
Bases de Datos No-SQL

Introducción: Non-SQL DBs

- ◆ Nuevas necesidades de las BDs
 - Escalar en horizontal: crecen en segmentos (procesar en más ordenadores)
 - ejecutarse en cualquier sistema
 - Modelos de datos flexibles
 - Desarrollo más rápido: metodologías Ágiles de desarrollo
 - Bajar el coste
- ◆ Modelos de datos no relacionales
- ◆ No usan SQL: consultas sin formato
- ◆ No hay esquema de la BD predefinido
- ◆ Estructuras de datos más flexibles: varios modelos
 - Atributo – valor, Documentos, XML, Gráficos, Columnas
 - <https://hostingdata.co.uk/nosql-database/>
- ◆ Tienen solo algunas Propiedades ACID (para ganar eficiencia)
 - Transacciones solo a nivel de Documento
 - NO control de Integridad relacional (no claves ajenas)
- ◆ No hay uniones de tablas: documentos integrados

Introducción: MongoDB

- ◆ Es una BD orientada a documentos:
 - No es para ficheros .pdf ni .doc
- ◆ Un *documento* es esencialmente una array asociativo
 - JSON object, PHP Array, Python Dictionar, Ruby Hash, etc
- ◆ Soporta índices primarios y secundarios
- ◆ Almacena datos serializados, formato BSON : ampliación de JSON

BD Relacionales → se “parece” a →	MongoDB
Vista y tabla	Colección
Fila (es plana)	Documento (estructura anidada)
Índice	Índice
Unión	Documento integrado (no hay)
Clave Ajena	Referencia a un objeto
Partición	Fragmento, Segmento (shard)
atributos	Campos (propiedades en objetos .js)

Ejemplo JSON-BSON: un doc. de colección Restaurante

```
{ "direccion": {  
  "edificio": "1007",  
  "coordenadas": [ -73.856077, 40.848447 ],  
  "calle": "Morris Park Ave",  
  "distrito": "10462"  
},  
"zona": "Bronx",  
"tipococina": "Bakery",  
"puntuaciones": [  
  { "fecha": { "$date": 1393804800000 }, "nivel": "A", " valor": 2 },  
  { "fecha": { "$date": 1378857600000 }, "nivel": "A", " valor": 6 },  
  { "fecha": { "$date": 1358985600000 }, "nivel": "A", " valor": 10 },  
  { "fecha": { "$date": 1322006400000 }, "nivel": "A", " valor": 9 },  
  { "fecha": { "$date": 1299715200000 }, "nivel": "B", " valor": 14 } ],  
"nombreRes": "Morris Park Bake Shop",  
"restaurante_id": "30075445"  
}
```

Ejemplo dos docs. en colección *Media*

```
{
  "Type": "CD",
  "Artist": "Nirvana",
  "Title": "Nevermind",
  "Genre": "Grunge",
  "Releasedate": "1991.09.24",
  "Tracklist": [
    {
      "Track" : "1",
      "Title" : "Smells Like Teen Spirit",
      "Length" : "5:02"
    },
    {
      "Track" : "2",
      "Title" : "In Bloom",
      "Length" : "4:15"
    }
  ]
}
```

```
{
  "Type": "Book",
  "Title": "Definitive Guide to MongoDB",
  "ISBN": "987-1-4302-5821-6",
  "Publisher": "Apress",
  "Author": [
    "Hows, David",
    "Plugge, Eelco",
    "Membrey, Peter",
    "Hawkins, Tim" ]
}
```

Una misma colección puede tener varios tipos diferentes:

- de documentos
- de campos con estructuras o simples

Introducción: el Modelo MongoDB

Conectar a un servidor →
Operaciones Objeto CONNECTION

UN SERVIDOR MONGO

(para nuestra Práctica)

miRedSocialAficiones

Bases de Datos →
Operaciones Objeto DATABASE

están compuestas de:

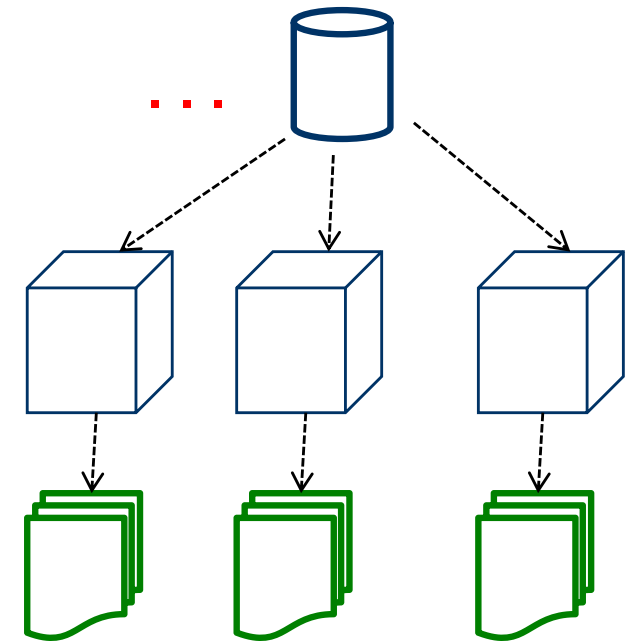
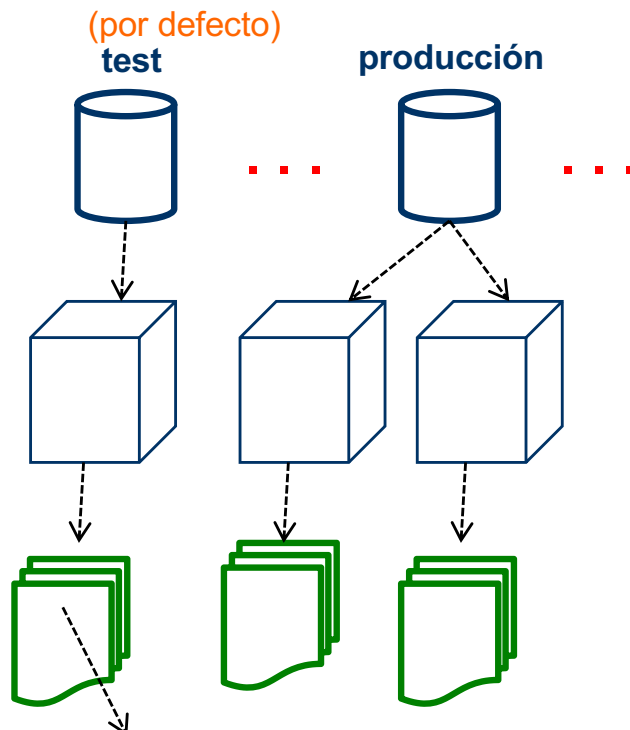
Colecciones →
Operaciones Objeto COLLECTION:
Insert, find, update, aggregate

están compuestas de:

