



# T& : BBDD NoSQL

## Conceptos Generales

BBDD Relacionales:

- 1) Consulta + Actualizaciones sobre registros individuales de forma casi inmediata

Transaccionabilidad

- 2) No están pensadas para hacer análisis posteriores

## Comparativa SQL vs NoSQL

### SQL

Estructurado

Centralizado

coherencia de cada  $\square$

ACID

Acceso aleatorio de los datos

Transaccionabilidad

escalab. Vertical

No penaliza

lectura, penaliza actualizaciones

Basically Available

Aunque  $\exists$  esté disponible, es posible q algún dato no lo esté

NoSQL

Soft state

Aunque consistencia, inmediata, los valores pueden cambiar (responsb. A  $\square$ )

Eventually Consistent

Aunque consistencia, permite op. de lectura  $\rightarrow$  Actualizaciones no inmediatas

Teorema CAP

como muchos

Un sist. distrib. no puede garantizar al =  $\ominus$

Consistencia:  $\leftarrow$  Disponib.: Tolerancia a particiones

### Sistemas CP

solo los On consistentes

están disponibles

HBase, MongoDB, ...

DynamoDB

Todos los On responden al =  $\ominus$  aunque no sean consistentes

repartidos a los Sist. Sist. bajo forma de errores en una partición

repartiendo a los Sist. Distrib. (RBAM)

Neatly ( $\times$  puede querer de forma distrib., pero con nodos muy concentrados dentro de un cluster)

Bien Config.

### Sistemas AC

Si repartimos a los Sist. bajo forma de errores en una partición

repartiendo a los Sist. Distrib. (RBAM)

Neatly ( $\times$  puede querer de forma distrib., pero con nodos muy concentrados dentro de un cluster)

Bien Config.

## Mongo DB



Orientada a Documentos

Estructura de datos compuesta por una sucesión de campos en forma clave:valor

= JSON

```
{
  Nombre: 'Juan',
  Edad: 28,
  Categoría: 'Supervisor',
  Centros: ['Arguelles',
            'Lagasca']
}
```

flexibilidad esquema dinámico

Representación = usada lenguajes program.

- 1 documento  $\rightarrow$  encapsula todo el entorno conceptual
- se adapta mejor a los problemas reales

## MongoDB

- permite indexar cualquier campo
- Facilita análisis de la info. gracias a sus \$()
- Distribuido  $\rightarrow$  esquema de replicación esclavo-maestro
- permite agregar documentos (= filas en tablas)

IDs generados de forma automática:

```
{
  _id: <ObjectId1>,
  username: "23xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```

Embedded sub-document

## Comandos Básicos

```
> Use midatabase: se conecta o crea
show collections
show dbs / show databases
quit();
```

```
> Crear documentos
>> db.foo.insertOne({ident: '1'})
>> db.foo.insertMany([
  {ident:'2'},
  {ident:'3'}
])
Many implica listas y listas
implican corchetes
```

```
> Actualizar documentos
>> db.foo.updateOne({ "ident": '1' }, {otras condiciones}),
{ $inc: { "ident": 100 } }) Los operadores siempre van con $ delante
>> db.foo.updateOne({ "ident": '1' }, { $set: { "ident": '100' } })
```

```
otros operadores:
$set: se puede utilizar para crear campos
$unset: para eliminarlos
$ne: not equal
operadores de comparación: $eq, $gt, $lt, $in, $gte, $lte ...
$multi: actualizar todos los registros si se usa update y no updateMany
```

```
> Actualizar documentos: crea un nuevo campo que se llame idnorm entero.
Pon un valor por defecto
```

```
>> db.foo.updateMany({$ne:{'id':111}},{$set:{idnorm:100}})
>> db.foo.updateMany({ident:'1'},{$unset:{idnorm:1}})
```

```
> Colecciones y subdocumentos
> Agregar elemento a listas: $push:{clave:valor} / $push:{clave:[listado de elementos]}
> $addToSet / $each: evita duplicados $addToSet:{clave:$each:[listado de elementos]})
```

```
>> db.empleados.updateOne({nombre:'Coseme'},{$push:{telefonos:9111221211}})
>> db.empleados.updateOne({nombre:'Coseme'},{$addToSet:{telefonos:9111221211,9123331211}))
```

```
> Eliminar elemento a listas: $pull:{clave:valor}
$pullAll:{clave:[listas valore]} / db.empleados.updateOne({nombre:'Coseme'},{$pullAll:{telefonos:[9111221211,9123331211]}})
```

# T&B: BBDD NoSQL

## Comandos Básicos

➤ La actualización puede ser muy complicada en función de la estructura del documento. Imaginad:

```
{ "nombre": "perico",  
  notas:[  
    {tipo:'practico', nota:2},  
    {tipo: 'teorico', nota: 10}  
  ]  
}
```

¿Cómo actualizas la nota del práctico a un 8?

```
db.foo.updateOne({nombre:'perico'}, {$set:{'notas.$elem.nota':8}}, {arrayFilters:[{"elem.nota":2}])  
  
.[$elem] .-> a cada elemento del array notas.nota (documento)  
arrayFilters -> establece la condición
```

### ➤ Consultas

```
find({filtrito1}, {filtritoN}, {clave:1}), y otros (ej. findOne). Por ejemplo:  
db.coleccion.find({clave:valor}, {clave:$operadorComparación:valor} ...,{clave:1})  
  
(clave:1) >> a esto se le llama 'proyección', y sirve para devolver unos campos determinados
```

```
➤ visualizar bien formado: .pretty()  
➤ ordenar: .sort({clave:1})  
➤ contar registros:.count()  
➤ limitar: limit()  
➤ distintos: distinct()
```

### ➤ Consultas múltiples:

```
➤ $and, $or, $not  
➤ Los operadores se ponen antes de la condición:
```

```
($operador:[{filtro}, {filtro}])
```

Donde filtro:  
{clave:\$operadorComparación:valor}  
{clave:valor}

Por ejemplo:

```
>> db.foo.find({$and:[{'ident':'2'}, {idnorm:$gte:100}]})  
>> db.foo.find({$or:[{'ident':'2', 'ident':'3'}]})
```

### ➤ Expresiones regulares:\$regex

```
>> db.foo.updateOne({'ident':'100'}, {$set:{nombre:'Raul'}})  
>> db.foo.updateOne({'ident':'3'}, {$set:{nombre:'Pepa'}})  
>> db.foo.find({nombre:{$regex:'Ra*'}})
```

### ➤ Colecciones y subdocumentos

➤ Buscar en listas: empleados de Madrid o Toledo

```
db.empleados.find({'localidad':{$in:['Madrid','Toledo']}})
```

➤ Buscar en subdocumentos:

```
db.empleados.find({'subdocumento.clave':valor})  
db.empleados.find({'subdocumento.clave':{$operadorComparación:valor}})  
db.empleados.find({'dependientes.nombre':'Juan'})
```

### ➤ Colecciones y subdocumentos

➤ Buscar en listas: empleados que se apelliden Villa o similar y cuyo hijo se llame juan

```
db.empleados.find({$and:[{'dependientes.nombre':'Juan'}, {apellido:$rege  
x:'Villa*'}]})
```