

NoSQL
Not Only SQL

APACHE
HBASE

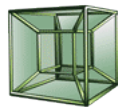
 **Cassandra**



CouchDB
relax

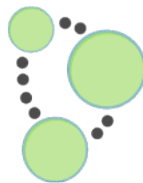


riak



mongoDB

HYPERTABLE INC



Neo4j



redis

En els últims anys estan eixint noves Bases de Dades que estan fugint del que ells anomenen "rigidesa" de les Bases de Dades Relacionals, on les dades estan molt estructurades: taules amb camps perfectament definits, i amb les restriccions que hem vist que es poden crear.

I si vulguérem guardar dades poc estructurades? Per exemple dels usuaris d'una determinada organització, que no sempre ens guardem la mateixa informació: unes vegades ens guardem el telèfon, altres no, altres ens guardem més d'un telèfon; altres vegades ens volem guardar la seua pàgina web, altres vegades les aficions, ... En el model relacional, molt estructurat, hauríem de preveure tots els camps necessaris, i en cas de que sorgira la necessitat d'una nova informació hauríem de canviar l'estructura de la taula.

O un altre exemple, volem guardar senzillament unes quantes "variables" de forma permanent. Podria ser per exemple guardar les preferències de l'usuari en una aplicació per a dispositius mòbils. Si ho dissenyàrem en una taula, seria una taula amb molts camps, però que després només hi haurà una fila, la de l'usuari.

Un altre exemple seria guardar informació de documents XML o JSON, els estàndars d'intercanvi d'informació. En el Model Relacional hauríem de guardar el document XML o JSON pràcticament en un camp, tot junt, i seria costós buscar dins de l'estructura XML o JSON.

Per a salvar aquestos inconvenients mencionats han surgit una sèrie d'iniciatives noves que s'han agrupat en el terme **NoSQL** (Not Only SQL). Són en definitiva Bases de Dades que no estan basades en el Model Relacional, i que en determinades ocasions poden ser més eficients que les Bases de Dades Relacionals precisament per fugir de la rigidesa que proporciona aquest model, amb dades tan ben estructurades. El terme clau és "en determinades ocasions", és a dir, per a guardar un determinat tipus d'informació. Així ens trobarem distints tipus de Bases de Dades NoSQL, depenent del tipus d'informació que vulguen guardar:

- **Bases de Dades Orientades a Objectes**, per a poder guardar objectes, com per exemple **DB4O**
- **Bases de Dades Orientades a Documents** (o simplement Bases de Dades Documentals), per a guardar documents de determinats tipus: XML, JSON, ... Per exemple **eXist** que guarda documents XML, o **MongoDB** que guarda la informació en un format similar a JSON.
- **Bases de Dades Clau-Valor**, per a guardar informació de forma molt senzilla, guardant únicament la clau (el nom de la propietat) i el seu valor
- **Bases de Dades Orientades a Grafs**, per a guardar estructures com els grafs, on hi ha una sèrie de nodes i arestes que comunicarien (relacionarien) els nodes
- Altres tipus com les **Bases de Dades Multivalor**, les **Bases de Dades Tabulars**, **Bases de Dades Orientades a Columna**, ...

En aquest tema només vuerem les Bases de Dades **Clau-Valor**, per la seua senzillesa, i una altra Base de Dades, **MongoDB**, per la seua utilització actual.

Segurament de tots els tipus de Bases de Dades NoSQL existents, la de més fàcil comprensió és la **Base de Dades Clau-Valor**. En ella s'aniran guardant parelles clau-valor, és a dir, el nom d'una determinada propietat i el seu valor. La clau ha de ser única, és a dir, no es pot repetir ja que en cas contrari no es podria tornar a recuperar.

Per tant no hi ha definició de taules ni cap altra estructura; es van guardant parelles clau-valor i prou. En el moment de recuperar la informació veurem que podrem obtenir el contingut d'una clau, o el de més d'una al mateix temps.

L'exemple que veurem de Base de Dades Clau-Valor és **Redis**, molt famosa per a seua potència i eficiència.

La clau sempre és de tipus String, i com ja hem comentat no poden haver dues claus iguals. En canvi en Redis els valors poden ser de 5 tipus diferents:

- **Cadenes de caràcters (String)**. Per exemple: **nom_1 --> Albert**
- **Registres (Hashes)** que seran claus que tindran subcamp (sub-clau). Per exemple: **empleat_1 --> [nom="Albert" , departament="10" , sou="1000.0"]**
- **Llistes (Lists)** que són conjunts ordenats de valors. Per exemple: **llista_1 --> { "Primer" , "Segon" , "Tercer" }**
- **Conjunts (Sets)** són conjunts desordenats de valors. No importa el seu ordre, i de fet serà impredecible l'ordre amb el qual els torna Redis. Per exemple: **colors --> { "Blau" , "Verd" , "Roig" }**
- **Conjunts Ordenats (Sorted Sets)**. Ja veurem la diferència entre llistes i conjunts ordenats.

Algunes característiques de Redis són:

- És una arquitectura client-servidor
- És extraordinàriament eficient quan es pot carregar tota en memòria, encara que si no pot carregar-la també funcionarà de forma molt ràpida. I a més manté una sincronització constant a disc per a fer les dades persistents. Aquesta tasca la fa en segon pla, de manera que no afecta al servei.
- Per a poder suportar ratios de lectura molt alts fa una replicació master/slave, és a dir que pot haver més d'un servidor, i un és el que actua de master i els altres són rèpliques del primer. El slave rep una còpia inicial de la Base de Dades sencera. A mida que es van realitzant escriptures en el master, es van enviant a tots els slaves connectats. Els clients es poden connectar als diferents servidors, bé siga al master o als slaves, sense haver de carregar sempre al master.

Redis està construït per a Linux. També funciona, però, des de Windows com veurem una miqueta més avant.

Instal·lació en Linux

El lloc des d'on baixar-lo és la pàgina oficial:

<http://redis.io>

En el moment de fer aquestos apunts, l'última versió estable és la **3.0.7**.

Ens baixem el fitxer, el descomprimim, i després des d'una terminal ens situem en el directori on s'ha descomprimit i fem **make** per a generar els executables. Després de molts avisos, s'hauria d'haver instal·lat bé, i sense ser necessaris els permisos d'administrador. Aquest seria el resum d'accions, fetes totes elles des d'una terminal (però no cal descomprimir des d'una terminal) i havent-nos situat prèviament en el lloc on està el fitxer baixat. També suposarem que el fitxer està col·locat en el lloc on volem que estiga instal·lat de forma definitiva. Recordeu que podeu descomprimir-lo de la manera que us resulte més còmoda:

```
tar xzf redis-3.0.7.tar.gz
```

o també el podeu descomprimir des del navegador d'arxius, com ja havíem comentat.

Una vegada descomprimit, des d'una terminal ens hem de situar en el directori acabat de descomprimir i fer **make**

```
cd redis-3.0.7  
make
```

Amb açò s'haurien d'haver generat els executables, i ja hauria de funcionar. Recordeu que no fan falta permisos d'administrador per a realitzar açò.

Per a posar en marxa el servidor, quasi que el més còmode serà obrir un terminal, situar-nos en el directori **redis-3.0.7/src** i des d'ahi executar **redis-server**. Hauria d'eixir una finestra similar a la següent, amb més o menys avisos (observeu que al principi de la imatge estan les ordres donades).

```
curs@Ubuntu: ~/redis-3.0.7/src
Fitxer Edita Visualitza Cerca Terminal Ajuda
curs@Ubuntu:~/redis-3.0.7$ cd src/
curs@Ubuntu:~/redis-3.0.7/src$ ./redis-server
7019:C 17 Feb 12:40:09.874 # Warning: no config file specified, using the default
t config. In order to specify a config file use ./redis-server /path/to/redis.co
nf
7019:M 17 Feb 12:40:09.874 # You requested maxclients of 10000 requiring at leas
t 10032 max file descriptors.
7019:M 17 Feb 12:40:09.874 # Redis can't set maximum open files to 10032 because
of OS error: Operation not permitted.
7019:M 17 Feb 12:40:09.874 # Current maximum open files is 4096. maxclients has
been reduced to 4064 to compensate for low ulimit. If you need higher maxclients
increase 'ulimit -n'.

                                Redis 3.0.7 (00000000/0) 64 bit

                                Running in standalone mode
                                Port: 6379
                                PID: 7019

                                http://redis.io

7019:M 17 Feb 12:40:09.876 # WARNING: The TCP backlog setting of 511 cannot be e
nforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
7019:M 17 Feb 12:40:09.876 # Server started, Redis version 3.0.7
7019:M 17 Feb 12:40:09.876 # WARNING overcommit_memory is set to 0! Background s
ave may fail under low memory condition. To fix this issue add 'vm.overcommit_me
memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.ove
rcommit_memory=1' for this to take effect.
7019:M 17 Feb 12:40:09.876 # WARNING you have Transparent Huge Pages (THP) suppo
rt enabled in your kernel. This will create latency and memory usage issues with
Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transpare
nt_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retai
n the setting after a reboot. Redis must be restarted after THP is disabled.
7019:M 17 Feb 12:40:09.876 * The server is now ready to accept connections on po
rt 6379
```

Entre altres coses diu que el servidor està en marxa esperant connexions al port 6379, que és el port per defecte de Redis. Aquesta finestra del terminal l'haurem de deixar en marxa. Quan vulguem detenir Redis, senzillament fem **ctrl-c**, i detindrem l'execució de forma ordenada (guardant-se les dades no guardades i tancant-se tot bé)

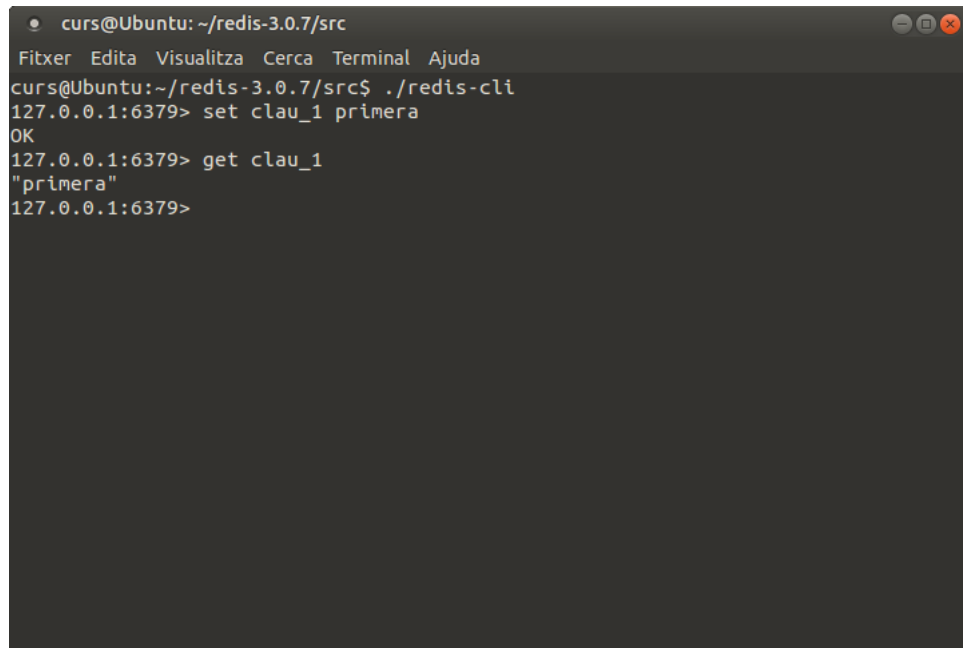
Podríem haver executat directament **redis-server** fent-li doble-clic des d'un explorador d'arxius, per exemple, però aleshores no podríem parar-lo i en definitiva controlar-lo tan còmodament.

Per a fer una connexió des d'un client, també des d'un terminal (un altre) executem **redis-cli**:

```
curs@Ubuntu: ~/redis-3.0.7/src
Fitxer Edita Visualitza Cerca Terminal Ajuda
curs@Ubuntu:~/redis-3.0.7/src$ ./redis-cli
127.0.0.1:6379>
```

Ja ha fet la connexió, concretament a localhost (127.0.0.1) i al port 6379, que havíem quedat que és el port per defecte.

Comprovem que sí que funciona. Encara no hi ha dades, perquè l'acabem d'instal·lar. I recordeu que és una Base de Dades clau-valor. Per a crear una entrada posarem **set clau valor**. Per a obtenir-la posarem **get clau**. En la imatge es pot comprovar:

A screenshot of a terminal window titled 'curs@Ubuntu: ~/redis-3.0.7/src'. The window has a menu bar with 'Fitxer', 'Edita', 'Visualitza', 'Cerca', 'Terminal', and 'Ajuda'. The terminal shows the following commands and output:

```
curs@Ubuntu:~/redis-3.0.7/src$ ./redis-cli
127.0.0.1:6379> set clau_1 primera
OK
127.0.0.1:6379> get clau_1
"primera"
127.0.0.1:6379>
```

Hem creat una clau anomenada **clau_1** amb el valor **primera**, com es pot comprovar en el moment d'obtenir-la amb **get**.

