MongoDB

Andrés Ramírez

Acerca de mí

- Ingeniería en sistemas computacionales
- Android developer 2015
- andres200493@gmail.com

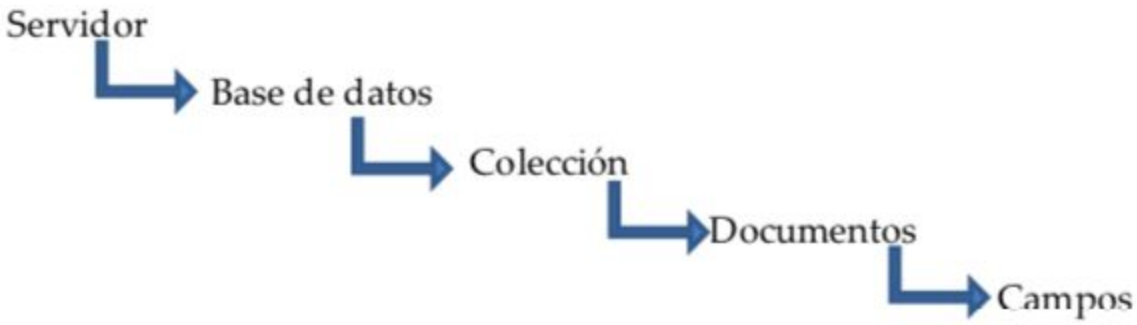


Figura 15: Arquitectura funcional MongoDB

```
{
   _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

- No SQL
- Orientado a documentos

https://hevodata.com/blog/install-mongodb-on-ubuntu/

https://platzi.com/blog/como-instalar-mongodb-en-window-linux-y-mac/

https://jafrancov.com/2013/09/instalar-mongodb-osx/

Crear DB

Show dbs

Use nombre_db

Eliminar DB

db.dropDatabase()

Crear colección

```
db.tbl.insert({campo: "valor", ...})

db.tbl.insertOne({campo: "valor", ...})


db.tbl.insertMany([
    {campo: "valor", ...},
    {campo: "valor", ...},
])
```

```
db.tbl.find().explain()
```

Buscar en colección

db.tbl.find({ "name.last": "Hopper" })

db.tbl.find({ _id: { $in: [5,ObjectId("507c35dd8fada716c89d0013")] } })

db.tbl.find({ birth: { $gt: new Date('1950-01-01') } })

db.tbl.find({ "name.last": { $regex: /^N/ } })

db.tbl.count({nombres:{$exists: true}})

db.tbl.distinct("nombre")

.limit(N) .skip(N)

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

Actualizar colección

```
db.tbl.find( { _id: 5 } )

db.tbl.update({"nombre": "Andres"}, {$set: {"edad" : 26}})
```

- $inc: incrementa o decrementa un campo en función del valor indicado
- $unset: elimina un campo añadido
- $push: añade un valor al campo, trabajando con arrays
- $addtoSet: añade valores a un array
- $pop: elimina el último elemento de un array
- $pull: elimina todas las concurrencias que coincidan con el valor indicado para el campo requerido
- $rename: renombrado de un campo
- $bit: actualización bit a bit de un campo

```
db.tbl.update({"nombre": "Andres"}, {$inc: {edad : 1}})
```

Eliminar

```
db.alumnos.remove({nombre: "Juan"})
```

```
db.alumnos.remove({"edad":{$lt:18}})
```

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Relaciones

Se pueden definir 3 tipos de relaciones:

- One-to-one (uno a uno)
- One-to-many embebido (uno-a-varios)
- One-to-many con referencias (uno-a-varios)

One-to-one

Embeberse dentro de otro

```
{ _id: <ObjectId3>,
  nombre: “Julio”,
  apellido1: “González”,
          ...
  dirección: {
                calle: “Avenida de la República”,
                numero: 678
                      ...
              }}
```

One-to-many embebido (uno-a-varios)

Embeben dentro de otro en una estructura de tipo array

```
{ _id: <ObjectId3>,
  nombre: "Julio",
  apellido1: "González",
          ...
  direcciones: [{
                calle: "Avenida de la República",
                numero: 678
                    ...},
               {calle: "Avenida de la República",
                numero: 678
                    ... }]}
```

```
{ _id: "edicionesSotileza",
  ciudad: "Santander",
          ...
  libros: [{
            nombre: "Historia de Cantabria",
            ISBN: "678789789"
                    ...},
            nombre: "El pleito de los nueve valles",
            ISBN: "2671982093"
                    ...
```

One-to-many con referencias (uno-a-varios)

Referencias al _id de los documentos relacionados, en vez de embeberlos por completo