Documentos →

están compuestos de:

**Campos simples y
estructuras complejas:**
“agregados” → →



```
{ "type": "Book",  
  "Title": "Definitive Guide to MongoDB",  
  "ISBN": "987-1-4302-5821-6",  
  "Publisher": "Apress",  
  "Author": [  
    "Hows, David",  
    "Plugge, Eelco",  
    "Membrey, Peter",  
    "Hawkins, Tim" ] }
```

Introducción: Conceptos

- ◆ Una Colección contiene documentos: Objetos
- ◆ Un documento se parece a una fila con sus atributos
- ◆ No tiene esquema, usa sintaxis BSON
- ◆ BSON es una serialización binaria de objetos tipo JSON
 - Mongo entiende JSON: puede importarlos
 - Tamaño máximo: 16 Mb (salvo uso de GridFS, ver mongofiles)
- ◆ Consultas: Sintaxis basada en JavaScript (interpretadas en la shell)
- ◆ Son anidadas: `db.libros.find(envio: {{mensajería:"ups"}});`
 - "envio" es documento integrado (un objeto)
 - Devuelve un "cursor" con el resultado (set de documentos)
- ◆ Cursor: (como en PLSQL)
 - Se puede iterar a lo largo del resultado
 - Es un set de enlaces a objetos (del resultado de la operación)
- ◆ La parte procedimental es JavaScript: shell y en archivos .js
- ◆ Los métodos de la Shell (editor Robo3T) “suelen” llamar a comandos de la BD
 - Inspecciona la función de un método (sin los paréntesis del final): `db.palabras.find` 7

Primeros Pasos en la shell: Objetos Importantes

- las comillas "xx" de Word dan error: valen " y ', son intercambiables
- Todas las operaciones que veremos son métodos de la Shell (el cliente)

Objeto **connection**: Para conectarse al *servidor*, abrir una sesión



```
var myConn = new Mongo("localhost");
```

Objeto **Database**: representa la BD que quieres usar (**solo una BD a la vez**)






- Se crea o selecciona (si ya existía), para trabajar con ella, **"actual"**: `use miBD`
 - la variable db se asocia a esa base de datos: es la BD "actual" o "current"
- para ver el nombre de la BD actual: `db.getName()`
- Asigna una BD a la var. `myDB = myConn.getDB("miBD");`
- Obtener el nombre : `myDB.getName();`
- Ver qué BDs tengo : `show dbs`
 - Una BD se muestra solo si tiene **algún documento**
- Borrar una DB `myDB.dropDatabase();`
- Ver información

```
misEstad = db.stats();
printjson(stats);
```


Conectarse a la BD: el Objeto *connection*

Method	Description
 <code>new Mongo(host:port)</code>	Connects to the MongoDB host and returns a new instance of a <code>Connection</code> object.
 <code>getDB(database)</code>	Returns a <code>Database</code> object specified by the database argument.
<code>setReadPrefMode(mode, tagSet)</code>	Sets the replica set read preference mode. The <code>mode</code> parameter can be <code>primary</code> , <code>primaryPreferred</code> , <code>secondary</code> , <code>secondaryPreferred</code> , or <code>nearest</code> . The <code>tagSet</code> parameter is an array of replica tag sets (see Hour 22, “Database Administration Using the MongoDB Shell”).
<code>getReadPrefMode()</code>	Returns the current read preference mode for MongoDB replica sets.
<code>getReadPrefTagSet()</code>	Returns the current read preference tag set for MongoDB replica sets.
<code>setSlaveOk()</code>	Enables read operations from secondary members in a replica set.

Crear una BD: el Objeto *Database* -I

Method	Description
 <code>addUser (document)</code>	Adds a user to a database based on the user configuration document specified (see Hour 4, “Configuring User Accounts and Access Control”).
 <code>auth (username, password)</code>	Authenticates a user to a database.
<code>changeUserPassword (username, password)</code>	Changes an existing user’s password.
<code>cloneCollection (fromHost, collection, query)</code>	Copies a collection from the fromHost MongoDB server to the current database. The optional query parameter specifies a query used to specify which documents in the connection are cloned.
<code>cloneDatabase (host)</code>	Copies a database from a remote host to the current host.
 <code>commandHelp (command)</code>	Returns help information for a database command.
<code>copyDatabase (srcDatabase, destDatabase, host)</code>	Copies the srcDatabase database name from a remote host to the destDatabase database name on the current host.
 <code>createCollection (name, options)</code>	Creates a new collection. The options parameter enables you to specify collection options, such as when creating a capped collection.
 <code>dropDatabase ()</code>	Removes the current database.
<code>eval (function, arguments)</code>	Sends the JavaScript function specified as the first parameter to the MongoDB server and evaluates it

Crear una BD: el Objeto *Database* -II

`getCollection(collection)`

there. Arguments for the function are specified as subsequent parameters. This enables you to execute code on the server without needing to transfer large amounts of data to the shell client.

Returns a `Collection` object for the collection specified. This is useful if you have a collection name that cannot be accessed using the MongoDB shell syntax, such as collections with a space in the name.

↙ `getCollectionNames()`

Lists all collections in the current database.

↙ `getMongo()`

Returns the `Mongo()` `Connection` object for the current connection.

↙ `getName()`

Returns the name of the current database.

↙ `getSiblingDB(database)`

Returns a `Database` object for another database on the same server.

↙ `help()`

Displays descriptions of common `db` object methods.

`hostInfo()`

Returns a document with information about the system MongoDB runs on.

`logout()`

Ends an authenticated session to this database.

`removeUser(username)`

Removes a user from a database.

Crear una BD: el Objeto *Database* -III

`repairDatabase()`

Runs a repair routine on the current database.

↙ `runCommand(command)`

Runs a database command. This is the preferred method for issuing database commands because it provides a consistent interface between the shell and drivers.

`serverStatus()`

Returns a document that provides an overview of the state of the database process.

↙ `shutdownServer()`

Shuts down the current mongod or mongos process cleanly and safely.

↙ `version()`

Returns the version of the mongod instance.