Si al programa **redis-cli** no li posem paràmetres, intentarà fer una connexió local (localhost). Si volem connectar a un servidor situat en una altra adreça, li la posem amb el paràmetre **-h adreça**, per exemple:

```
redis-cli -h 99.66.214.106
```

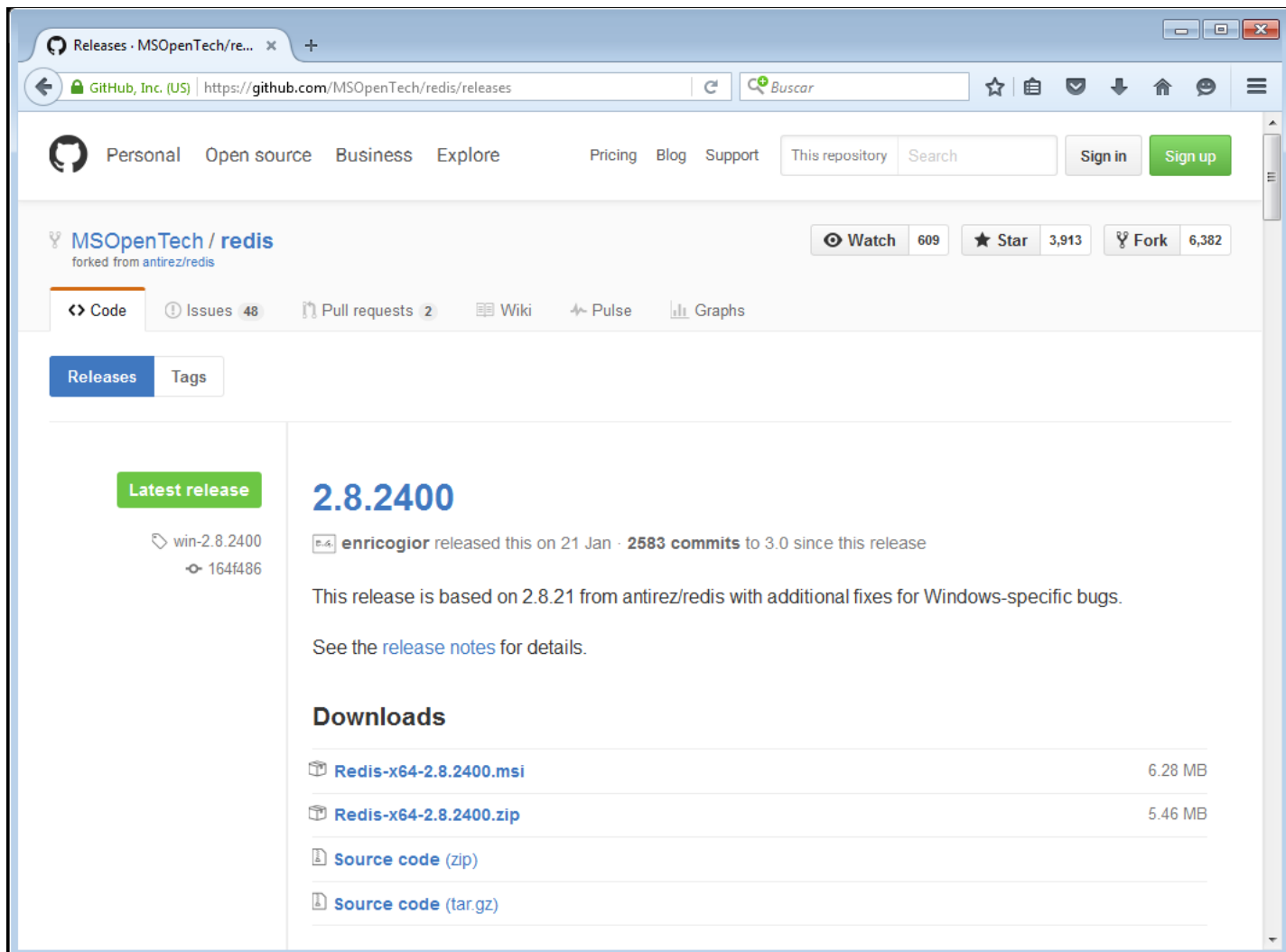
ens faria la connexió al servidor.

Instal·lació en Windows de 64 bits

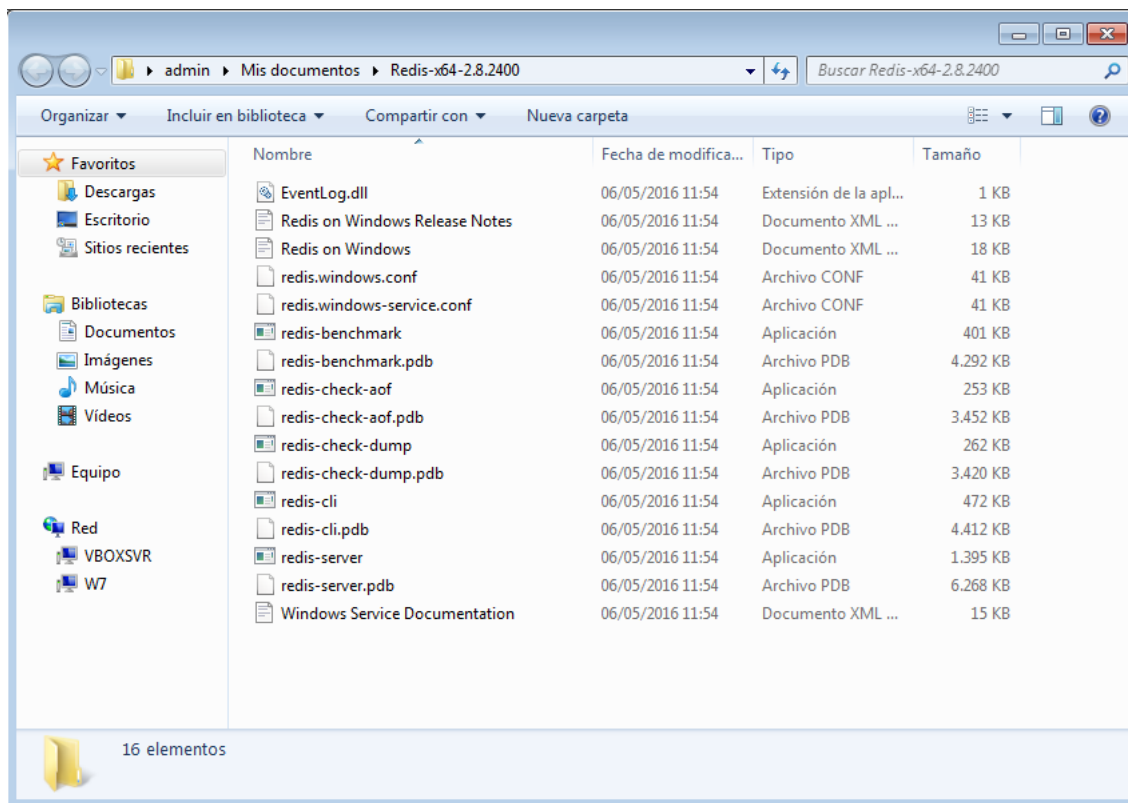
Encara que Redis està construït per a Linux, hi ha versions per a Windows, preferiblement de 64 bits. També podrem trobar versions de 32 bits, però molt més antigues.

El lloc on poder baixar els fitxers de Redis per a Windows de 64 bits és:

<https://github.com/MSOpenTech/redis/releases>



Ens baixem el **zip**, el descomprimim, i ja ho tindrem disponible (sense fer **make** ni res). Observeu com en la carpeta resultat de descomprimir ja tenim els executables **redis-server** i **redis-cli** que són els que ens interessen:



Executem **redis-server** directament i ja el tindrem en marxa:

```
C:\Users\usuari\Redis 2.8.24\redis-server.exe

[848] 06 May 17:38:57.864 # Warning: no config file specified, using the default
config. In order to specify a config file use C:\Users\usuari\Redis 2.8.24\red
is-server.exe /path/to/redis.conf

Redis 2.8.2400 (00000000/0) 64 bit
Running in stand alone mode
Port: 6379
PID: 848

http://redis.io

[848] 06 May 17:38:57.883 # Server started, Redis version 2.8.2400
[848] 06 May 17:38:57.883 * The server is now ready to accept connections on por
t 6379
```

Executem també el **redis-cli** i el resultat serà el mateix que en Linux. Hem incorporar una nova clau i després obtenim el seu contingut:

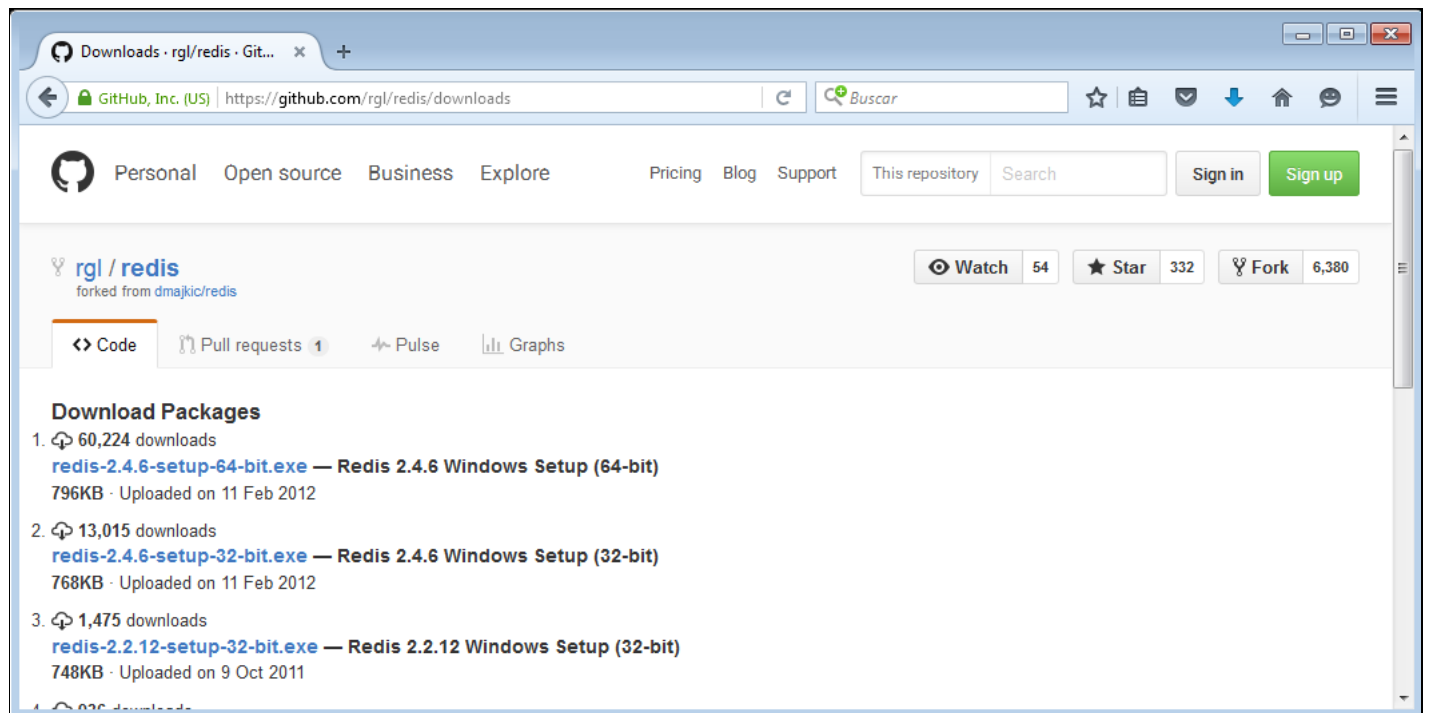
```
C:\Users\usuari\Redis 2.8.24\redis-cli.exe

127.0.0.1:6379> set clau_1 primera
OK
127.0.0.1:6379> get clau_1
"primera"
127.0.0.1:6379> _
```

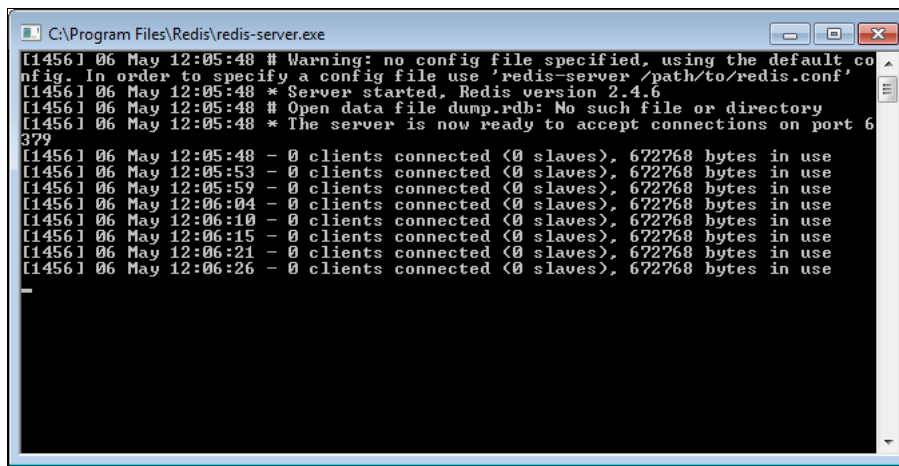
Instal·lació en Windows de 32 bits

És un poc més complicada de trobar. I sobretot, és una versió prou més antiga. Podem descarregar-la de la següent adreça:

