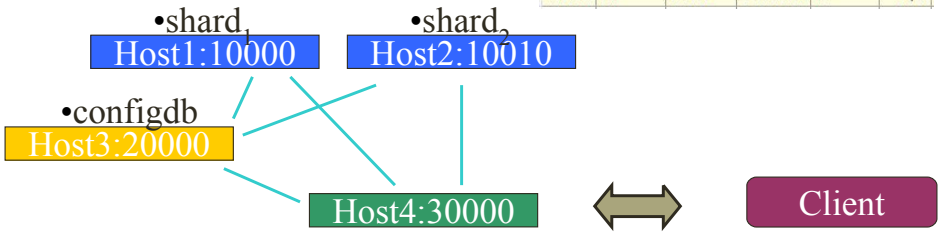


Sharding

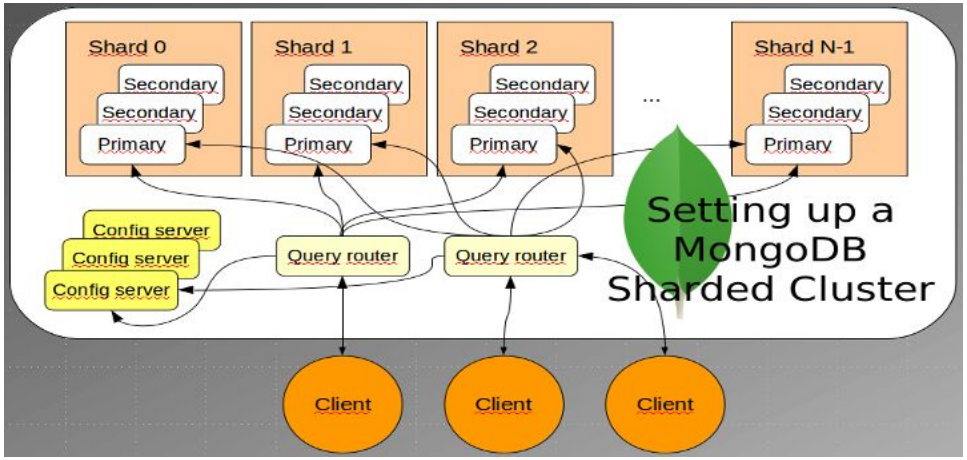
Sharding is the process of splitting the data between different machines, or shards

- Partition your data
- Scale write throughput
- Increase capacity
- Auto-balancing

id	company	customer	article	currency	price
4250250	020	073000	5994537812	00	142,50
4250251	020	073000	5994537852	00	141,12
4250252	020	073000	5994537854	00	105,99
4250253	020	073000	5994537856	00	109,52
4250254	020	073000	5994537862	00	131,49
4250255	020	073000	5994567308	00	29,86
4250256	020	073000	5994567422	00	57,13
4250257	020	073000	5994567428	00	68,59
4250258	020	073000	5994605089	00	51,09
4250259	020	073000	5994607975	00	93,93
4250260	020	073000	5994701005	00	74,22



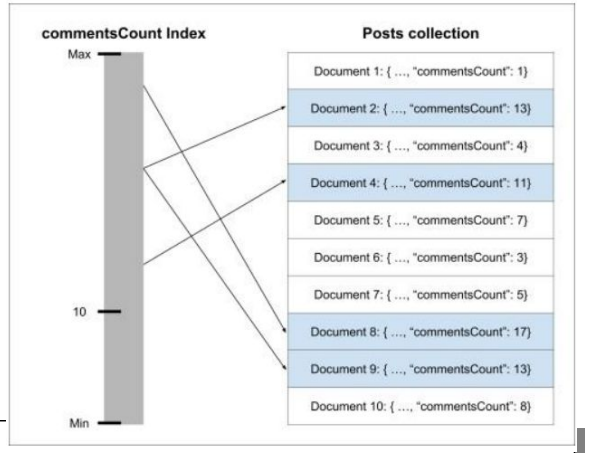
Mixed



MongoDB indexing

Indexes are a unique data structure that enables the database engine to efficiently resolve queries

```
{
"_id":
ObjectId("52d02240e4b01d67d71ad577"),
"title": "First Blog Post",
"comments": [
],
"commentsCount": 12
}
```




```
db.posts.createIndex( { commentsCount: -1 } ) //descending index
db.posts.find({ commentsCount: { $gt: 10 } });
```

Document Store

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (JSON, BSO
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key		Reference
Partition		Shard

```
> db.user.findOne({age:39})
{
      "_id" : ObjectId("5114e0bd42…"),
      "first" : "John",
      "last" : "Doe",
      "age" : 39,
      "interests" : [
            "Reading",
            "Mountain Biking ]
      "favorites": {
            "color": "Blue",
            "sport": "Soccer"}
}
```


CRUD

- Create
  - db.collection.insertOne( )
  - db.collection.insertMany()
- Read
  - db.collection.find()
- Update
  - db.collection.updateOne( )
  - db.collection.updateMany( )
- Delete
  - db.collection.deleteOne()
  - db.collection.deleteMany()

…..