Primeros Pasos: Objeto *Collection*

- Crear una colección: (el objeto “db” es un objeto Database)

```
myColec = db.getCollection("miColeccion");
```

- Dos modos de acceder a la colección:

Ver el estado `db.miColeccion.stats()`

Otro modo `myColec.stats()`

- Crear varias colecciones y mostrar sus nombres

```
01 conect1 = new Mongo("localhost");
02 use myDB; // crea o asigna myDB como "actual"
   unaDB = conect1.getDB("myDB"); // myDB debe existir
03 collections = unaDB.getCollectionNames();
04 print("Colecciones para Empezar:");
05 printjson(collections);

06 unaDB.createCollection("newCollectionA");
07 unaDB.createCollection("newCollectionB");
08 print("Colecciones Después del create:");
09 collections = unaDB.getCollectionNames();
10 printjson(collections);
```

Primeros Pasos: Objeto *Collection*

- Borrar una colección:

```
coll = db.getCollection("miColeccion")
coll.drop()
```

- Descripción en texto (no es código) de la Estructura de colección: Palabras









```
{ palabra: <una_palabra>,    // Pensar tu "Modelo de Datos"
  primera: <su_primera_letra>, // se ve al final del pdf
  ultima: <su_ultima_letra>,
  tamaño: <numero_de_caracteres>,
  letras:
[<array_de_caracteres_sin_repetir_en_la_palabra>],
  estadis: {
    vocales:<numero_de_vocales>,
    consonantes:<numero_de_consonantes>},
  caractsets: [
    { "tipo": <consonantes_vocales_otro>,
      "caracts":
[<array_de_caracteres_de_un_tipo_en_la_palabra>]}],
    . . .
  ], }
```

Primeros Pasos: Objeto *Collection*







- Un documento de la colección Palabras // **Pensar bien la estructura**

```
{ "_id":{"str":"52d87454483398c8f2429277"}, →lo crea MongoDB  
  "palabra":"the",                      si no lo defines tú  
  "primera":"t",  
  "ultima":"e",  
  "tamaño":3,  
  "letras":["t","h","e"],  
  "estadis": {"vocales":1, "consonantes":2},  
  "caractsets": [  
    {"tipo":"consonantes","caracts":["t","h"]},  
    {"tipo":"vocales","caracts":["e"]} ]  
}
```



Colecciones de Datos: Objeto *Collection* -I

Method	Description
 <code>aggregate()</code>	Provides access to the aggregation pipeline (see Hour 9, “Utilizing the Power of Grouping, Aggregation, and Map Reduce”).
 <code>count()</code>	Returns a count of the number of documents in a collection or those that match a query.
 <code>copyTo(newCollection)</code>	Copies the documents in this collection into a new collection on the same server.
 <code>createIndex()</code>	Builds an index on a collection. Use <code>ensureIndex()</code> .
<code>dataSize()</code>	Returns the size of the collection.
 <code>distinct(field, query)</code>	Returns an array of documents that have distinct values for the specified field, based on the <code>query</code> object.
 <code>drop()</code>	Removes the specified collection from the database.
 <code>dropIndex(index)</code>	Removes the specified index from this collection.
<code>dropIndexes()</code>	Removes all indexes from this collection.
<code>ensureIndex(keys, options)</code>	Creates an index if it does not currently exist (see Hour 21, “Working with MongoDB Data in Node.js Applications”).
 <code>find(query, projection)</code>	Performs a query on a collection and returns a <code>Cursor</code> object (see Hour 6, “Finding Documents in the MongoDB Collection from the MongoDB Shell”).
<code>findAndModify(document)</code>	Atomically modifies and returns a single document (see Hour 8, “Manipulating MongoDB Documents in a Collection”).

Colecciones de Datos: Objeto *Collection-II*

 <code>findOne(query, projection)</code>	Performs a query and returns a single document (see Hour 6).
<code>getIndexes()</code>	Returns an array of documents that describe the existing indexes on a collection.
 <code>group(document)</code>	Provides a basic aggregation that groups documents in a collection by a specific field or fields (see Hour 9).
 <code>insert(document)</code>	Inserts a new document into the collection (see Hour 8).
<code>isCapped()</code>	Returns <code>true</code> if a collection is a capped collection; otherwise, returns <code>false</code> .
 <code>mapReduce(map, reduce, options)</code>	Provides a map reduce data aggregation (see Hour 9).
<code>reIndex()</code>	Rebuilds all existing indexes on this collection.
 <code>remove(query, justOne)</code>	Removes documents from this collection based on the query parameter. If <code>justOne</code> is <code>true</code> , only the first document found is removed.
<code>renameCollection(target, dropTarget)</code>	Changes the name of this collection to the value specified by <code>target</code> . If <code>dropTarget</code> is <code>true</code> , the <code>target</code> collection is dropped before renaming the current collection.
 <code>save(document)</code>	Provides a wrapper around an <code>insert()</code> and <code>update()</code> to insert new documents. If the document is a new document, it is inserted; if it is an existing document, it is updated.

Colecciones de Datos: Objeto *Collection*-III

<code>stats()</code>	Returns a document with stats for the collection.
<code>storageSize()</code>	Returns a document with the total size used by the collection in bytes.
<code>totalSize()</code>	Returns a document with the total size of a collection, including the size of all documents and all indexes on a collection.
<code>totalIndexSize()</code>	Returns a document with the total size used by the indexes on a collection.
 <code>update(query, update, options)</code>	Modifies one or more documents in the collection (see Hour 8).
<code>validate()</code>	Performs diagnostic operations on a collection.

ESPECIALES:

 `getIndexStats()`
 `explain()`




Da estadísticas del árbol B+ del índice

Da información del plan de ejecución de la query

createCollection(nombre,[opciones])

Opciones: es un objeto con algunas de las propiedades indicadas debajo

Ejemplo: `db.createCollection("newCollection", {autoIndexID: false})`

Role	Description
 capped	Boolean. When true, the collection is a capped collection that will not grow bigger than the maximum size specified by the <code>size</code> attribute. Default: false.
autoIndexID	Boolean. When true, an <code>_id</code> field is automatically created for each document added to the collection and an index on that field is implemented. This should be false for capped collections. Default: true.
 size	Specifies the size in bytes for the capped collection. The oldest document is removed to make room for new documents.
 max	Specifies the maximum number of documents allowed in a capped collections. The oldest document is removed to make room for new documents.