<https://github.com/rgl/redis/downloads>

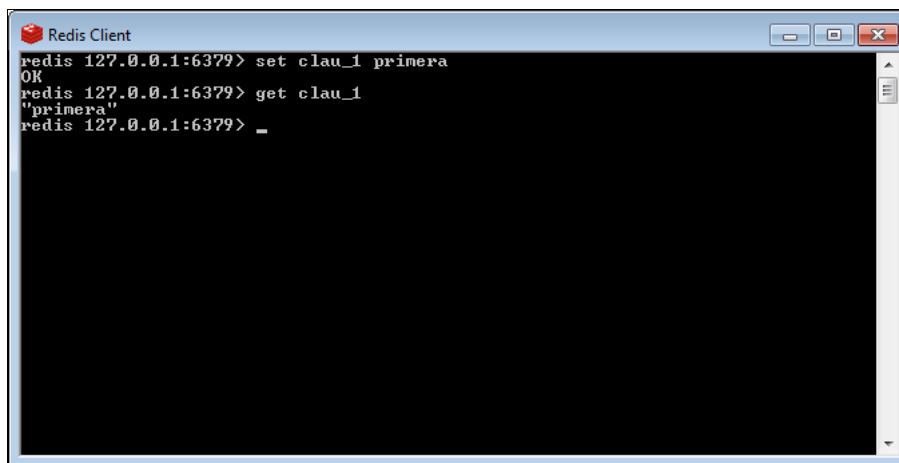


Aquesta sí que és d'instal·lació (no de descompressió únicament). La documentació suggereix que s'active com un servei. Per això, en la llista de programes de l'Inici no està el servidor, únicament el **client**. Però el podem trobar en el directori on s'ha instal·lat: **C:\Archivos de programa\Redis**. Trobarem tant **redis-server.exe** com **redis-cli.exe**. Posem en marxa el servidor fent doble-clic a **redis-server.exe**:



```
C:\Program Files\Redis\redis-server.exe
[1456] 06 May 12:05:48 # Warning: no config file specified, using the default configuration. In order to specify a config file use 'redis-server /path/to/redis.conf'
[1456] 06 May 12:05:48 * Server started, Redis version 2.4.6
[1456] 06 May 12:05:48 # Open data file dump.rdb: No such file or directory
[1456] 06 May 12:05:48 * The server is now ready to accept connections on port 6379
[1456] 06 May 12:05:48 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:05:53 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:05:59 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:06:04 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:06:10 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:06:15 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:06:21 - 0 clients connected (0 slaves), 672768 bytes in use
[1456] 06 May 12:06:26 - 0 clients connected (0 slaves), 672768 bytes in use
```

I ja només falta posar en marxa el client (des del menú o fent doble-clic a **redis-cli.exe**). En la imatge es veu com podem connectar, crear una clau i tornar el seu valor:



```
Redis Client
redis 127.0.0.1:6379> set clau_1 primera
OK
redis 127.0.0.1:6379> get clau_1
"primera"
redis 127.0.0.1:6379> _
```

2.2 - Entron gràfic: Redis Desktop Manager

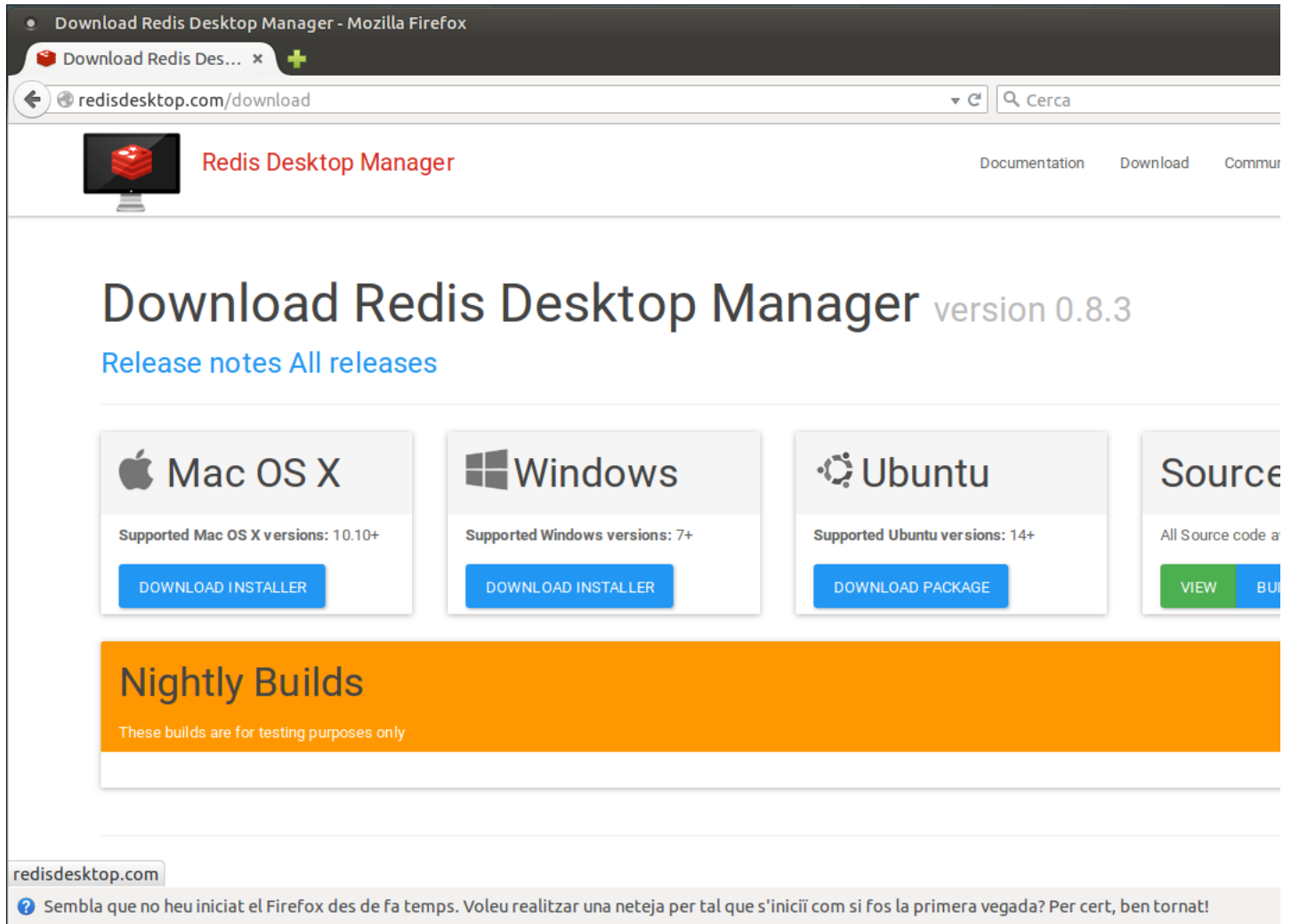
Com hem comprovat en el punt anterior, la connexió que fem des del client és a través de consola. Per tant haurem de posar comandos i ens contestarà la seua execució.

Podem instal·lar-nos una aplicació gràfica que faça un poc més atractiva la presentació.

La instal·lació d'aquesta eina és **totalment optativa**, no cal que la feu. De fet, ens els exemples que es mostraran en tot el tema només s'utilitzarà el mode consola.

És completament independent del servidor, i podem instal·lar-la perfectament sense tenir el servidor, utilitzant-la aleshores per a connectar a un servidor remot.

El podem baixar lliurement de la pàgina oficial redisdesktop.com on podrem comprovar que tenim per a totes les plataformes:



Instal·lació en Ubuntu

Ens baixarem el paquet que haurem d'instal·lar bé amb el Centre de programari d'Ubuntu, o bé senzillament des d'una terminal, situats al directori on l'hem baixat fent

```
sudo dpkg -i redis-desktop-manager_0.8.3-120_amd64.deb
```

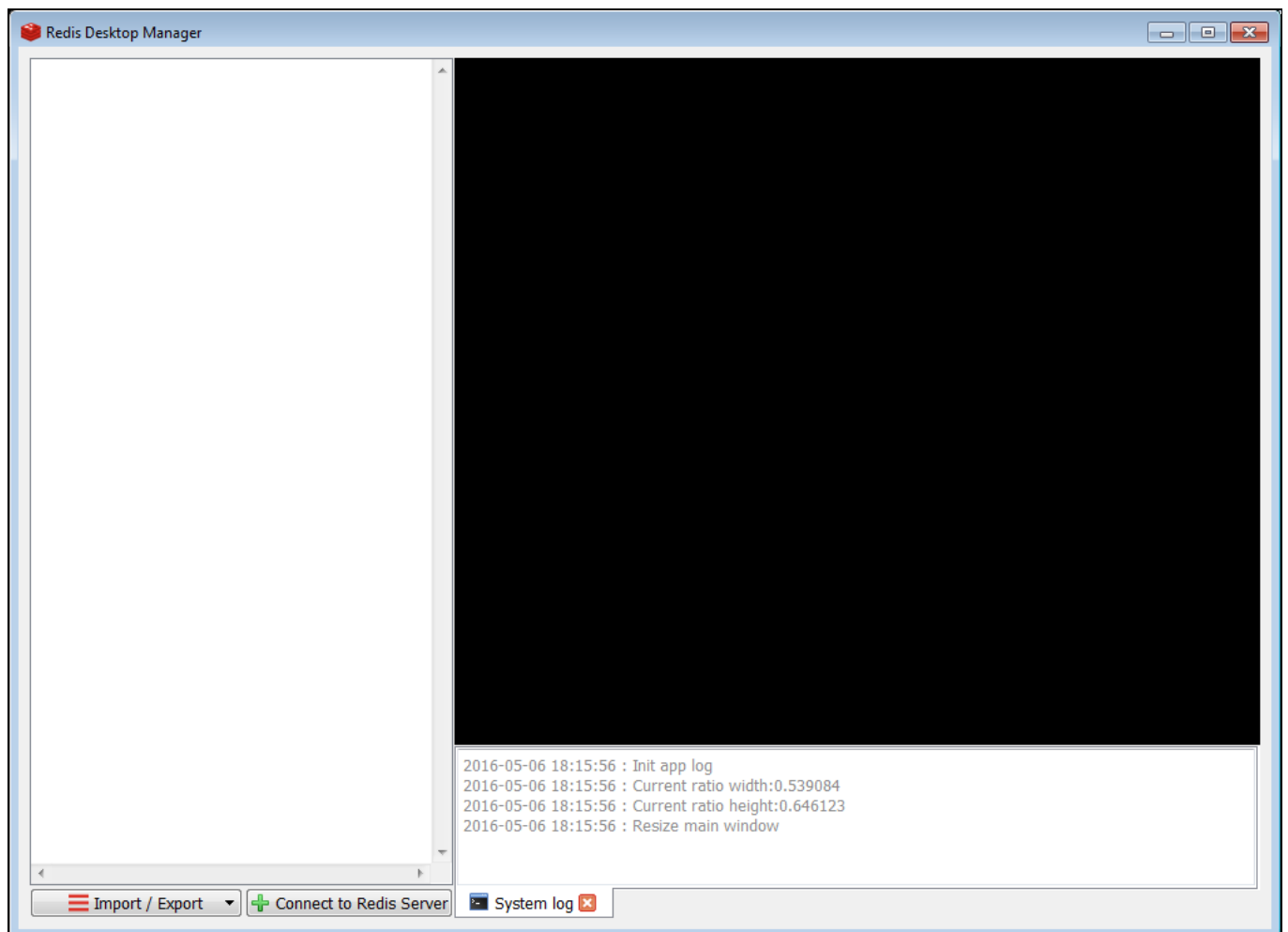
Com veieu fan falta permisos d'administrador, i si no hi ha problemes de dependències, s'instal·larà sense problemes.

Com que és igual que en Windows, si teniu curiositat mireu la imatge posterior de quan està instal·lat, per a comprovar la connexió al servidor.

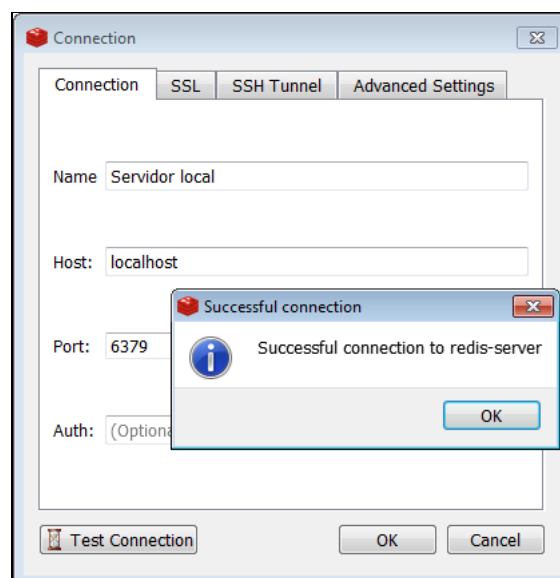
Instal·lació en Windows

En Windows el que ens baixarem és un exe. L'executem (permetent l'execució quan ho pregunta Windows) i li podem donar a totes les opcions per defecte.