Create Operations

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

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

Insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

Read Operations

Read operations retrieves documents from a collection; i.e. queries a collection for documents.

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

Update Operations

Update operations modify existing documents in a collection

```
db.users.updateMany(                    <-- collection
   { age: { $lt: 18 } },                <-- update filter
   { $set: { status: "reject" } }       <-- update action
)
```

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

Delete Operations

Delete operations remove documents from a collection

```
db.users.deleteMany(                    <-- collection
   { status: "reject" }                 <-- delete filter
)
```

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

CRUD Example

```
> db.users.insertOne({
      name: "John",
      age: 39
})
```

```
> db.users.find ()
{
      "_id" : ObjectId("51…"),
      "name" : "John",
      "age" : 39
}
```

```
> db.users.updateOne(
      {"_id" : ObjectId("51…")},
      {
            $set: {
                  age: 40,
                  salary: 7000}
      }
)
```

```
> db.users.deleteOne({
      "name": ”John”
})
```


MongoDB shell (VM)

- https://www.mongodb.com/docs/mongodb-shell
  - https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/inventory.crud.json

Sample document

{ "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" }
{ "item": "notebook", "qty": 50, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "A" }
{ "item": "paper", "qty": 100, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "D" }
{ "item": "planner", "qty": 75, "size": { "h": 22.85, "w": 30, "uom": "cm" }, "status": "D" }
{ "item": "postcard", "qty": 45, "size": { "h": 10, "w": 15.25, "uom": "cm" }, "status": "A" }


```
mongoimport   --authenticationDatabase admin --username root --password tmjnjAkwRy77 --db test
--collection inventory  --drop --file inventory.crud.json
```

MongoDB shell (Docker)

- https://docs.mongodb.org/getting-started/shell/

Sample document

```
{ "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" }
{ "item": "notebook", "qty": 50, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "A" }
{ "item": "paper", "qty": 100, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "D" }
{ "item": "planner", "qty": 75, "size": { "h": 22.85, "w": 30, "uom": "cm" }, "status": "D" }
{ "item": "postcard", "qty": 45, "size": { "h": 10, "w": 15.25, "uom": "cm" }, "status": "A" }
```

```
docker cp ./inventory.crud.json docker-bitnami-mongodb-1:/tmp

docker-compose exec mongodb bash

mongoimport  --db test --collection restaurants --drop --file /tmp/inventory.crud.json
```


MongoDB shell: Insert

```
mongosh admin --username root --password YUsQCG3zg9rE    //(si no funciona password usar
                                                //        bitnami)
use test
db.createUser({ user: "Alice",pwd: "Moon1234", roles: [ "readWrite", "dbAdmin" ]} )
exit

mongosh  test --username Alice --password Moon1234


db.inventory.insertMany([
  // MongoDB adds the _id field with an ObjectId if _id is not present
  { item: "journal", qty: 25, status: "A",
     size: { h: 14, w: 21, uom: "cm" }, tags: [ "blank", "red" ] },
  { item: "notebook", qty: 50, status: "A",
     size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank" ] },
  { item: "paper", qty: 100, status: "D",
     size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank", "plain" ] },
  { item: "planner", qty: 75, status: "D",
     size: { h: 22.85, w: 30, uom: "cm" }, tags: [ "blank", "red" ] },
  { item: "postcard", qty: 45, status: "A",
     size: { h: 10, w: 15.25, uom: "cm" }, tags: [ "blue" ] }
]);
```