Filtros de validación al crear documentos de esta colección (lo vemos en las prácticas)

`db.createCollection("newCollection", validator, validationLevel)`

Colecciones Limitadas (capped)

- ◆ Solo pueden crecer hasta un máximo:
 - `Size` : Tamaño en bytes
 - `max` : número de documentos
- ◆ El documento más antiguo
 - se borra cuando no cabe el nuevo que llega
- ◆ *No existe el borrado de documentos*
- ◆ *Puedes borrar todos, pero hay que volver a crear la colección.*
- ◆ *EJ: un Log de tamaño fijo.*

Manejar Colecciones: Consultas con *find*

Obtener documentos de una colección:

- ◆ `findOne(query, projection)` devuelve el primer documento
 - `query` : objeto con las condiciones
 - `projection` : campos que devolverá
- ◆ `find(query, projection)` Devuelve todos los que cumplen `query`:
 - Mediante un objeto *cursor* que apunta a lista de documentos
- ◆ Projection es: `{ lista de pares campoi: valori, ... }`
 - Donde `valori` es 1 para incluir `campoi` y 0 para excluirlo
 - Solo pueden ponerse todo 1's o todo 0's sin mezclar,
 - excepto el `{_id:0}` que puede ir con 1's
- ◆ Son métodos del objeto *Collection*

Manejar Colecciones: Condiciones del Objeto *query*

- ◆ Contiene operadores con condiciones
- ◆ Ej: documentos con el campo `total` mayor (`$gt`) de 10

y cuyo nombre (`nombre`) es "test "

```
db.miCole({total:{$gt:10}, nombre:'test'}, {nombre,1})
```

si la condición es "igual", se usa ":" en vez de "\$e"

- ◆ EJ: si tenemos:

```
{  
  nombre:"test",  
  datos: { altura:74, ojos:'azul'}  
}
```

Se puede usar la notación de punto "."

```
{datos.ojos:'azul'}
```

- ◆ No hacen falta comillas excepto en las Strings

Parámetro de consulta: operadores de Objeto *Query-I*

Operator	Description
<code>field:value</code>	Matches documents with fields that have a value equal to the value specified. For example: <code>{name:"myName"}</code>
<code>\$gt</code>	Matches values that are greater than the value specified in the query. For example: <code>{size:{\$gt:5}}</code>
<code>\$gte</code>	Matches values that are equal to or greater than the value specified in the query. For example: <code>{size:{\$gte:5}}</code>
<code>\$in</code>	Matches any of the values that exist in an array specified in the query. For example: <code>{name:{\$in:['item1', 'item2']}}</code>
<code>\$lt</code>	Matches values that are less than the value specified in the query. For example: <code>{size:{\$lt:5}}</code>
<code>\$lte</code>	Matches values that are less than or equal to the value specified in the query. For example: <code>{size:{\$lte:5}}</code>
<code>\$ne</code>	Matches all values that are not equal to the value specified in the query.

Parámetro de consulta: operadores de Objeto *Query-II*

	For example: <code>{name:{\$ne:"badName"}}</code>
<code>\$nin</code>	Matches values that do not exist in an array specified to the query. For example: <code>{name:{\$in:['item1', 'item2']}}</code>
<code>\$or</code>	Joins query clauses with a logical OR and returns all documents that match the conditions of either clause. For example: <code>{\$or:[{size:{\$lt:5}},{size:{\$gt:10}}]}</code>
<code>\$and</code>	Joins query clauses with a logical AND and returns all documents that match the conditions of both clauses. For example: <code>{\$and:[{size:{\$lt:5}},{size:{\$gt:10}}]}</code>
<code>\$not</code>	Inverts the effect of a query expression and returns documents that do not match the query expression. For example: <code>{\$not:{size:{\$lt:5}}}</code>
<code>\$nor</code>	Joins query clauses with a logical NOR and returns all documents that fail to match both clauses. For example: <code>{\$nor:{size:{\$lt:5}},{name:"myName"}}</code>
<code>\$exists</code>	Matches documents that have the specified field. For example: <code>{specialField:{\$exists:true}}</code>

Parámetro de consulta: operadores de Objeto *Query-III*

\$type	Selects documents if a field is of the specified BSON type number (Table 1.1 in Hour 1, “Introducing NoSQL and MongoDB,” lists the BSON type numbers). For example: <pre>{specialField:{ \$type:<BSONtype>}}</pre>
\$mod	Performs a modulo operation on the value of a field and selects documents with a specified result. The value for the modulo operation is specified as an array—the first number is the number to divide by, and the second number is the remainder. For example: <pre>{number:{ \$mod: [2,0] }}</pre>
\$regex	Selects documents in which values match a specified regular expression. For example: <pre>{myString:{ \$regex:'some.*exp' }}</pre>
\$all	Matches arrays that contain all elements specified in the query. For example: <pre>{myArr:{ \$all:['one','two','three'] }}</pre>
\$elemMatch	Selects documents if an element in the array of subdocuments has fields that match all the specified \$elemMatch condition. For example: <pre>{myArr:{ \$elemMatch:{value:{ \$gt:5 },size:{ \$lt:3 } } }}</pre>
\$size	Selects documents if the array field is a specified size. For example: <pre>{myArr:{ \$size:5 }}</pre>

Manejar Colecciones con método *aggregate*



- ◆ El método **aggregate** es del objeto Collection con varios parámetros:

`aggregate([operador1, operador2, ...])`

- ◆ Es el efecto de “pipeline” :
 - Se aplica *operador1* a toda la colección dando un resultado
 - Al resultado del operador1 se aplica el *operador2*
- ◆ Los posibles *operadores estándar* están en las siguientes diapositivas
- ◆ Un *operador* también puede ser una función del usuario
- ◆ El \$ delante del nombre de un campo es para diferenciarlo de un string
- ◆ Ejemplo con tres operadores:

```
db.palabras.aggregate([
  { $match: { primera:{$gte:"t"} } }, //filtra docs >= "t"
  { $group: { _id: "$primera", // agrupa por ese campo
    totalcaracteres: { $sum: "$tamaño" }, //campo agreg
    totalpalabras: { $sum: 1 } } }, // otro campo agreg
  {$sort: { _id:1}} ]) // por "primera" ascendente
```

Parametro *operador* del método *aggregate*

Operator	Description
 \$project	<p>Reshapes the documents by renaming, adding, or removing fields. You can also recompute values and add subdocuments. For example, the following will include title and exclude name:</p> <pre>{ \$project: { title: 1, name: 0 } }</pre> <p>Consider an example of renaming name to title:</p> <pre>{ \$project: { title: "\$name" } }</pre> <p>Now consider an example of adding a new field total and computing its value from price and tax fields:</p> <pre>{ \$project: { total: { \$add: ["\$price", "\$tax"] } } }</pre>
 \$match	<p>Filters the document set using the query operators discussed in previous hours. For example:</p> <pre>{ \$match: { value: { \$gt: 50 } } }</pre>
\$limit	<p>Restricts the number of documents that can be passed to the next pipe in the aggregation. For example:</p> <pre>{ \$limit: 5 }</pre>
\$skip	<p>Specifies the number of documents to skip before processing the next pipe in the aggregation. For example:</p> <pre>{ \$skip: 10 }</pre>