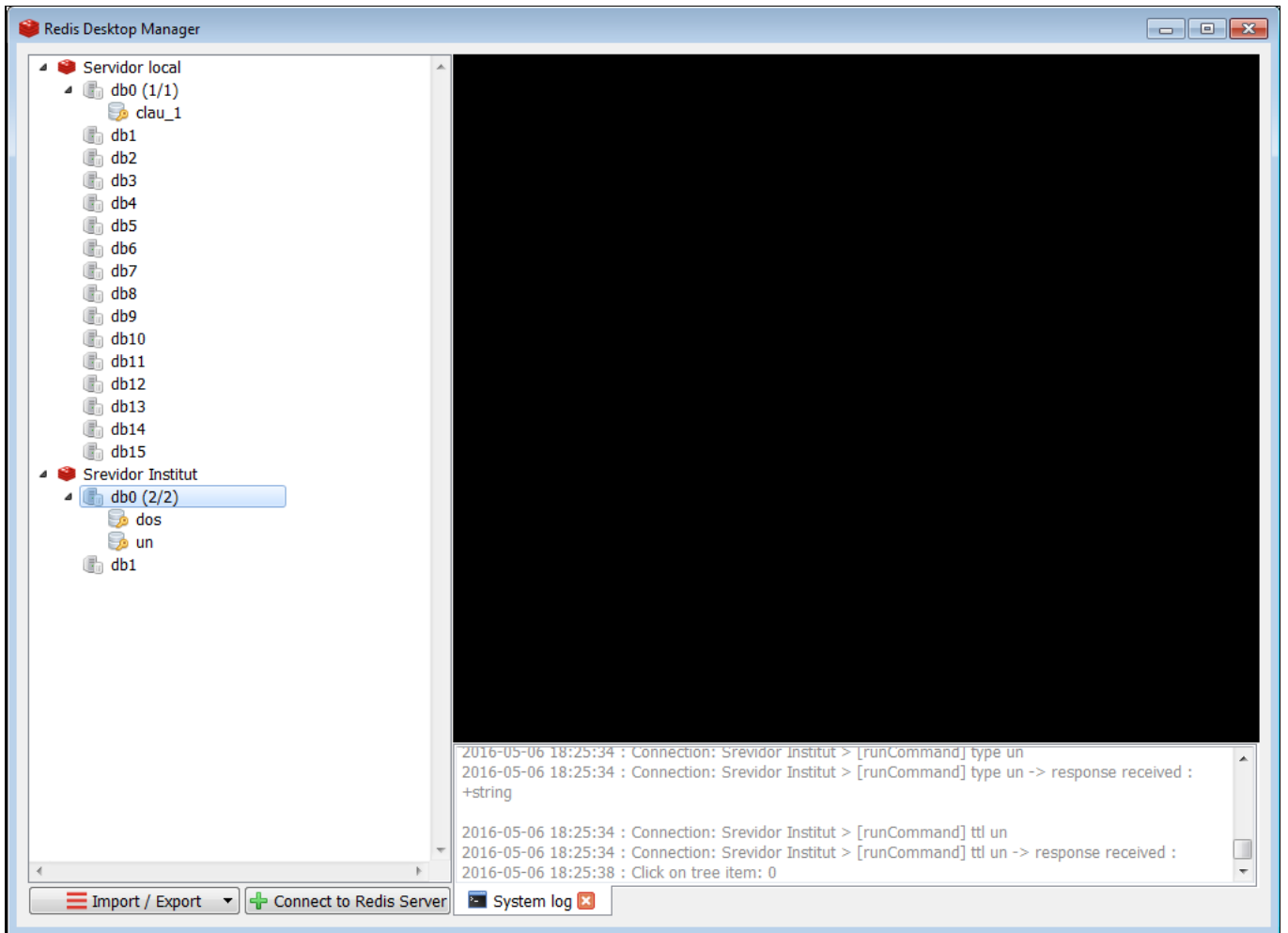
Quan l'executem, ens eixirà la següent pantalla:



Podem comprovar que baix tenim el botó per a connectar a un servidor. Per a connectar al servidor local li posem com a adreça **localhost**. Per a connectar al servidor remot li posem com a adreça **la IP del servidor**. En la imatge s'ha fet el test de connexió.



En aquesta imatge es veu com hem connectat perfectament als dos servidors.



Anem a veure la utilització de Redis. Ens connectarem com a clients i intentarem fer operacions.

- Les primeres seran les més senzilles, utilitzant únicament el tipus de dades **String**.
- Posteriorment mirarem com treballar amb les claus: buscar-ne una, veure si existeix, buscar unes quantes, ...
- Després ja anirem a pels tipus de dades més complicats:
 - **Hash**
 - **List**
 - **Set**
 - **Sorted Set**

És el tipus de dades més senzill, més bàsic. Serà una cadena de caràcters de tipus **binary safe** en la qual normalment guardarem les habituals cadenes de caràcters, però que també podríem guardar imatges o objectes serialitzats de Java. La grandària màxima és de 512Mb.

Ara veurem els comandos més habituals que afecten a aquest tipus. Com a norma general, hem de ser conscients que els comandos no són sensibles a majúscules o minúscules, però les claus i els valors sí que ho són. És a dir, el comando **get** també es pot escriure **GET** o **Get**. Però la clau **Hola** és diferent de la clau **hola**.

Concretament, els comandos que veurem seran:

get , **set** , **setex** , **psetex** , **mget** , **mset** , **append** , **strlen** , **getrange** , **setrange** , **incr** , **decr** , **incrby** , **decrby**

GET

Sintaxi

```
get clau
```

Torna el valor de la clau especificada, sempre que siga de tipus **String**. Si la clau és d'un altre tipus, donarà error. I si la clau no existeix, tornarà el valor especial **nil**.

Exemples

```
127.0.0.1:6379> get clau_1
"primera"
127.0.0.1:6379> get clau_2
(nil)
```

SET

Sintaxi

```
set clau valor
```

Assigna a la clau especificada com a primer paràmetre el valor especificat com a segon paràmetre. Si el valor consta de més d'una paraula, haurà d'anar entre cometes dobles.

Redis sempre guardarà el valor com a string, encara que nosaltres pensem que li passem un valor enter o real.

I una altra característica és que si la clau existeix ja, matxacarà el seu contingut, com era de esperar.

Exemples

```
127.0.0.1:6379> set clau_2 segona
OK
127.0.0.1:6379> set text "Un text amb més d'una paraula"
OK
```

```
127.0.0.1:6379> set quatre 4
OK
127.0.0.1:6379> get quatre
"4"
127.0.0.1:6379> set pi 3.14159265359
OK
127.0.0.1:6379> get pi
"3.14159265359"
```

Nota

Si poseu algun accent, en tornar el valor (fent get) us semblarà que no s'ha guardat bé. Sí que s'haurà guardat bé, el que passa és que posteriorment no es visualitza bé en fer el get. Es pot comprovar entrant en el client amb l'opció **raw**, és a dir **redis-cli --raw**

El comando **SET** té una opció molt interessant, que servirà per a donar un temps de vida a la clau, transcorregut el qual desapareix la clau (amb el seu valor, clar). Açò s'anomena **temps d'expiració** i s'aconsegueix amb el paràmetre **EX** del comando **SET** seguit del número de segons que volem que dure la clau.

Exemple

```
127.0.0.1:6379> set clau_3 tercera ex 10
OK
127.0.0.1:6379> get clau_3
"tercera"
127.0.0.1:6379> get clau_3
(nil)
```

Primer sí que existeix, però al cap de 10 segons ha deixat d'existir.

De forma equivalent es pot expressar el temps en milisegons, amb el paràmetre **PX** en compte de **EX**.

Havíem comentat al principi, que si en el moment de fer el **SET** la clau ja existia, es reemplaçarà el seu contingut. Podem modificar aquest comportament amb el paràmetre **NX** (Not eXists): si no existia la clau, la crearà amb el valor, però si ja existia, la deixarà com estava. Ens ho indicarà dient OK en cas de crear-la i NIL en cas de no crear-la perquè ja existia.

```
127.0.0.1:6379> set clau_4 quarta nx
OK
127.0.0.1:6379> set clau_1 quarta nx
(nil)
127.0.0.1:6379> get clau_4
"quarta"
127.0.0.1:6379> get clau_1
"nil"
```

I de forma inversa, si posem el paràmetre **XX**, si ja existeix la clau, reemplaçarà el valor, però si no existia, no farà res.

SETEX

Sintaxi

```
setex clau segons valor
```

Funciona igual que el **SET** amb el paràmetre **EX**: crearà la clau amb el valor, però tindrà una existència dels segons indicats.

PSETEX

Sintaxi

```
psetex clau milisegons valor
```

Funciona igual que l'anterior, però el que especifiquem són els milisegons d'existència.

MGET

Sintaxi

```
mget clau1 clau2 clauN
```

Torna una llista de valors, els de les claus indicades.

Exemple

```
127.0.0.1:6379> set mes1 gener
OK
127.0.0.1:6379> set mes2 febrer
OK
127.0.0.1:6379> set mes3 març
OK
127.0.0.1:6379> mget mes1 mes2 mes3
1) "gener"
2) "febrer"
3) "març"
```

Nota

Recordeu que els caràcters com vocals accentuades, ç, ñ, ... s'han introduït bé, però potser no es visualitzen bé. Es pot evitar entrant en el client d'aquesta manera **redis-cli --raw**

Si alguna de les claus no existeix, tornarà **nil** en el seu lloc

MSET

Sintaxi

```
mset clau1 valor1 clau2 valor2 clauN valorN
```

Assigna els valors corresponents a les claus. És una operació atòmica: es posen (o canvien) tots els valors a l'hora.

```
127.0.0.1:6379> mset mes4 abril mes5 maig mes6 juny mes7 juliol
OK
127.0.0.1:6379> mget mes1 mes2 mes3 mes4 mes5 mes6 mes7
1) "gener"
2) "febrer"
3) "març"
4) "abril"
5) "maig"
6) "juny"
7) "juliol"
```

```
4) "abril"  
5) "maig"  
6) "juny"  
7) "juliol"
```

Si no vulguérem reemplaçar valors, podríem utilitzar el comando **MSETNX**, totalment equivalent, però hauríem de tenir en compte que si alguna ja existeix i per tant no pot canviar el valor, no faria l'operació, és a dir, tampoc crearia les altres.

APPEND

Sintaxi

```
append clau1 valor1
```

Si la clau no existeix la crea assignant-li el valor (com el **SET**), però si ja existeix, concatena el valor al final de la cadena que ja hi havia.

```
127.0.0.1:6379> append salutacio Hola  
(integer) 4  
127.0.0.1:6379> get salutacio  
"Hola"  
127.0.0.1:6379> append salutacio ", com va?"  
(integer) 13  
127.0.0.1:6379> get salutacio  
"Hola, com va?"
```

STRLEN

Sintaxi

```
strlen clau1
```

Torna el número de caràcters que hi ha en el valor de la clau. Si la clau no existeix, tornarà 0. Si la clau és d'un altre tipus, tornarà error.

```
127.0.0.1:6379> strlen salutacio  
(integer) 13  
127.0.0.1:6379> strlen sal  
(integer) 0
```

GETRANGE

Sintaxi

```
getrange clau1 inici final
```

Extrau una subcadena del valor de la clau (ha de ser de tipus **String**) des del número de caràcter d'inici fins al número del final (ambdós inclosos). El primer caràcter és el 0. Si posem de final un número major que l'últim, igual ho traurà fins el final.

Es poden posar també valor negatius que ens ajuden a agafar la cadena des del final. El -1 és l'últim caràcter, el -2 el penúltim, ... I es poden barrejar números positius i negatius. Així, **rang 0 -1** és tota la cadena.

```
127.0.0.1:6379> getrange salutacio 1 3  
"ola"  
127.0.0.1:6379> getrange salutacio 6 50  
"com va?"  
127.0.0.1:6379> getrange salutacio 6 -1  
"com va?"  
127.0.0.1:6379> getrange salutacio -7 -5  
"com"  
127.0.0.1:6379> getrange salutacio 0 -1  
"Hola, com va?"
```

SETRANGE

Sintaxi

```
setrange clau1 desplaçament valor
```

Substitueix part del valor de la cadena, a partir del desplaçament, amb el vaolr proporcionat. No s'admenten en desplaçament valors negatius.

```
127.0.0.1:6379> get salutacio  
"Hola, com va?"  
127.0.0.1:6379> setrange salutacio 4 ". C"  
(integer) 13  
127.0.0.1:6379> get salutacio  
"Hola. Com va?"
```

INCR

Sintaxi

```
incr clau1
```

A pesar de que Redis guarda els strings com a tals, com a cadenes de caràcters, en algunes ocasions és capaç de transformar la cadena a un número. És el cas del comando **INCR**, que converteix la cadena en un enter (si pot) i incrementa aquest valor en una unitat.

Si la clau no existeix la crea assumint que valia 0, i per tant després valdrà 1.

Si el valor de la clau no era un número enter, donarà un error.

```
127.0.0.1:6379> set compt1 20
OK
127.0.0.1:6379> get compt1
"20"
127.0.0.1:6379> incr compt1
(integer) 21
127.0.0.1:6379> get compt1
"21"
127.0.0.1:6379> incr compt2
(integer) 1
127.0.0.1:6379> get compt2
"1"
127.0.0.1:6379> incr clau_1
(error) ERR value is not an integer or out of range
127.0.0.1:6379> set compt3 4.25
OK
127.0.0.1:6379> incr compt3
(error) ERR value is not an integer or out of range
```

DECR

Sintaxi

```
decr clau1
```

Decrementa en una unitat el valor de la clau (senpre que siga un enter).

Pot agafar valors negatius.

Si la clau no existeix la crea assumint que valia 0, i per tant després valdrà -1.

```
127.0.0.1:6379> decr compt2
(integer) 0
127.0.0.1:6379> decr compt2
(integer) -1
127.0.0.1:6379> get compt2
"-1"
```

INCRBY

Sintaxi

```
incrby clau1 increment
```

Incrementa el valor de la clau en el número d'unitats indicat en **increment** (el valor ha de ser enter). L'increment pot ser negatiu.

```
127.0.0.1:6379> incrby compt1 10
(integer) 31
127.0.0.1:6379> incrby compt1 -20
(integer) 11
```

DECRBY

Sintaxi

```
decrby clau1 decrement
```

Decrementa el valor de la clau el número d'unitat indicat en **decrement**.

```
127.0.0.1:6379> decrby compt1 5
(integer) 6
```

Ara anem a veure comandos que ens permeten treballar amb les claus, per a buscar-les, veure si existeixen, etc. No importarà el tipus de les claus (de moment només hem treballat amb claus de tipus **String**, però si ja en tinguérem dels altres tipus també es veurien afectades). En cap cas d'aquests comandos accedirem al valor de les claus.