MongoDB shell: Queries

```
db.inventory.find()  //SELECT * FROM inventory

{ "_id" : ObjectId("5caceb8ba6e93908cd7382e7"), "item" : "journal", "qty" : 25, "status" :
"A", "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "tags" : [ "blank", "red" ]

        //top level field: SELECT * FROM inventory WHERE status = "D"
db.inventory.find( { status: "D" } )

                // Match an Embedded Document
db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )

                // Match a Field in an Embedded Document
db.inventory.find( { "size.uom": "in" } )

                // Match an Element in an Array
db.inventory.find( { tags: "red" } )

                // Match an Array Exactly
db.inventory.find( { tags: ["red", "blank"] } )

// AND Conditions: SELECT * FROM inventory WHERE status = "A" AND qty < 30
db.inventory.find( { status: "A", qty: { $lt: 30 } } )

//Or Conditions:SELECT * FROM inventory WHERE status = "A" OR qty < 30
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```


MongoDB shell: Update

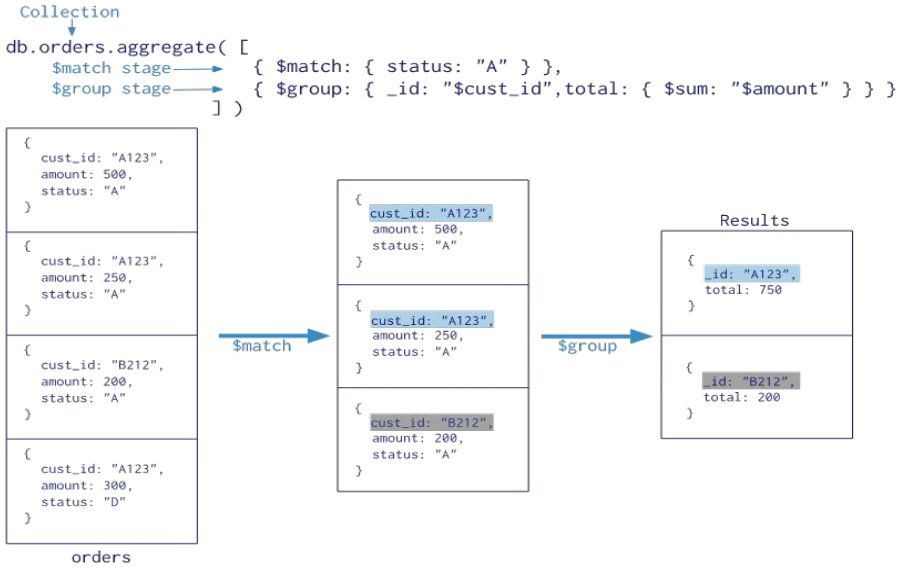
```
                //single document

db.inventory.updateOne(
  { item: "paper" },
  {
    $set: { "size.uom": "cm", status: "P" },
    $currentDate: { lastModified: true }
  }
)



                // multiple documents

db.inventory.updateMany(
  { "qty": { $lt: 50 } },
  {
    $set: { "size.uom": "in", status: "P" },
    $currentDate: { lastModified: true }
  }
)
```

MongoDB shell: Delete

```
//Remove All Documents
db.inventory.deleteMany({})

//Remove All Documents that Match a Condition
db.inventory.deleteMany({ status : "A" })

// Use the justOne
db.inventory.deleteOne( { status: "D" } )

//Drop a Collection
db.inventory.drop()
```

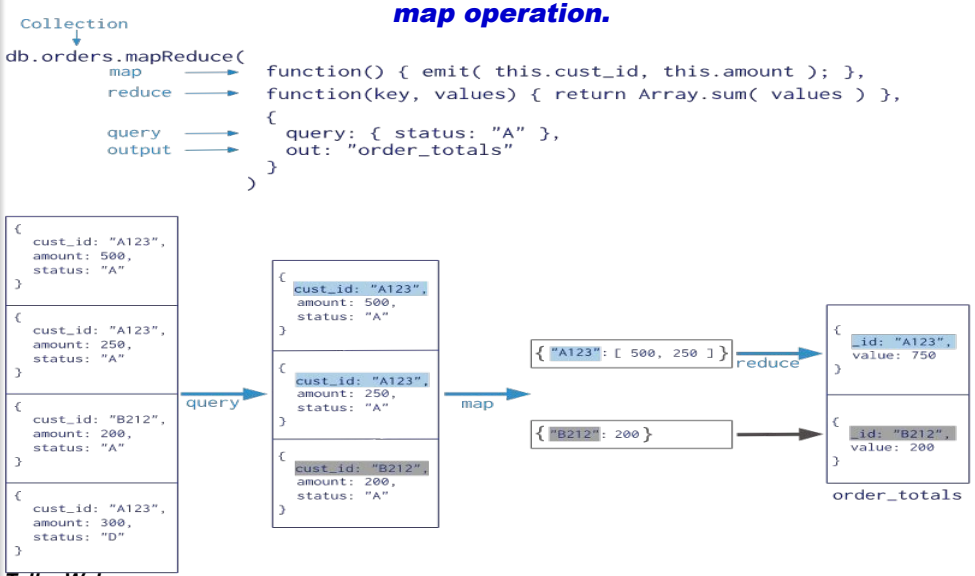

MongoDB shell: Data Aggregation

Documents enter a multi-stage pipeline that transforms the documents into an aggregated result



MongoDB shell: Data Aggregation

Map-Reduce operations have two phases: a map stage that processes each document and emits one or more objects for each input document, and reduce phase that combines the output of the map operation.



MongoDB shell

- https://docs.mongodb.org/getting-started/shell/
- Tarea: Crear una base, usuario, definir un documento propio y ejercitar operaciones CRUD