Parametro *operador* del método *aggregate*





- ➔ `$unwind` The value of `$unwind` should be an array field name (you must prefix the array field name with a `$` so that it is read as a field name and not a string). The array will be split, and a separate document will be created for each value. For example:

```
{ $unwind: "$myArr" }
```
 - ➔ `$group` Groups the documents into a new set of documents for the next level in the pipe. The fields of the new object must be defined in the `$group` object. You can also apply group expression operators to the multiple documents in the group. For example, to sum the `value` field:

```
{ $group: { set_id: "$o_id", total: { $sum: "$value" } } }
```
 - ➔ `$sort` Sorts the documents before passing them on to the next pipe in the aggregation. The sort specifies an object with `field:<sort_order>` properties, where `<sort_order>` is 1 for ascending and -1 for descending. For example:

```
{ $sort: { name: 1, age: -1 } }
```
-

Operadores de \$group en el método *aggregate*

Operator	Description
 \$addToSet	Returns an array of all the unique values for the selected field for each document in that group. For example: <code>colors: { \$addToSet: "\$color" }</code>
\$first	Returns the first value for a field in a group of documents. For example: <code>firstValue: { \$first: "\$value" }</code>
\$last	Returns the last value for a field in a group of documents. For example: <code>lastValue: { \$last: "\$value" }</code>
 \$max	Returns the highest value for a field in a group of documents. For example: <code>maxValue: { \$max: "\$value" }</code>
\$min	Returns the lowest value for a field in a group of documents. For example: <code>minValue: { \$min: "\$value" }</code>
\$avg	Returns an average of all the values for a field in a group of documents. For example: <code>aveValue: { \$avg: "\$value" }</code>
 \$push	Returns an array of all values for the selected field for each document in that group of documents. For example: <code>username: { \$push: "\$username" }</code>
 \$sum	Returns the sum of all the values for a field in a group of documents. For example: <code>total: { \$sum: "\$value" }</code>

Actualizar una Colección con método *update* : Formato

- ◆ El método `update` es del objeto `Collection` con varios parámetros:

`update(query, update, upsert, multi, writeConcern)`

- ◆ *query*: (la condición) es un obj. que identifica qué docs quieres cambiar

- ◆ *update*: objeto que indica qué campos y valores se han de cambiar

- usa operadores de la tabla de operadores del objeto `update`

- ◆ *upsert*: cuando no se encuentra ningún doc. que cumpla la query, ...

- si *true*: crea un doc con los datos
 - si *false*: no crea nada

- ◆ *multi*: cuántos docs actualiza?

- si *true*, actualiza todos los docs que cumplan query
 - si *false*, solo actualiza el primero

- ◆ Ejemplo simple:

```
----- query ----- , ----- update -----, -upsert-, -multi-  
update({categoria:"nueva"}, {$set:{categoria:"vieja"}}, false, true);
```

- ◆ Evita que otro proceso escriba mientras se ejecuta esta operación

```
update({categoria:"nueva", $isolated:1}, // nivel de aislamiento de una Transacción  
      {$set:{categoria:"vieja"}}, false, true);
```


Actualiza Colecciones: el parámetro *update* –objeto–

- ◆ Debemos indicar: qué operación y sobre qué campo
- ◆ Se pueden crear objetos simples **update** (parámetro del método **update**):

```
{
  <operator>: {<field_operation>, <field_operation>, . . .},
  <operator>: {<field_operation>, <field_operation>, . . .}
  . . .
}
```

- ◆ Permite muchas operaciones en un solo objeto update,
- ◆ Ejemplo: dada la estructura de documento de colección “colPruebas”:

```
{
  nombre: "myName",
  contadorA: 0,
  contadorB: 0,
  dias: ["Monday", "Wednesday"],
  puntuaciones: [ {id:"test1", puntua:94}, {id:"test2",
puntuacion:85}, {id:"test3", puntua:97}]
}
```

Actualizaciones con el parámetro *update* –objeto-

- ◆ Si quieres hacer estas operaciones en una **sola Transacción**:
 - Incrementar el contadorA en 5 unidades
 - Incrementar el contadorB en 1 unidades
 - Poner el nombre a "miNombre"
 - Añadir "Friday" al array de dias
 - Ordenar las puntuaciones por el campo "puntua" con valores crecientes

- ◆ El parámetro *update* a utilizar es este:

```
{  
  $inc:{contadorA:5, contadorB:1},  
  $set:{nombre:"miNombre"},  
  $push{dias:"Friday"},  
  $sort:{puntua:1}  
}
```


Actualizar una Colección con el método *update*

- ◆ Ejemplo de la colección colPruebas: (Instrucción UPDATE completa)

```
colPruebas.update(  
  { $and:[{nombre: "miNombre"},  
    { primera: "q"},{ultima:"y"}]},    ← la query  
  { $inc:{contadorA:5, contadorB:1},  
    $set:{nombre:"tuNombre"},  
    $push{dias:"Friday"},  
    $sort:{puntua:1}},                ← el param. update  
  false, false)                      ← upsert y multi  
Si no existe      Aplica a  
No actualiza      Un solo doc.
```

Ejemplo con upserting en otra colección: penúltimo parámetro = true

```
coleColores.update({color:"azure"}, {$set:{red:0,  
green:127, blue:255}}, true, false);  
Misma operación a  
varios atributos  
Aplica a  
Un solo doc.
```

Operadores del parámetro *update* –un objeto–

Operator	Description
\$inc	Increments the value of the field by the specified amount. Operation format: field:inc_value
\$rename	Renames a field. Operation format: field:new_name
\$setOnInsert	Sets the value of a field when a new document is created in the update operation. Operation format: field:value
\$set	Sets the value of a field in an existing document. Operation format: field:new_value
\$unset	Removes the specified field from an existing document. Operation format: field:""
\$	Acts as a placeholder to update the first element that matches the query condition in an update.
Operaciones sobre Arrays	
\$addToSet	Adds elements to an <u>existing array</u> only if they do not already exist in the set. Operation format: array_field:new_value
\$pop	Removes the first or last <u>item of an array</u> . If the pop_value is -1, the first element is removed. If the pop_value is 1, the last element is removed. Operation format: array_field:pop_value

Operadores del parámetro *update* –un objeto–

Más Operaciones sobre Arrays

\$pullAll	Removes multiple values from an array. The values are passed in as an array to the field name. Operation format: <code>array_field:[value1, value2, ...]</code>
\$pull	Removes items from an array that match a query statement. The query statement is a basic query object with field names and values to match. Operation format: <code>array_field:[<query>]</code>
\$push	Adds an item to an array. Simple array format: <code>array_field:new_value</code> Object array format: <code>array_field:{field:value}</code>
\$each	Modifies the \$push and \$addToSet operators to append multiple items for array updates. Operation format: <code>array_field:{\$each:[value1, ...]}</code>
\$slice	Modifies the \$push operator to limit the size of updated arrays.
\$sort	Modifies the \$push operator to reorder documents stored in an array. Operation format: <code>array_field:{\$slice:<num> }</code>
\$bit	Performs bitwise AND OR updates of integer values. Operation format: <code>integer_field:{and:<integer> }</code> <code>integer_field:{or:<integer> }</code>