La llista de comandos de claus que veurem és:

`keys` , `exists` , `del` , `type` , `rename` , `renamenx` , `expire` , `pexpire` , `ttl` , `pttl` , `persist`

KEYS

Sintaxi

```
keys patró
```

Torna totes les claus que coincideixen amb el patró. En el patró podem posar caràcters comodí:

- `*` : equival a 0 o més caràcters. Per exemple "Mar*a" podria tornar "Mara", "Maria", "Marta", "Margarita", ...
- `?` : equival exactament a un caràcter. Per exemple "Mar?a" podria tornar "Maria" o "Marta", però no "Mara", "Margarita", ...
- `[ab]` : serà cert si en el lloc corresponent hi ha un dels caràcters especificats entre els claudàtors. Per exemple "Mar[itja]" podria tornar "Maria" o "Marta", però no "Marga"

Per a tornar totes les claus utilitzarem `keys *`

```
127.0.0.1:6379> keys *
1) "mes3"
2) "salutacio"
3) "pi"
4) "clau_4"
5) "compt2"
6) "mes6"
7) "mes1"
8) "mes4"
9) "clau_1"
10) "compt3"
11) "quatre"
12) "mes7"
13) "mes2"
14) "mes5"
15) "compt1"
16) "clau_2"
17) "text"
```

```
127.0.0.1:6379> keys mes?
1) "mes5"
2) "mes3"
3) "mes7"
4) "mes2"
5) "mes6"
6) "mes4"
7) "mes1"
```

```
127.0.0.1:6379> keys c*
1) "clau_4"
2) "compt2"
3) "clau_1"
4) "compt3"
5) "compt1"
6) "clau_2"
```

```
127.0.0.1:6379> keys mes[125]
1) "mes5"
2) "mes2"
3) "mes1"
```

Nota

L'ordre en el qual es mostraran les claus no té per què ser igual per a tots nosaltres. El més normal és que cadascú de nosaltres tinga un ordre diferent. Açò té a veure amb els conjunts (**set**), un tipus de dades que veurem més avant.

EXISTS

Sintaxi

```
exists clau
```

Torna 1 si la clau existeix, i 0 si no existeix. No importa de quin tipus siga la clau.

```
127.0.0.1:6379> exists clau_1
(integer) 1
127.0.0.1:6379> exists clau_25
(integer) 0
```

DEL

Sintaxi

```
del clau1 clau2 clauN
```

Elimina la clau o claus especificades. Si posem més d'una clau i alguna no existeix, la ignorarà i sí que esborrarà les altres.

```
127.0.0.1:6379> del compt2
(integer) 1
127.0.0.1:6379> del mes6 mes7 mes8 mes9
(integer) 2
```

Observeu que ens indica quantes claus ha esborrat. En el primer exemple ha esborrat la clau especificada, i en el segon diu que ha esborrat 2, que seran **mes6** i **mes7**, ja que **mes8** i **mes9** no existien.

TYPE

Sintaxi

```
type clau
```

Torna el tipus de la clau especificada. Els valors possibles són:

- string
- hash
- list
- set
- zset (conjunt ordenat)

Exemples

```
127.0.0.1:6379> type clau_1
string
```

RENAME

Sintaxi

```
rename clau novaclau
```

Canvia el nom de la clau a la clau nova, conservant el valor. Dóna error si la clau antiga no existeix. Si la clau nova ja existia reemplaçarà el seu valor.

Exemples

```
127.0.0.1:6379> get salutacio
"Hola. Com va?"
127.0.0.1:6379> rename salutacio saludar
OK
127.0.0.1:6379> get salutacio
(nil)
127.0.0.1:6379> get saludar
"Hola. Com va?"
127.0.0.1:6379> rename clau_22 clau_23
(error) ERR no such key
```

RENAMENX

Sintaxi

```
renamenx clau novaclau
```

Igual que l'anterior però únicament si la clau nova no existia. Si ja existia no fa res (tornant 0 per a indicar-ho).

Exemples

```
127.0.0.1:6379> renamenx compt1 compt3
(integer) 0
127.0.0.1:6379> get compt1
"9"
```

Estem suposant que la clau **compt3** ja existeix

EXPIRE

Sintaxi

```
expire clau segons
```

Assigna com a temps d'expiració de la clau els segons especificats. Si ja tenia temps d'expiració, el modifica posant-li aquest valor especificat.

En cas que a una clau amb temps d'expiració li canviem el nom amb **RENAME**, continuarà amb temps d'expiració que li quedava.

PEXPIRE

Sintaxi

```
pexpire clau milisegons
```

El mateix però en milisegons

TTL

Sintaxi

```
ttl clau
```

Torna el temps de vida (fins l'expiració) d'una clau. Si la clau no té temps d'expiració, torna -1.

Exemples

```
127.0.0.1:6379> expire compt3 10
(integer) 1
127.0.0.1:6379> ttl compt3
(integer) 6
127.0.0.1:6379> ttl compt3
(integer) 3
127.0.0.1:6379> ttl compt3
(integer) 0
127.0.0.1:6379> get compt3
(nil)
```

PTTL

Sintaxi

```
pttl clau
```

Igual que l'anterior, però ens torna el temps en milisegons.

PERSIST

Sintaxi

```
persist clau
```

Elimina el temps d'expiració d'una clau, si és que en tenia. Ara la clau no expirarà mai.

Exemples

```
127.0.0.1:6379> expire compt1 20
(integer) 1
127.0.0.1:6379> ttl compt1
(integer) 12
127.0.0.1:6379> ttl compt1
(integer) 7
127.0.0.1:6379> persist compt1
(integer) 1
127.0.0.1:6379> ttl compt1
(integer) -1
127.0.0.1:6379> get compt1
"9"
```

Ja havíem comentat que el tipus **Hash** és una espècie de registre, amb subcamps (en realitat hauríem de dir sub-claus). Pot tenir qualsevol nombre de subcamps que seran de tipus String.

Redis és molt eficient en quant a l'espai que ocupen els **Hash**, i sobretot en el temps de recuperació de les dades.

Els comandos que vam veure per al **String** no es poden aplicar al **Hash**. Tanmateix els comandos del **Hash** són molt similars a aquells, començant sempre per **H**.

Veurem els següents comandos:

hset , **hget** , **hgetall** , **hdel** , **hkeys** , **hvals** , **hmget** , **hmset** , **hexists** , **hsetnx** , **hincrby**

HSET

Sintaxi

```
hset clau camp valor
```

Assigna al camp especificat de la clau especificada el valor especificat. Si el valor consta de més d'una paraula, haurà d'anar entre cometes dobles.

Si la clau no existia, la crearà, i si ja existia, senzillament afegirà el camp. I si d'aquesta clau ja existia el camp, modificarà el seu valor.

Evidentment, en claus diferents poden haver camps amb els mateixos noms.

Exemples

```
127.0.0.1:6379> hset empleat_1 nom Andreu
(integer) 1
127.0.0.1:6379> hset empleat_1 departament 10
(integer) 1
127.0.0.1:6379> hset empleat_1 sou 1000.0
(integer) 1
```

```
127.0.0.1:6379> hset empleat_2 nom Berta
(integer) 1
127.0.0.1:6379> hset empleat_2 sou 1500.0
(integer) 1
```

HGET

Sintaxi

```
hget clau camp
```

Torna el valor del camp de la clau. Si no existia (el camp o la clau) torna **nil**. Només podem especificar un camp.

Exemples

```
127.0.0.1:6379> hget empleat_1 nom
"Andreu"
127.0.0.1:6379> hget empleat_1 departament
"10"
127.0.0.1:6379> hget empleat_2 nom
"Berta"
127.0.0.1:6379> hget empleat_2 departament
(nil)
```

HGETALL

Sintaxi

```
hgetall clau
```

Torna una llista amb tots els camps i els seus valors de la clau. La seqüència és: camp1 valor1 camp2 valor2 ... Però no ens podem fiar que l'ordre siga el mateix ordre que quan el vam definir.

Exemples

```
127.0.0.1:6379> hgetall empleat_1
1) "nom"
2) "Andreu"
```

```
3) "departament"  
4) "10"  
5) "sou"  
6) "1000.0"
```

HDEL

Sintaxi

```
hdel clau camp1 camp2 campN
```

Elimina el o els camps especificats. Si no existeixen algun d'ells, senzillament l'ignora i si que elimina els altres.

Exemples

```
127.0.0.1:6379> hdel empleat_1 departament  
(integer) 1  
127.0.0.1:6379> hgetall empleat_1  
1) "nom"  
2) "Andreu"  
3) "sou"  
4) "1000.0"
```

HKEYS

Sintaxi

```
hkeys clau
```

Torna una llista amb els camps de la clau. Si la clau no existia, torna una llista buida

Exemples

```
127.0.0.1:6379> hkeys empleat_1  
1) "nom"  
2) "sou"
```

HVALS

Sintaxi

```
hvals clau
```

Torna una llista amb els valors (únicament els valors) de tots els camps de la clau. Si la clau no existia, torna una llista buida

Exemples

```
127.0.0.1:6379> hvals empleat_1  
1) "Andreu"  
2) "1000.0"
```

Altres Comandos

També existeixen altres comandos, de funcionament com cabria esperar (els hem vist tots en el cas de **String**):

- **hmget**: Torna més d'un camp de la clau
- **hmset**: assigna més d'un camp a una clau
- **hexists**: indica si existeix el subcamp de la clau
- **hsetnx**: assigna únicament en cas de que no existisca el camp.
- **hincrby**: incrementa el camp de la clau

Les **Llistes** en **Redis** són llistes de Strings ordenades, on cada element està associat a un índex de la llista. Es poden recuperar els elements tant de forma ordenada (per l'índex) com accedint directament a una posició.

Els elements es poden afegir al principi, al final o també en una posició determinada.

La llista es crea en el moment en què s'insereix el primer element, i desapareix quan llevem l'últim element que quede.

Estan molt ben optimitzades per a la inserció i per a la consulta.

Els comandos que afecten a les llistes comencen quasi tots per **L**, excepte alguns que comencen per **R** indicant que fan l'operació per la dreta.

Per cert, els valors dels elements es poden repetir.

La llista de comandos que afecten a llistes que veurem és:

lpush , **rpush** , **lpop** , **rpop** , **lset** , **lindex** , **linsert** , **lrange** , **llen** , **lrem** , **ltrim**

LPUSH

Sintaxi

```
lpush clau valor1 valor2 valorN
```

Introdueix els valors a la llista (creant la clau si és necessari). Les insereix en la primera posició, o també podríem dir que per l'esquerra (**Left PUSH**), imaginant que els elements estan ordenats d'esquerra a dreta. Si posem més d'un valor, s'aniran introduint sempre en la primera posició. El comando tornarà el número d'elements (strings) de la llista després de la inserció.

Exemples

```
127.0.0.1:6379> lpush llista1 primera segona tercera
(integer) 3
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "tercera"
2) "segona"
3) "primera"
```

```
127.0.0.1:6379> lpush llista1 quarta cinquena
(integer) 5
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "cinquena"
2) "quarta"
3) "tercera"
4) "segona"
5) "primera"
```

Nota

Per a veure el contingut de la llista utilitzarem el comando **lrange llista 0 -1**, que torna la llista sencera. Veurem de forma més completa aquest comando amb posterioritat.

RPUSH

Sintaxi

```
rpush clau valor1 valor2 valorN
```

Introdueix els valors a la llista (creant la clau si és necessari). Les insereix en l'última posició, o també podríem dir que per la dreta (**Right PUSH**), imaginant que els elements estan ordenats d'esquerra a dreta. El comando tornarà el número d'elements (strings) de la llista després de la inserció.

Exemples

```
127.0.0.1:6379> rpush llista1 sisena setena
(integer) 7
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "cinquena"
2) "quarta"
3) "tercera"
4) "segona"
5) "primera"
```

- 6) "sisena"
- 7) "setena"

LPOP

Sintaxi

```
lpop clau
```

Torna i elimina el primer element (el de més a l'esquerra).

Exemples

```
127.0.0.1:6379> lpop llista1  
"cinquena"
```

```
127.0.0.1:6379> lrange llista1 0 -1  
1) "quarta"  
2) "tercera"  
3) "segona"  
4) "primera"  
5) "sisena"  
6) "setena"
```

RPOP

Sintaxi

```
rpop clau
```

Torna i elimina l'últim element (el de més a la dreta).

Exemples

```
127.0.0.1:6379> rpop llista1  
"setena"
```

```
127.0.0.1:6379> lrange llista1 0 -1  
1) "quarta"  
2) "tercera"  
3) "segona"  
4) "primera"  
5) "sisena"
```

LSET

Sintaxi

```
lset clau index valor
```

Substitueix el valor de la posició indicada per l'índex. Tant la clau com l'element de la posició indicada han d'existir, sinó donarà error. Ara la **L** no significa **Left** sinó **List**.

La primera posició és la 0. I també es poden posar números negatius: -1 és l'últim, -2 el penúltim, ...

Exemples

```
127.0.0.1:6379> lset llista1 2 quarta  
OK
```

```
127.0.0.1:6379> lrange llista1 0 -1  
1) "quarta"  
2) "tercera"  
3) "quarta"  
4) "primera"  
5) "sisena"
```

```
127.0.0.1:6379> lset llista1 -1 cinquena  
OK
```

```
127.0.0.1:6379> lrange llista1 0 -1  
1) "quarta"  
2) "tercera"  
3) "quarta"  
4) "primera"  
5) "cinquena"
```

Observeu com es poden repetir els valors

LINDEX

Sintaxi

```
lindex clau index
```

Torna l'element situat en la posició indicada per l'índex, **però sense eliminar-lo de la llista**.