```
{ _id: "edicionesSotileza",
  ciudad: "Santander",
  ...}
```

```
{ _id: "HdC1",
  nombre: "Historia de Cantabria",
  ISBN: "678789789"
  ...
  editor: "edicionesSotileza"}
```

Ventajas

Esquema muy flexible: cada documento de la colección puede almacenar campos diferentes.

Lenguaje de consulta y manipulación sencillos (operaciones CRUD).

Facilidad de integración con aplicaciones gracias al uso del lenguaje BSON, fácilmente traducible a JSON.

Accesibilidad a los datos.

Posibilidad de realizar lecturas en instancias secundarias, repartiendo la carga de trabajo.

Desventajas

Aplicaciones clientes más complejas de desarrollar al trabajar con esquemas flexibles, desnormalizados y dinámicos.

Robomongo

MongoDb
compass

Campo	Tipo	Único
_id	ObjectId	Si
username	String	Si
phone	String	No - Contacto
email	String	No - Contacto
level	Int	No - Acceso
group	String	No - Acesso

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

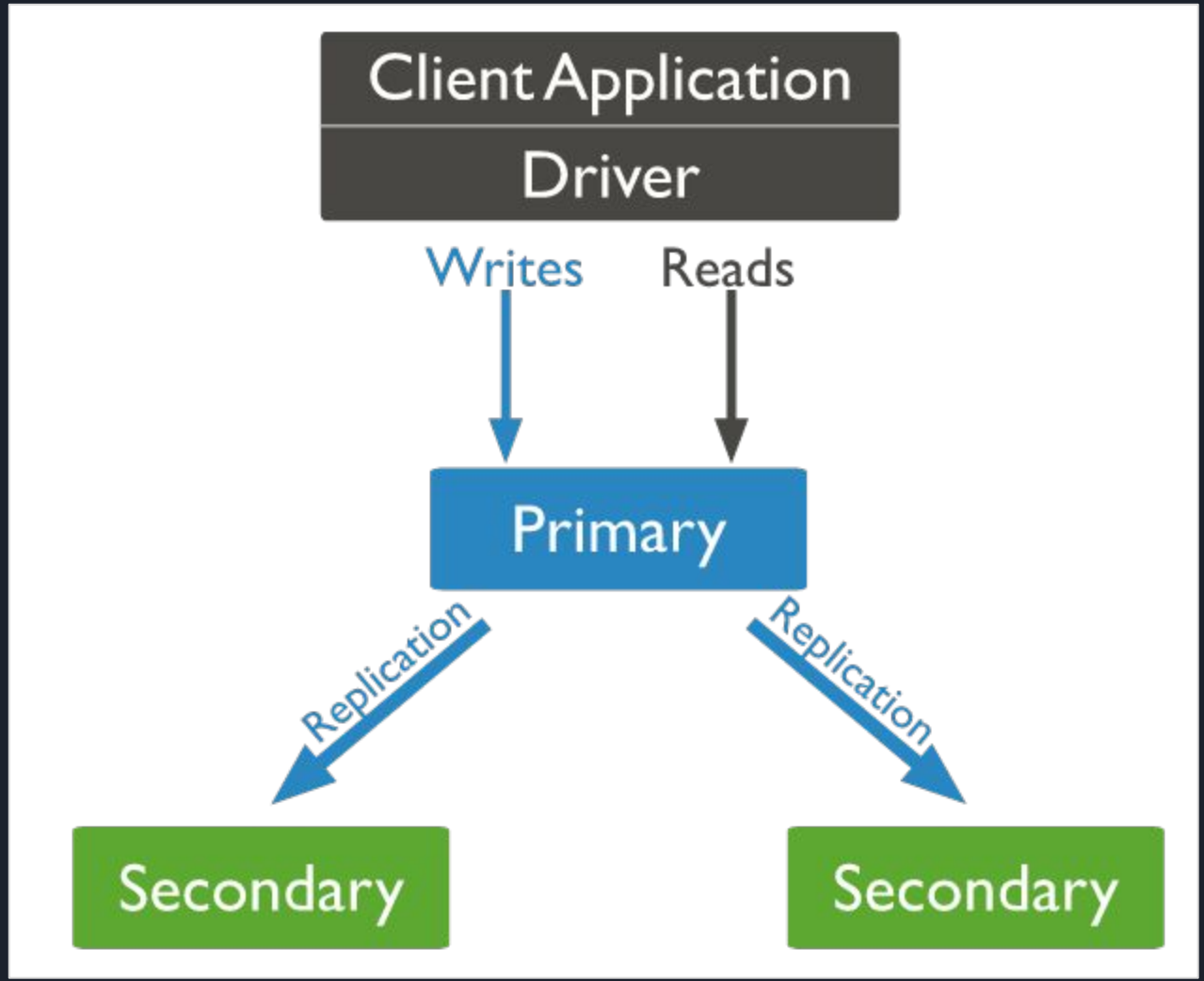
access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document


Client Application
Driver
Writes
Reads
Primary
Replication
Replication
Secondary
Secondary


salamat
Dakujem
teşekkür ederim
GRACIAS
TAK
SUWUN
धन्यवाद
ASANTE
hvala
salamat
TAKK
HVALA
mersi
شكرا
جزيل
Ευχαριστώ
감사합니다
GRAZAS
kiitos
merci
GRAZZii
DANKE
salamat
TAKK
MAHALO
Рахмет
kiitos
Thank you
arigato
ARIGATO
takk
suwun
धन्यवाद
HVALA
GRAZIE
DAKUJEM
teşekkür ederim
kiitos
hvala
MERCI
mahalo
GRACIAS
GRAZAS
TAKK
ありがとう
DANKE
salamat
SUWUN
Благодарам
TAKK
ASANTE
多謝
SALAMAT
gra2ie
спасибо
gracias