Otras actualizaciones: *insert* y *remove*

- ◆ Ejemplo de la colección colPruebas:

```
miColecc.insert ({nombre: "myName",  
                  contadorA:5, contadorB:1,  
                  dias: ["Monday", "Wednesday"]});
```

→ Qué "_id" tendrá?
Lo inventa MongoDB

Borra todos los docs

```
miColec.remove({primera:'a'}, false);
```

Borra el primer doc que cumple la condición

```
miColecc.remove({primera:'a'}, true);
```

Nivel de Garantía en la escritura: “Write Concern”

- ◆ “Write Concern” es un nivel de garantía frente a la pérdida de datos
- ◆ Determina cuando responde MongoDB después de haber escrito a disco
- ◆ Se decide por el usuario en el momento de la conexión
- ◆ Si es fuerte: informa cuando ya ha terminado la escritura en disco
 - Incluso en los servidores réplica → → ¿Qué son?
- ◆ Si es débil: informa cuando está planificada la escritura
- ◆ Estos son los niveles: Velocidad frente a Fiabilidad: según aumenta el Level

Level	Description
-1	Network errors are ignored.
0	No write acknowledgment is required.
1	Write acknowledgment is requested.
2	Write acknowledgment is requested across the primary server and one secondary server in the replica set.
majority	Write acknowledgment is requested for a majority of servers in the replica set.

Obtener el estado de un Escritura: getLastError

Dos modos: (comando de la BD ; método de la Shell, es un envoltorio del otro)

◆ (BD) El comando getLastError solo devuelve:

- null si la última operación devolvió true, o bien
- un string con el mensaje de error que ha sucedido
- Ejemplo con la BD wordsDB:

```
wordsColl.insert({word:"the"});  
results = wordsDB.runCommand( { getLastError: 1}); // <--- uso comando BD  
if(results.err){  
    print("ERROR: " + results.err); }
```

◆ (shell) El método getLastError() del obj. Database da mucha más información :

- el estado, la operación que lo produjo ,
- número de docs. modificados, el mensaje de error
- y otras propiedades que están en la tabla
- Ejemplo con la BD wordsDB

```
wordsColl.insert({word:"the"});  
lastError = wordsDB.getLastError(); // <--- uso del método de la shell  
if(lastError){  
    print("ERROR: " + lastError); }
```

Propiedades del objeto devuelto por getLastError

Option	Description
ok	Boolean. true when the getLastError command completes successfully.
err	String describing the error. null if no error occurred on the last request.
code	Reports the preceding operation's error code.
connectionId	The ID of the connection. (un entero)
lastOp	The optime timestamp in the oplog of the change for request to a replica set member when the last operation was a write or update.
n	Reports the number of documents updated or removed for update and move operations.
updateExisting	Boolean. true if an update affects at least one document and does not result in an upsert.
upserted	ObjectId of the new object if an update request resulted in an insert.
wnote	Boolean. true if the error relates to a write concern.
wtimeout	Boolean. true if the getLastError timed out because of the wtimeout setting.
waited	Number of milliseconds waited before timeout if the last operation timed out because of the wtimeout setting.
wtime	Number of milliseconds spent waiting for the preceding operation to complete. If getLastError timed out, wtime and waited should be the same.

Códigos de Tipos de Datos

Type	Number	→ USO: en scripts
Double	1	
String	2	
Object	3	
Array	4	
Binary data	5	
Object ID	7	
Boolean	8	
Date	9	
Null	10	
Regular expression	11	
JavaScript	13	
Symbol	14	
JavaScript (with scope)	15	
32-bit integer	16	
Timestamp	17	
64-bit integer	18	
Min key	255	
Max key	127	

Operadores Aritméticos

(Parte procedimental en JavaScript)

Para $y = 4$ tenemos estos resultados

Operator	Description	Example	Resulting x
+	Addition	$x=y+5$	9
		$x=y+"5"$	"45"
		$x="Four"+y+"4"$	"Four44"
-	Subtraction	$x=y-2$	2
++	Increment	$x=y++$	4
		$x=++y$	5
--	Decrement	$x=y--$	4
		$x=--y$	3
*	Multiplication	$x=y*4$	16
/	Division	$x=10/y$	2.5
%	Modulous (remainder of division)	$x=y\%3$	1

Operadores de Asignación

Operator	Example	Equivalent Arithmetic Operators	Resulting x
=	x=5	x=5	5
+=	x+=5	x=x+5	10
-=	x-=5	x=x-5	5
=	x=5	x=x*5	50
/=	x/=5	x=x/5	2
%=	x%=5	x=x%5	0

Operadores de Comparación

Para x = 10 tenemos estos resultados

Operator	Description	Example	Result
==	Is equal to (value only)	x==8	false
		x==10	true
===	Both value and type are equal	x===10	true
		x==="10"	false
!=	Is not equal	x!=5	true
!==	Both value and type are not equal	x!== "10"	true
		x!==10	false
>	Is greater than	x>5	true
>=	Is greater than or equal to	x>=10	true
<	Is less than	x<5	false
<=	Is less than or equal to	x<=10	true

Operadores de Comparación Lógicos

Operator	Description	Example	Result
&&	and	(x==10 && y==5)	true
		(x==10 && y>x)	false
	or	(x>=10 y>x)	true
		(x<10 && y>x)	false
!	not	!(x==y)	true
		!(x>y)	false
	mix	(x>=10 && y<x x==y)	true
		((x<y x>=10) && y>=5)	true
		(!(x==y) && y>=10)	false
&&	and	(x==10 && y==5)	true
		(x==10 && y>x)	false

Condicionales : IF, CASE

- ◆ If (miNombre == "pepito") { hacer_algo() }
- ◆ If (miNombre == "pepito")
 { hacer_algo() }
else { hacer_otra_cosa() }
- ◆ If (miNombre == "pepito") { hacer_algo() }
 else if (miNombre == "Juanito" { hacer_juanito() }
 else { hacer_en_caso_contrario() }
- ◆ switch(expression){
 case value1:
 <code to execute> break;
 case value2:
 <code to execute> break;
 default:
 <code to execute if not value1 or value2>
}

Bucles: DO, FOR y WHILE

```
var misDias = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
```

```
var i=0;
```

```
do{
```

```
    var undia=misDias[i++];
```

```
    print("It's " + undia + "\n");
```

```
} while (undia != "Wednesday");
```

→ var misDias -> indica ámbito local (USA ESTE SIEMPRE!!)