Exemples

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "tercera"
3) "quarta"
4) "primera"
5) "cinquena"
```

```
127.0.0.1:6379> lindex llista1 0
"quarta"
```

```
127.0.0.1:6379> lindex llista1 3
"primera"
```

```
127.0.0.1:6379> lindex llista1 -1
"cinquena"
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "tercera"
3) "quarta"
4) "primera"
5) "cinquena"
```

LINSERT

Sintaxi

```
linsert clau BEFORE | AFTER valor1 valor2
```

Insereix el valor2 abans o després (segons el que triem) de la primera vegada que troba el valor1. No substitueix, sinó que insereix en una determinada posició. Els elements que van després de l'element introduït veuran actualitzat el seu índex.

Exemples

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "tercera"
3) "quarta"
4) "primera"
5) "cinquena"
```

```
127.0.0.1:6379> linsert llista1 AFTER quarta segona
(integer) 6
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "segona"
3) "tercera"
4) "quarta"
5) "primera"
6) "cinquena"
```

```
127.0.0.1:6379> linsert llista1 BEFORE cinquena sisena
(integer) 7
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "segona"
3) "tercera"
4) "quarta"
5) "primera"
6) "sisena"
7) "cinquena"
```

Si intentem inserir abans o després un element que no existeix, tornarà -1 indicant que no l'ha trobat i no farà la inserció.

```
127.0.0.1:6379> linsert llista1 BEFORE desena setena
(integer) -1
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "segona"
3) "tercera"
4) "quarta"
5) "primera"
6) "sisena"
7) "cinquena"
```

LRANGE

Sintaxi

```
lrange clau inici final
```

Torna els elements de la llista que hi ha entre els index inici i final, ambdós inclosos. El primer element és el 0. Es poden posar valors negatius, sent -1 l'últim, -2 el penúltim, ...

Exemples

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "segona"
3) "tercera"
4) "quarta"
5) "primera"
6) "sisena"
7) "cinquena"
```

```
127.0.0.1:6379> lrange llista1 2 4
1) "tercera"
2) "quarta"
3) "primera"
```

```
127.0.0.1:6379> lrange llista1 1 -2
1) "segona"
2) "tercera"
3) "quarta"
4) "primera"
5) "sisena"
```

```
127.0.0.1:6379> lrange llista1 4 4
1) "primera"
```

LLEN

Sintaxi

```
llen clau
```

Torna el número d'elements de la llista

Exemples

```
127.0.0.1:6379> llen llista1
(integer) 7
```

LREM

Sintaxi

```
lrem clau número valor
```

Elimina elements de la llista que coincidisquen amb el valor proporcionat. Ja sabem que els valors es poden repetir. Amb el número indiquem quants elements volem que s'esborren: si posem 1 s'esborrarà el primer element amb aquest valor, si posem 2 s'esborraran els dos primers elements (els de més a l'esquerra) que tingen aquest valor. Si posem 0 s'esborraran tots els elements amb aquest valor

Exemples

```
127.0.0.1:6379> rpush llista1 segona
(integer) 8
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "segona"
3) "tercera"
4) "quarta"
5) "primera"
6) "sisena"
7) "cinquena"
8) "segona"
```

```
127.0.0.1:6379> lrem llista1 1 segona
(integer) 1
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "quarta"
2) "tercera"
3) "quarta"
4) "primera"
5) "sisena"
6) "cinquena"
7) "segona"
```

```
127.0.0.1:6379> lrem llista1 0 quarta
(integer) 2
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "tercera"
2) "primera"
3) "sisena"
4) "cinquena"
5) "segona"
```

LTRIM

Sintaxi

```
ltrim clau inici final
```

Elimina els elements que queden fora dels índex inici i final, és a dir elimina els que estiguen a l'esquerra d'inici, i els que estiguen a la dreta de final.

Exemples

```
127.0.0.1:6379> ltrim llista1 1 -2
OK
```

```
127.0.0.1:6379> lrange llista1 0 -1
1) "primera"
2) "sisena"
3) "cinquena"
```

Els **Sets** de **Redis** són conjunts de valors de tipus String no ordenats. Podrem afegir, actualitzar i esborrar aquestos elements de forma còmoda i eficient. No es permetran els valors duplicats.

A més **Redis** ens ofereix operacions interessants com la unió, intersecció i diferència de conjunts.

Com sempre, els comandos són específics, és a dir no ens valen els de Strings, Hash o List. Tots els comandos comencen per **S**.

Aquesta és la llista dels que veurem:

sadd , **smembers** , **sismember** , **scard** , **srem** , **spop** , **srandmember** , **sunion** , **sunionstore** , **sdiff** , **sdiffstore** , **sinter** , **sinterstore** , **smove**

SADD

Sintaxi

```
sadd clau valor1 valor2 valorN
```

Afegeix els valors al conjunt (creant la clau si és necessari). Recordem que l'ordre no és important, i que no es poden repetir els valors; si intentem introduir un repetit, no donarà error, però no l'introduirà. El comando tornarà el número d'elements que realment s'han afegit.

Exemples

```
127.0.0.1:6379> sadd colors roig verd blau  
(integer) 3
```

```
127.0.0.1:6379> sadd colors verd groc  
(integer) 1
```

SMEMBERS

Sintaxi

```
smembers clau
```

Torna tots els valors del conjunt. Si la clau no existeix tornarà un conjunt buit. Recordeu que l'ordre dels elements no és predecible

Exemples

```
127.0.0.1:6379> smembers colors  
1) "groc"  
2) "verd"  
3) "roig"  
4) "blau"
```

SISMEMBER

Sintaxi

```
sismember clau valor
```

Comprova si el valor està en el conjunt, tornant 1 en cas afirmatiu i 0 en cas negatiu.

Exemples

```
127.0.0.1:6379> sismember colors verd  
(integer) 1
```

```
127.0.0.1:6379> sismember colors negre  
(integer) 0
```

SCARD

Sintaxi

```
scard clau
```

Torna la cardinalitat, és a dir, el número d'elements del conjunt en l'actualitat.

Exemples

```
127.0.0.1:6379> scard colors
(integer) 4
```

SREM

Sintaxi

```
srem clau valor1 valor2 valorN
```

Elimina els valors del conjunt. Si el conjunt es queda buit, eliminarà la clau també. Si algun dels valor no és cap element del conjunt, senzillament s'ignorarà. El comando torna el número d'elements realment eliminat.

Exemples

```
127.0.0.1:6379> srem colors verd negre
(integer) 1
```

```
127.0.0.1:6379> smembers colors
1) "groc"
2) "roig"
3) "blau"
```

SPOP

Sintaxi

```
spop clau
```

Torna i elimina un valor **aleatori** del conjunt. Recordeu que a més de tornar-lo, l'elimina del conjunt.

Exemples

```
127.0.0.1:6379> smembers colors
1) "groc"
2) "roig"
3) "blau"
```

```
127.0.0.1:6379> spop colors
"groc"
```

```
127.0.0.1:6379> smembers colors
1) "roig"
2) "blau"
```

SRANDMEMBER

Sintaxi

```
randmember clau
```

Molt paregut a l'anterior. Torna un valor aleatori del conjunt, però en aquesta ocasió no l'elimina del conjunt.

Exemples

```
127.0.0.1:6379> randmember colors
"blau"
```

```
127.0.0.1:6379> smembers colors
1) "roig"
2) "blau"
```

SUNION

Sintaxi

```
union clau1 clau2 clauN
```

Torna la unió dels elements dels conjunts especificats. És una unió correcta, és a dir, no es repetirà cap valor.

No modifica cap conjunt, i el resultat únicament es torna, no es guarda en cap lloc de forma permanent.

Exemples

```
127.0.0.1:6379> smembers colors
1) "roig"
2) "blau"
```

```
127.0.0.1:6379> sadd colors1 verd roig groc
(integer) 3
```

```
127.0.0.1:6379> smembers colors1
1) "groc"
2) "roig"
3) "verd"
```

```
127.0.0.1:6379> sunion colors colors1
1) "verd"
2) "groc"
3) "roig"
4) "blau"
```

SUNIONSTORE

Sintaxi

```
sunionstore clau_destí clau1 clau2 clauN
```

Igual que l'anterior, però ara sí que es guarda el resultat de la unió en un conjunt, **clau_destí** (el primer especificat). Si la clau_destí ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:6379> smembers colors
1) "roig"
2) "blau"
```

```
127.0.0.1:6379> smembers colors1
1) "groc"
2) "roig"
3) "verd"
```

```
127.0.0.1:6379> sunionstore colors2 colors colors1
(integer) 4
```

```
127.0.0.1:6379> smembers colors2
1) "verd"
2) "groc"
3) "roig"
4) "blau"
```

SDIFF

Sintaxi

```
sdiff clau1 clau2 clauN
```

Torna la diferència dels elements del primer conjunt respecte de la unió de tots els altres. És a dir, torna els elements del primer conjunt que no pertanyen a cap dels altres.

No modifica cap conjunt, i el resultat únicament es torna, no es guarda en cap lloc de forma permanent.

Exemples

```
127.0.0.1:6379> smembers colors
1) "roig"
2) "blau"
```

```
127.0.0.1:6379> smembers colors1
1) "groc"
2) "roig"
3) "verd"
```

```
127.0.0.1:6379> sdiff colors1 colors
1) "verd"
2) "groc"
```

SDIFFSTORE

Sintaxi

```
sdiffstore clau_destí clau1 clau2 clauN
```

Igual que l'anterior, però ara sí que es guarda el resultat de la diferència en un conjunt, **clau_destí** (el primer especificat). Si la clau_destí ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:6379> smembers colors
1) "roig"
```

```
2) "blau"
```

```
127.0.0.1:6379> smembers colors1  
1) "groc"  
2) "roig"  
3) "verd"
```

```
127.0.0.1:6379> sdiffstore colors3 colors1 colors  
(integer) 2
```

```
127.0.0.1:6379> smembers colors3  
1) "verd"  
2) "groc"
```

SINTER

Sintaxi

```
sinter clau1 clau2 clauN
```

Torna la intersecció dels elements dels conjunts. És a dir, torna els elements que pertanyen a tots els conjunts especificats.

No modifica cap conjunt, i el resultat únicament es torna, no es guarda en cap lloc de forma permanent.

Exemples

```
127.0.0.1:6379> smembers colors  
1) "roig"  
2) "blau"
```

```
127.0.0.1:6379> smembers colors1  
1) "groc"  
2) "roig"  
3) "verd"
```

```
127.0.0.1:6379> sinter colors colors1  
1) "roig"
```

SINTERSTORE

Sintaxi

```
sinterstore clau_destí clau1 clau2 clauN
```

Igual que l'anterior, però ara sí que es guarda el resultat de la intersecció en un conjunt, **clau_destí** (el primer especificat). Si la clau_destí ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:6379> smembers colors  
1) "roig"  
2) "blau"
```

```
127.0.0.1:6379> smembers colors1  
1) "groc"  
2) "roig"  
3) "verd"
```

```
127.0.0.1:6379> sinterstore colors4 colors colors1  
(integer) 1
```

```
127.0.0.1:6379> smembers colors4  
1) "roig"
```

SMOVE

Sintaxi

```
smove clau_font clau_destí valor
```

Meneja el valor del conjunt origen (el primer conjunt) al conjunt destí (el segon). Això suposarà eliminar-lo del primer i afegir-lo al segon. Tornarà 1 si l'ha menejat, i 0 si no l'ha menejat.

Exemples

```
127.0.0.1:6379> smembers colors  
1) "roig"  
2) "blau"
```

```
127.0.0.1:6379> smembers colors1  
1) "groc"  
2) "roig"
```

```
3) "verd"
```

```
127.0.0.1:6379> smove colors1 colors verd  
(integer) 1
```

```
127.0.0.1:6379> smembers colors  
1) "verd"  
2) "roig"  
3) "blau"
```

```
127.0.0.1:6379> smembers colors1  
1) "groc"  
2) "roig"
```


Els **Sets Ordenats (Sorted Set)** de **Redis** són Sets que a més de guardar els valors, guarden també una **puntuació (score)** per a cada valor, i **Redis** mantindrà el conjunt **ordenat** per aquesta puntuació.

Els valors no es podran repetir, però sí les puntuacions.

Alguns dels comandos seran iguals que els del **Set**, ja que un conjunt ordenat no deixa de ser un conjunt, però amb la informació de la puntuació (i que no es poden repetir els valors). En aquesta ocasió començaran per **Z**.

I aquesta és la llista dels que veurem:

zadd , **zcard** , **zscore** , **zcount** , **zrange** , **zrangebyscore** , **zrank** , **zrem** , **zremrangebyscore** , **zincrby**

ZADD

Sintaxi

```
zadd clau puntuació1 valor1 puntuació2 valor2 puntuacióN valorN
```

Afegeix els valors al conjunt (creant la clau si és necessari) amb les puntuacions corresponents. Les puntuacions seran Strings de valors reals (float). No es poden repetir els valors, però sí les puntuacions. Si intentem introduir un valor repetit, el que farà serà actualitzar la puntuació. El comando tornarà el número d'elements que realment s'han afegit.

Exemples

```
127.0.0.1:6379> zadd puntuacions 1 Nom1 2 Nom2 5 Nom3 4 Nom4  
(integer) 4
```

```
127.0.0.1:6379> zrange puntuacions 0 -1  
1) "Nom1"  
2) "Nom2"  
3) "Nom4"  
4) "Nom3"
```

ZCARD

Sintaxi

```
zcard clau
```

Torna la cardinalitat, és a dir, el número d'elements del conjunt ordenat en l'actualitat.

Exemples

```
127.0.0.1:6379> zcard puntuacions  
(integer) 4
```

ZSCORE

Sintaxi

```
zscore clau valor
```

Torna la puntuació (score) del valor especificat del conjunt ordenat. Si no existeix el valor o no existiaix la clau, torna nil.

Exemples

```
127.0.0.1:6379> zscore puntuacions Nom3  
"5"
```

```
127.0.0.1:6379> zscore puntuacions Nom7  
(nil)
```

ZCOUNT

Sintaxi

```
zcount clau min max
```

Torna el número de valors que estan entre les puntuacions especificades (ambdues incloses).

Exemples

```
127.0.0.1:6379> zcount puntuacions 2 5
(integer) 3
```

ZRANGE

Sintaxi

```
zrange clau inici final [withscores]
```

Torna els elements del conjunt ordenat que hi ha entre els index inici i final, ambdós inclosos. I es trauen per ordre ascendent de puntuació. El primer element és el 0. Es poden posar valors negatius, sent -1 l'últim, -2 el penúltim, ... Opcionalment podem posar **WITHSCORES** per a que ens torne també la puntuació de cada element

Exemples

```
127.0.0.1:6379> zrange puntuacions 0 -1
1) "Nom1"
2) "Nom2"
3) "Nom4"
4) "Nom3"
```

```
127.0.0.1:6379> zrange puntuacions 0 -1 withscores
1) "Nom1"
2) "1"
3) "Nom2"
4) "2"
5) "Nom4"
6) "4"
7) "Nom3"
8) "5"
```

Si vulguérem traure el conjunt en ordre invers de puntuació, utilitzaríem el comando **ZREVRANGE** (reverse range).

```
127.0.0.1:6379> zrevrange puntuacions 0 -1 withscores
1) "Nom3"
2) "5"
3) "Nom4"
4) "4"
5) "Nom2"
6) "2"
7) "Nom1"
8) "1"
```

ZRANGEBYSCORE

Sintaxi

```
zrangebyscore clau min max [withscores]
```

Torna els elements del conjunt ordenat que tenen una puntuació compresa entre **min** i **max** (ambdues incloses). I es trauen per ordre ascendent de puntuació. Opcionalment podem posar **WITHSCORES** per a que ens torne també la puntuació de cada element.

Exemples

```
127.0.0.1:6379> zrangebyscore puntuacions 2 5
1) "Nom2"
2) "Nom4"
3) "Nom3"
```

```
127.0.0.1:6379> zrangebyscore puntuacions 2 5 withscores
1) "Nom2"
2) "2"
3) "Nom4"
4) "4"
5) "Nom3"
6) "5"
```

Si vulguérem que les puntuacions foren estrictament majors que la puntuació mínima i/o estrictament menor que la puntuació màxima, posaríem un **parèntesi** davant de **min** i/o **max**:

```
127.0.0.1:6379> zrangebyscore puntuacions 2 (5 withscores
1) "Nom2"
2) "2"
3) "Nom4"
4) "4"
```

I si vulguérem traure el conjunt en ordre invers de puntuació, utilitzaríem el comando **ZREVRANGEBYSCORE** (reverse range). Cuideu que com va en ordre invers, ara el valor màxim ha de ser el primer, i el mínim el segon.

```
127.0.0.1:6379> zrevrangebyscore puntuacions 5 2 withscores
1) "Nom3"
2) "5"
```

```
3) "Nom4"  
4) "4"  
5) "Nom2"  
6) "2"
```

ZRANK

Sintaxi

```
zrank clau valor
```

Torna el número d'ordre de l'element amb el valor especificat. El primer valor és el 0. Si no existeix, torna **nil**.

Exemples

```
127.0.0.1:6379> zrank puntuacions Nom1  
(integer) 0
```

```
127.0.0.1:6379> zrank puntuacions Nom4  
(integer) 2
```

```
127.0.0.1:6379> zrank puntuacions Nom7  
(nil)
```

Si volem saber el número d'ordre però des del final de la llista (en ordre invers), hem d'utilitzar **ZREVRANK**:

```
127.0.0.1:6379> zrevrank puntuacions Nom1  
(integer) 3
```

```
127.0.0.1:6379> zrevrank puntuacions Nom4  
(integer) 1
```

ZREM

Sintaxi

```
zrem clau valor1 valor2 valorN
```

Elimina els elements amb els valors especificats. Si algun valor no existeix, senzillament l'ignora. Torna el número d'elements realment eliminats.

Exemples

```
127.0.0.1:6379> zrem puntuacions Nom1  
(integer) 1
```

```
127.0.0.1:6379> zrange puntuacions 0 -1 withscores  
1) "Nom2"  
2) "2"  
3) "Nom4"  
4) "4"  
5) "Nom3"  
6) "5"
```

ZREMRANGEBYSCORE

Sintaxi

```
zremrangebyscore clau min max
```

Elimina els elements amb puntuació compresa entre el mínim i el màxim de forma inclusiva. Si volem fer-ho de forma excusiva (sense incloure les puntuacions dels extrems) posarem un parèntesi davant del mínim i/o el màxim. Torna el número d'elements realment eliminats.

Exemples

```
127.0.0.1:6379> zremrangebyscore puntuacions (2 4  
(integer) 1
```

```
127.0.0.1:6379> zrange puntuacions 0 -1 withscores  
1) "Nom2"  
2) "2"  
3) "Nom3"  
4) "5"
```

ZINCRBY

Sintaxi

```
zincrby clau increment valor
```

Incrementa la puntuació de l'element especificat. El valor de la puntuació a incrementar és un número real. Torna el valor la puntuació final de l'element. Si l'element no existia, l'inserirà, assumint una puntuació inicial de 0.

Exemples

```
127.0.0.1:6379> zincrby puntuacions 1.5 Nom2  
"3.5"
```

```
127.0.0.1:6379> zincrby puntuacions 2.75 Nom5  
"2.75"
```

```
127.0.0.1:6379> zrange puntuacions 0 -1 withscores  
1) "Nom5"  
2) "2.75"  
3) "Nom2"  
4) "3.5"  
5) "Nom3"  
6) "5"
```

La utilització des de Java és molt senzilla, i podrem utilitzar tots els comandos que hem vist.

Haurem d'incorporar en el projecte una llibreria. Podem utilitzar per exemple la de **Jedis** (acrònim de **Java Redis**). A la següent adreça la podeu trobar (no és l'última versió, però ens val):

<http://search.maven.org/remotecontent?filepath=redis/clients/jedis/2.4.2/jedis-2.4.2.jar>

Per a fer proves, creem un nou projecte anomenat **Tema8**, i incorporem la llibreria de **Jedis**. Per tenir-ho un poc ordenat, creem un paquet anomenat **proves**.

Connexió

La connexió és tan senzilla com el següent:

```
Jedis con = new Jedis("localhost");
con.connect();
```

És a dir, obtenim un objecte **Jedis** passant-li al constructor l'adreça del servidor, i després connectem amb el mètode **connect**.

Si el servidor no el tenim en la mateixa màquina, només haurem de substituir **localhost** per l'adreça on estiga el servidor.

L'objecte **Jedis** representarà una connexió amb el servidor **Redis**. Tots els comandos que hem vist per a utilitzar des del client de Redis, seran mètodes d'aquest objecte. Només haurem d'anar amb compte amb el que ens torna el servidor quan fem una petició. Moltes vegades serà un **String**, però en moltes altres ocasions seran col·leccions: sets, lists, ...

Per a tancar la connexió:

```
con.close();
```

Comandos que tornen Strings

Tots els comandos que vam veure seran ara mètodes de l'objecte **Jedis**. En concret obtenir el valor d'una clau (comando **get clau**) serà el mètode **get** al qual li passarem la clau com a paràmetre:

Ací tenim ja la primera prova:

```
import redis.clients.jedis.Jedis;

public class Prova1 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        System.out.println(con.get("salutacio"));

        con.close();
    }
}
```

Si volem guardar una clau amb un valor, utilitzarem el mètode **set(clau,valor)**, al qual com veiem li hem de passar els dos paràmetres:

```
import redis.clients.jedis.Jedis;

public class Prova2 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        String valor = "Aquesta clau és una clau creada des de Java";
        con.set("clau_Java", valor);

        System.out.println(con.get("clau_Java"));
        con.close();
    }
}
```

Comandos que tornen conjunts

En moltes ocasions, el que ens tornarà **Redis** no és un únic String, sinó un conjunt de Strings. Açò serà cert quan treballem amb **Sets**, però també en moltes altres ocasions. Per exemple quan demanem unes quantes claus amb **MGET**, o quan utilitzem **KEYS** per a que ens torne les

claus que coincideixen amb el patró.

En algunes ocasions ho haurem d'arreglar amb un objecte **Set**, quan no importe l'ordre. En altres ocasions amb un objecte **List** quan aquest ordre sí que importe.

Per exemple, si volem obtenir amb **MGET** els valors d'unes quantes claus, sí que importa l'ordre (el primer valor és de la primera clau, el segon de la segona). Aleshores ho obtindrem en un **List**:

```
import java.util.List;
import redis.clients.jedis.Jedis;

public class Prova3 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        List<String> c = con.mget("mes1", "mes2", "mes3");
        for (String s : c)
            System.out.println(s);

        con.close();
    }
}
```

Però en canvi si volem obtenir totes les claus utilitzarem el mètode **keys** passant-li el patró com a paràmetre. L'ordre no importa, i a més no el podem predir. Per tant hem d'arreglar el resultat amb un **Set**:

```
import java.util.Set;
import redis.clients.jedis.Jedis;

public class Prova4 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        Set<String> c = con.keys("*");
        for (String s : c)
            System.out.println(s);

        con.close();
    }
}
```

I evidentment també serà el cas quan accedim als tipus de dades **List**, **Set** i segurament també **Hash**.

Per a accedir a tot el contingut d'un **List** utilitzem per exemple **lrange 0 -1**. Si utilitzem aquest mètode de **Jedis** ens tornarà un **List** (de Java):

```
import java.util.List;
import redis.clients.jedis.Jedis;

public class Prova5 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        List<String> ll = con.lrange("llista1", 0, -1);
        for (String e : ll)
            System.out.println(e);

        con.close();
    }
}
```

Si és un **Set** accedirem amb el mètode **smembers** que tornarà un **Set** (de Java):

```
import java.util.Set;
import redis.clients.jedis.Jedis;

public class Prova6 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        Set<String> s = con.smembers("colors");
        for (String e : s)
            System.out.println(e);

        con.close();
    }
}
```

I en el cas de **Hash**, amb el mètode **hkeys** podem obtenir tots els camps (subclaus), i a partir d'ells els seus valors. El que torna **hkeys** és un **Set** (de Java).

```
import java.util.Set;
import redis.clients.jedis.Jedis;
```

```

public class Prova7 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        Set<String> subcamps = con.hkeys("empleat_1");
        for (String subcamp : subcamps)
            System.out.println(subcamp + ": " + con.hget("empleat_1", subcamp));

        con.close();
    }
}

```

Tractament dels conjunts ordenats

Els conjunts ordenats (**Sorted Sets**) tenen mètodes específics, igual que tots els tipus. Alguns d'aquests mètodes tornen Strings, i un altres conjunts (**Set de Java**). No hi ha problema amb aquests tipus, que ja els hem tractat.

Però hi ha uns mètodes que tornen llistes **amb més d'un valor per cada element**. És el cas dels mètodes **ZRANGE** (amb les variants **ZREVRANGE**, **ZRANGEBYSCORE** i **ZREVRANGEBYSCORE**), que tenen la possibilitat de dur el paràmetre **WITHSCORES**. En aquest cas cada element constarà del valor i de la puntuació. El que tornaran els mètodes és un **Set de Tuples**, objecte proporcionat per **Jedis**, que disposarà dels mètodes **getElement()** per al valor i **getScore()** per a la puntuació.

```

import java.util.Set;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;

public class Prova8 {

    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();

        Set<Tuple> conjOrd = con.zrangeWithScores("puntuacions", 0, -1);
        for (Tuple t : conjOrd)
            System.out.println(t.getElement() + " ---> " + t.getScore());

        con.close();
    }
}

```

Segurament **MongoDB** és el més famós dels Sistemes Gestors de Bases de Dades **NoSQL**.

El nom de **MongoDB** prové de la paraula anglesa *humongous*, que significa enorme, que és el propòsit d'aquesta Base de Dades: guardar grans quantitats d'informació. És de codi obert i està programada en C++. El va crear l'empresa **10gen** (actualment **MongoDB Inc.**)

És una SGBD **Documental**, és a dir, que servirà per a guardar documents. La manera interna de guardar-los és en format **BSON** (Binary JSON) que en essència és una variant del JSON per a poder guardar físicament les dades d'una manera més eficient.

En un servidor Mongo poden haver més d'una Base de Dades (encara que nosaltres només en gasterem una: **test**).

- En cada Base de Dades la informació es guardarà en **col·leccions**.
- Cada col·lecció constarà d'uns quants **documents**.
- I cada document seran una serie de dades guardades en forma de **clau-valor**, dels tipus suportats per MongoDB, i amb el format JSON (en realitat BSON)

Per tant, en Mongo no hi ha taules. Mirem uns exemples de documents JSON per a guardar la informació de llibres i autors. Depenen de com s'haja d'accedir a la informació ens podem plantejar guardar els llibres amb els seus autors, o guardar els autors, amb els seus llibres. Fins i tot ens podríem guardar els dos, per a poder accedir de totes les maneres, encara que és a costa de doblar la informació.

De la primera manera, guardant els llibres amb el seu autor, podríem tenir documents amb aquesta estructura, que es podrien guardar en una col·lecció anomenada **Llibres**:

```
{
  _id:101,
  _titol:"El secret de Khadrell",
  autor: {
    nom:"Pep",
    cognoms:"Castellano Puchol",
    any_naixement:1960
  },
  isbn:"84-95620-72-3"
},
{
  _id:102,
  _titol:"L'Ombra del Vent",
  autor: {
    nom:"Carlos",
    cognoms:"Ruiz Zafon",
    pais:"Espanya"
  },
  pagines:490,
  editorial:"Planeta"
}
```

Observeu com els objecte no tenen per qè tenir la mateixa estructura. La manera d'accedir al nom d'un autor seria aquesta: *objecte.autor.nom*

Una manera alternativa de guardar la informació, com havíem comentat abans seria organitzar per autors, amb els seus llibres. D'aquesta manera podríem anar omplint la col·lecció **Autors** amb un o més documents d'aquest estil:

```
{
  _id: 201,
  nom:"Pep",
  cognoms:"Castellano Puchol",
  any_naixement:1960,
  llibres: [
    {
      titol:"El secret de Khadrell",
      isbn:"84-95620-72-3"
    },
    {
      titol:"Habitació 502",
      editorial:"Tabarca"
    }
  ]
},
{
  _id:202,
  nom:"Carlos",
  cognoms:"Ruiz Zafon",
  pais:"Espanya",
  llibres: [
    {
      titol:"L'Ombra del Vent",
      pagines:490,
      editorial:"Planeta"
    }
  ]
}
```

Observeu com per a un autor, ara tenim un array (els claudàtors: []) amb els seus llibres.