→ misDias -> (sin "var") indica ámbito global

```
for (var x=1; x<=3; x++){
```

```
    for (var y=1; y<=3; y++){
```

```
        print(x + " X " + y + " = " + (x*y) + "\n");
```

```
    }
```

```
} // → también existe "while (condición) { . . . }
```

```
var misDias = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
```

```
for (var hoy in misDias){
```

```
    print("It's " + misDias[hoy] + "\n");
```

```
}
```

→ Interrupir ejecución:

- break : parar esa interacción y salir del bucle
- continue: parar esa iteración y seguir con la siguiente iteración

Funciones

```
function saludar(name, city){
    var retStr = "";
    retStr += "Hello " + name + "\n";
    retStr += "Welcome to " + city + "!";
    return retStr;
}
var unSaludo = saludar("Pepito", "Madrid");
print(unSaludo);
```

// funciones sin nombre para usarlas solo una vez

```
function calcular(num1, num2, calcFunction){
    return calcFunction(num1, num2);
}
```

→ Ejemplo de uso: `print(calcular(5, 10, function(n1, n2){ return n1 + n2; }));`

Ejecución de Scripts:

- Se pueden salvar las instrucciones en fichero nombre.js
- Y ejecutar desde ventana CMD de windows: `mongo nombre.js`
- Desde dentro del editor: `load nombre.js`

Manipular Strings

Method	Description
<code>charAt(index)</code>	Returns the character at the specified index.
<code>charCodeAt(index)</code>	Returns the Unicode value of the character at the specified index.
<code>concat(str1, str2, ...)</code>	Joins two or more strings, and returns a copy of the joined strings.
<code>fromCharCode()</code>	Converts Unicode values to actual characters.
<code>indexOf(subString)</code>	Returns the position of the first occurrence of a specified <code>subString</code> value. Returns -1 if the substring is not found.
<code>lastIndexOf(subString)</code>	Returns the position of the last occurrence of a specified <code>subString</code> value. Returns -1 if the substring is not found.
<code>match(regex)</code>	Searches the string and returns all matches to the regular expression.
<code>replace(subString/regex, replacementString)</code>	Searches the string for a match of the substring or regular expression and replaces the matched substring with a new substring.

Manipular Strings

<code>search(regex)</code>	Searches the string based on the regular expression and returns the position of the first match.
<code>slice(start, end)</code>	Returns a new string that has the portion of the string between the <code>start</code> and <code>end</code> positions removed.
<code>split(sep, limit)</code>	Splits a string into an array of substrings based on a separator character or regular expression. The optional <code>limit</code> argument defines the maximum number of splits to make, starting from the beginning.
<code>substr(start, length)</code>	Extracts the characters from a string, beginning at a specified <code>start</code> position, through the specified <code>length</code> of characters.
<code>substring(from, to)</code>	Returns a substring of characters between the <code>from</code> and <code>to</code> index.
<code>toLowerCase()</code>	Converts the string to lower case.
<code>toUpperCase()</code>	Converts a string to upper case.
<code>valueOf()</code>	Returns the primitive string value.

Manipular Strings

Escape	Description	Example	Output String
\'	Single quote mark	"couldn\'t be"	couldn't be
\"	Double quote mark	"I \"think\" I \"am\""	I "think" I "am"
\\	Backslash	"one\\two\\three"	one\two\tthree
\n	New line	"I am\nI said"	I am I said
\r	Carriage return	"to be\r or not"	to be or not
\t	Tab	"one\ttwo\tthree"	one two three
\b	Backspace	"correctoin\b\b\bion"	correction
\f	Form feed	"Title A\fTitle B"	Title A then Title B

Manipular Arrays

Method	Description
<code>concat(arr1, arr2, ...)</code>	Returns a joined copy of the array and the arrays passed as arguments.
<code>indexOf(value)</code>	Returns the first index of the value in the array or -1 if the item is not found.
<code>join(separator)</code>	Joins all elements of an array separated by the separator into a single string. If no separator is specified, a comma is used.
<code>lastIndexOf(value)</code>	Returns the last index of the value in the array or -1 if the value is not found.
<code>pop()</code>	Removes the last element from the array and returns that element.
<code>push(item1, item2, ...)</code>	Adds one or more new elements to the end of an array and returns the new length.
<code>reverse()</code>	Reverses the order of all elements in the array.
<code>shift()</code>	Removes the first element of an array and returns that element.
<code>slice(start, end)</code>	Returns the elements between the start and end index.
<code>sort(sortFunction)</code>	Sorts the elements of the array. The <code>sortFunction</code> is optional.
<code>splice(index, count, item1, item2...)</code>	At the <code>index</code> specified, <code>count</code> number items are removed. Then any optional items passed in as arguments are inserted at <code>index</code> .
<code>toString()</code>	Returns the string form of the array.
<code>unshift()</code>	Adds new elements to the beginning of an array and returns the new length.
<code>valueOf()</code>	Returns the primitive value of an array object.





Arrays:añadir/eliminar elementos

→ 1º Elemento del índice es 0




Statement	Value of x	Value of arr
<code>var arr = [1,2,3,4,5];</code>	undefined	1,2,3,4,5
<code>var x = 0;</code>	0	1,2,3,4,5
<code>x = arr.unshift("zero");</code>	6 (length)	zero,1,2,3,4,5
<code>x = arr.push(6,7,8);</code>	9 (length)	zero,1,2,3,4,5,6,7,8
<code>x = arr.shift();</code>	zero	1,2,3,4,5,6,7,8
<code>x = arr.pop();</code>	8	1,2,3,4,5,6,7
<code>x=arr.splice(3,3,"four", "five","six");</code>	4,5,6	1,2,3,four,five,six,7
<code>x = arr.splice(3,1);</code>	four	1,2,3,five,six,7
<code>x = arr.splice(3);</code>	five,six,7	1,2,3

Métodos del objeto *Cursor-I*

(un *cursor* es devuelto por `find` y `aggregate`)

Method	Description
<code>batchSize(size)</code>	Specifies the number of documents MongoDB returns to the client in a single network response. The default is 20.
<code>count()</code>	Returns a count of the documents represented by the <code>Cursor</code> object.
 <code>explain()</code>	Returns a document that describes the query execution plan, including index usage, that will be used to return the query results. This is a great way to troubleshoot slower queries or optimize requests.
 <code>forEach(function)</code>	Iterates though each document represented in the <code>Cursor</code> and applies the specified JavaScript function. The JavaScript function should accept a document as the only parameter. For example: <pre>myColl.find().forEach(function(doc) { print("name: " + doc.name); });</pre>
 <code>hasNext()</code>	Used while iterating through a cursor using <code>next()</code> . Returns <code>true</code> if the cursor has additional documents and can be iterated.
<code>hint(index)</code>	Forces MongoDB to use one or more specific indexes for a query. The index can be a string such as <pre>hint("myIndex_1")</pre> It can also be a document with the index name as the property and 1 as the value. For example: <pre>hint({ myIndex: 1})</pre>
<code>limit(maxItems)</code>	Constrains the size of a cursor's result set to the number specified in <code>maxItems</code> .
 <code>map(function)</code>	Iterates through each document in the <code>Cursor</code> and applies a function. The return value of each iteration is added to an array that is returned. For example:

Métodos del objeto *Cursor*-II

	<pre>names = myColl.find().map(function(doc) { return doc.name; });</pre>
<code>max(indexBounds)</code>	Specifies maximum values for fields in documents that should be returned by the <code>Cursor</code> . For example: <pre>max({ height: 60, age: 10 })</pre>
<code>min()</code>	Specifies minimum values for fields in documents that should be returned by the <code>Cursor</code> . For example: <pre>min({ height: 60, age: 10 })</pre>
<code>next()</code>	Returns the next document from the <code>Cursor</code> and increments the iteration index. 
<code>objsLeftInBatch()</code>	Returns the number of documents left in the current <code>Cursor</code> batch. As you iterate past the last item in the batch, a new request is made to the server to retrieve the next batch of documents.
<code>readPref(mode, tagSet)</code>	Specifies a read preference to a <code>Cursor</code> to control how the client directs queries to a replica set.
<code>size()</code>	Returns a count of the documents in the <code>Cursor</code> after applying <code>skip()</code> and <code>limit()</code> methods.
<code>skip(n)</code>	Returns another <code>Cursor</code> that begins returning results only after skipping <i>n</i> number of documents.
<code>snapshot()</code>	Forces the <code>Cursor</code> to use the index on the <code>_id</code> field, which ensures that the <code>Cursor</code> returns each document only once.
<code>sort(sortObj)</code>	Returns results ordered according to the specification in <code>sortObj</code> . The <code>sortObj</code> object should include the fields in order of sort and specify a value of -1 for descending or 1 for ascending. For example: <pre>sort({ name: 1, age: -1 })</pre> 
<code>toArray()</code>	Returns an array of JavaScript objects representing all documents returned by the <code>Cursor</code> . 

Planear tu Modelo de Datos: Analiza tus Datos

Escribe ejemplos con datos concretos:

- ◆ *¿Qué objetos básicos necesito?*
- ◆ *¿Qué atributos tiene cada tipo de objeto?*
- ◆ *¿Qué relaciones hay entre esos objetos: 1 a 1, 1 a N, N a N*

Todos los tipos de documentos deben tener algunos campos comunes para búsquedas globales a la Colección

→ Un D. Clases ayuda

Depende mucho del uso que vayas a hacer:

- ◆ *¿Con que frecuencia ... :*
 - *se añaden objetos nuevos?*
 - *se borran objetos?*
 - *se modifican objetos?*
 - *se acceden objetos?*
- ◆ *¿Cómo serán las consultas de un objeto...:*
 - *¿desde la clave?*
 - *¿desde qué campos?*
 - *¿Comparando con otros campos?*
- ◆ *¿Cómo serán las consultas de un grupos de tipos de objeto?*
- ◆ ***Escribe qué consultas necesitas hacer a esta BD***

Planear tu Modelo de Datos: Normalizar Documentos

- ◆ Significa algo muy diferente que en las DB relacionales:
 - Organizar documentos y colecciones para
 - minimizar la redundancia y las dependencias
 - tener una sola copia de un objeto y se referencie desde varios documentos
- ◆ Ej.: varios Clientes compran en la misma tienda
- ◆ Consiste en separar determinadas propiedades de un objeto
 - y almacenarlas como documentos (objetos) aparte en otra colección.
 - Se hace cuando las propiedades son comunes a otros objetos, luego . . .
 - entre el objeto y esas propiedades hay relaciones de 1 a N y de N a N
- ◆ A tener en cuenta : NO es atómica la actualización de
 - un objeto Normalizado (compuesto usando referencias)
- ◆ Si necesitas que sea atómica, entonces,
 - desnormaliza el objeto incluyendo todas las propiedades en el mismo objeto
 - y así tratar su contenido en una sola actualización.

Planear tu Modelo de Datos: DesNormalizar Documentos

- ◆ Desnormalizar documentos en MongoDB consiste
 - en incluir un objeto (alguna propiedad) dentro de otro.
- ◆ Es útil cuando sucede alguna de estas situaciones.
 - para objetos con relaciones 1 a 1.
 - relaciones 1 a N pero son pequeños.
 - se actualizan muy poco
- ◆ El beneficio es la eficiencia:
 - En una sola consulta obtienes todas las propiedades de ese objeto
 - En la actualización: la unicidad
- ◆ Ej: un cliente tiene dirección de su casa y dirección de la oficina,
 - que a su vez tienen calle, número, ciudad y distrito.

Funciones Avanzadas: MongoDB

- ◆ Escala fácilmente en sentido horizontal (+Alto rendimiento)
 - Es transparente a la aplicación
 - La partición y equilibrado es automático por rangos de claves
 - Se llaman Colecciones Segmentadas (sharded)
 - Cada segmento (shard) permite replicas iguales para seguridad.
- ◆ Fácil sets de replica: (DB duplicada para protección de fallos de discos)
 - Tiene driver que : lee y escribe síncrona en primario
 - lee solo del secundario
 - la replica se hace asíncrona en secundarios
 - Si cae uno, cambia primario a otro
- ◆ Índices, Lenguaje de Consultas a medida, Shell interactiva
- ◆ Framework de Agregación: Tuberías, Mapea/reduce
- ◆ MongoDB tiene arquitectura cliente – servidor
 - Cliente: La shell es un interprete de Javascript

Otras non-sql BDs

- ◆ Basado en datos con muchas relaciones
 - Adecuadas para datos sociales hiperconectados, geolocalización
 - Un diagrama
 - Una BD : [Neo4j](#)
- ◆ Basadas en lógica
 - Hechos + Reglas + motor inferencia
 - Una BD: Datalog
- ◆ Otras:
 - Multidimensionales
 - Cloud
 - XML

Bibliografía

- ◆ Sams Teach Yourself NoSQL with MongoDB in 24 Hours . Brad Dayley
ISBN-10: 0-13-384442-0. 2014 (en Safari de la FDI)
- ◆ MongoDB Basics Hows, David. 2014. (en Safari de la FDI)
- ◆ MongoDB Cookbook. Nayak, Amol. Packt Publishing, Limited, 2014. (en Safari de la FDI)
- ◆ 50 Tips and Tricks for MongoDB Developers. Chodorow, Kristina.
O'Reilly Media, Incorporated, 2011. (en Safari de la FDI)
- ◆ Introducción a las bases de datos NoSQL usando MongoDB. Antonio Sarasa. Barcelona : Editorial UOC, 2016