Quina de les dues maneres és millor per a guardar la informació? Doncs depén de l'accés que s'haja de fer a les dades. La millor serà segurament aquella que depenent dels accessos que s'hagen de fer, torne la informació de forma més ràpida.

Podrem instal·lar MongoDB en qualsevol plataforma. I fins i tot sense tenir permisos d'administrador, com veurem en el cas d'Ubuntu.

Instal·lació en Linux

Per a poder fer la instal·lació més bàsica, podrem fer-lo sense permisos d'administrador. Si els tenim tot és més còmode, però si no en tenim també ho podem fer, com veurem i remarcarem a continuació.

De la pàgina de **MongoDB** (<https://mongodb.org>) ens baixem la versió apropiada per al nostre Sistema Operatiu. Observeu com en el cas de Linux hi ha moltes versions. En el cas d'**Ubuntu 14.04 de 64 bits**, aquest fitxer és: **mongodb-linux-x86_64-ubuntu1404-3.2.1.tgz**. Però recordeu que us heu d'assegurar de la versió

Senzillament descomprimem aquest fitxer on vulguem, i ja estarà feta la instal·lació bàsica.

Per defecte el directori de la Base de Dades és **/data/db**

L'únic problema que podríem tenir si no som administradors és que no tinguem permís per crear aquest directori. Aleshores crearem un altre directori i en el moment d'arrancar el servidor, li especificarem aquest lloc.

La manera d'arrancar el servidor serà:

```
<directori arrel MongoDB>/bin/mongod
```

Opcionalment li podem dir on està la Base de Dades (si no ho especifiquem assumirà que està en **/data/db**):

```
<directori arrel MongoDB>/bin/mongod --dbpath <directori de la BD>
```

Resumint, i estant situats al directori on hem descomprimit MongoDB:

- Si som administradors:
 - Creem el directori de dades:

```
mkdir /data
mkdir /data/db
```

- Arranquem el servidor:

```
./bin/mongod
```

- Si no som administradors:
 - Creem el directori de dades:

```
mkdir data
mkdir data/db
```

- Arranquem el servidor:

```
./bin/mongod --dbpath ./data/db
```

La següent imatge il·lustra aquesta segona opció:

```

• curs@Ubuntu: ~/mongodb-linux-x86_64-ubuntu1404-3.2.1
Fitxer Edita Visualitza Cerca Terminal Ajuda
curs@Ubuntu:~$ cd mongodb-linux-x86_64-ubuntu1404-3.2.1
curs@Ubuntu:~/mongodb-linux-x86_64-ubuntu1404-3.2.1$ mkdir data
curs@Ubuntu:~/mongodb-linux-x86_64-ubuntu1404-3.2.1$ mkdir data/db
curs@Ubuntu:~/mongodb-linux-x86_64-ubuntu1404-3.2.1$ ./bin/mongod --dbpath ./data/db
2016-02-17T13:58:34.474+0100 I CONTROL [initandlisten] MongoDB starting : pid=7445 port=27017 dbpath=./data/db 64-bit
2016-02-17T13:58:34.474+0100 I CONTROL [initandlisten] db version v3.2.1
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] git version: a14d55980c2cdc565d4704a7e3ad37e4e535c1b2
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1f 6 Jan 2014
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] allocator: tcmalloc
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] modules: none
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] build environment:
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] distmod: ubuntu1404
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] distarch: x86_64
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] target_arch: x86_64
2016-02-17T13:58:34.476+0100 I CONTROL [initandlisten] options: { storage: { dbPath: "./data/db" } }
2016-02-17T13:58:34.500+0100 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=1G,session_max=20000,
ax=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(
000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2016-02-17T13:58:34.580+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'alw
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'alwa
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.582+0100 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '
c.data'
2016-02-17T13:58:34.583+0100 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-02-17T13:58:34.663+0100 I NETWORK [initandlisten] waiting for connections on port 27017

```

Una vegada en marxa el servidor, no hem de tancar aquesta terminal, ja que parariem el servidor. Per a connectar un client, obrim una segona terminal i executem el client **mongo**:

```
./bin/mongo
```

```

• curs@Ubuntu: ~/mongodb-linux-x86_64-ubuntu1404-3.2.1
Fitxer Edita Visualitza Cerca Terminal Ajuda
curs@Ubuntu:~/mongodb-linux-x86_64-ubuntu1404-3.2.1$ ./bin/mongo
MongoDB shell version: 3.2.1
connecting to: test
Server has startup warnings:
2016-02-17T13:58:34.580+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'alw
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'alwa
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten]
>

```

Per a provar el seu funcionament, anem a fer un parell de comandos: un per a guardar un document i un altre per a recuperar-lo.

Per a qualsevol operació s'ha de posar **db** seguit del nom de la col·lecció, i després l'operació que volem fer. Amb el sgüent:

```
> db.example.save( {msg:"Hola, què tal?"} )
```

Ens contestarà:

```
WriteResult({ "nInserted" : 1 })
```

Indicant que ha inserit un document en la col·lecció **example** (si no estava creada, la crearà).

I amb el següent comando recuperem la informació:

```
> db.example.findOne()
```

Que ens tornarà:

```
{ "_id" : ObjectId("56cc130590d651d45ef3d3be"), "msg" : "Hola, què tal?" }
```

Tot ho fa en la mateixa terminal, i a cadascú de nosaltres ens donarà un número diferent en **ObjectId**. En la següent imatge es veuen les dues operacions:

```
curs@Ubuntu: ~/mongodb-linux-x86_64-ubuntu1404-3.2.1
Fitxer Edita Visualitza Cerca Terminal Ajuda
curs@Ubuntu:~/mongodb-linux-x86_64-ubuntu1404-3.2.1$ ./bin/mongo
MongoDB shell version: 3.2.1
connecting to: test
Server has startup warnings:
2016-02-17T13:58:34.580+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'alw
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] **          We suggest setting it to 'never'
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten]
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'alwa
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten] **          We suggest setting it to 'never'
2016-02-17T13:58:34.581+0100 I CONTROL [initandlisten]
> db.example.save( {msg:"Hola, què tal?"} )
WriteResult({ "nInserted" : 1 })
> db.example.findOne()
{ "_id" : ObjectId("56cc130590d651d45ef3d3be"), "msg" : "Hola, què tal?" }
>
```

En realitat estem connectats a una Base de Dades anomenada **test**. Podem crear i utilitzar més d'una Base de Dades, però en aquest curs tindràs més que suficient amb aquesta Base de Dades. Per a comprovar-ho podem executar la següent sentència, que ens torna el nom de la Base de Dades:

```
> db.getName()
test
```

Instal·lació en Windows

No ofereix cap dificultat. Ens baixem la versió apropiada de MongoDB per a Windows, depenent de si és de 32 o 64 bits la nostra versió, que resultarà ser un .msi directament executable. En el moment de fer aquestos apunts, el de la versió de 64 bits era el fitxer:

[mongodb-win32-x86_64-2008plus-ssl-3.2.3-signed.msi](#)

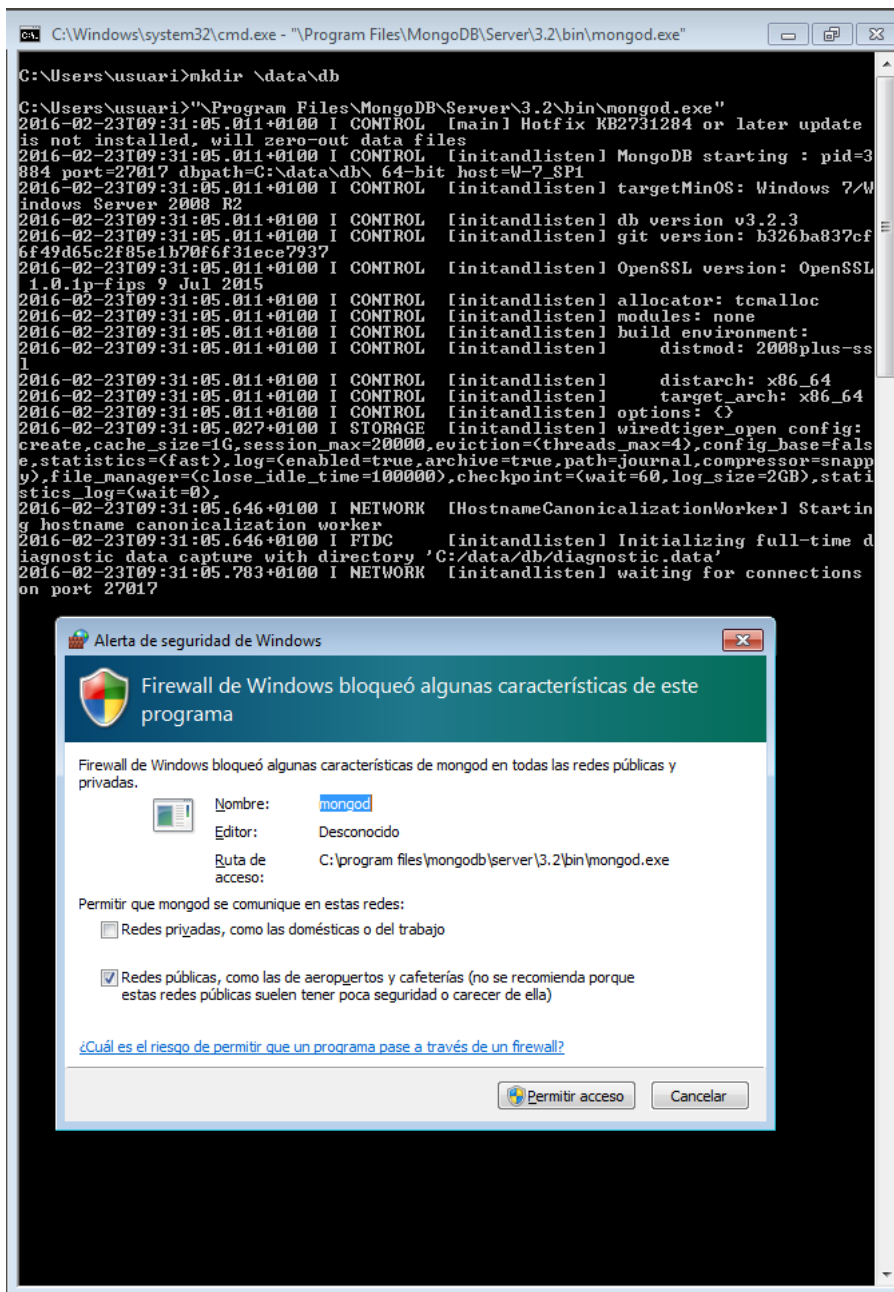
Una vegada baixat, executem el fitxer. Haurem d'acceptar la llicència, instal·lar la versió completa, i acceptar quan Windows ens avise que un programa vol instal·lar software. El de sempre.

Com en el cas de Linux, abans d'executar el servidor haurem de tenir el directori creat. Per defecte el directori serà **data\db**

Aqueste serien les ordres per a crear el directori i després arrancar el servidor:

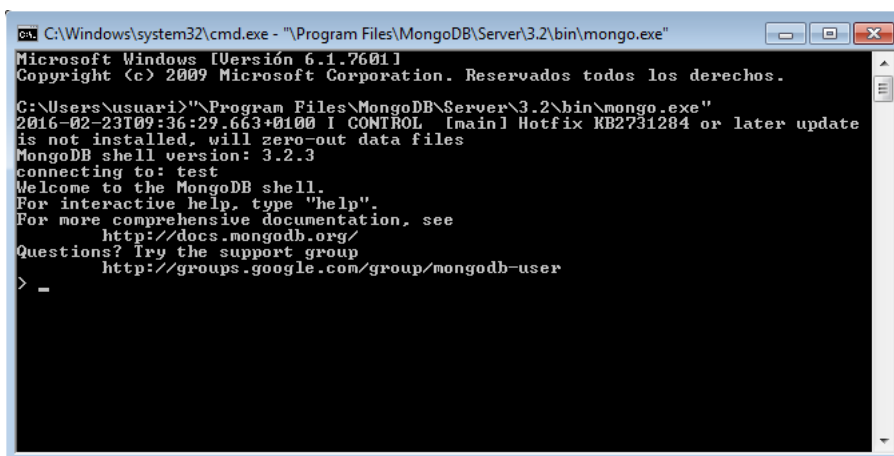
```
mkdir \data\db
C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe
```

En aquest a imatge s'observa que en intentar posar en marxa el servidor, el Firewall de Windows ho detecta, i sol·licita permès per posar-lo en marxa. Acceptem i prou:



Per a connectar-nos com a clients, ho haurem de fer des d'una altra terminal, ja que si tanquem aquesta pararem el servidor. El programa és **mongo.exe**:

```
C:\Program Files\MongoDB\Server\3.2\bin\mongo.exe
```



Per a provar el seu funcionament, anem a fer un parell de comandos: un per a guardar un document i un altre per a recularar-lo.

Per a qualsevol operació s'ha de posar **db** seguit del nom de la col·lecció, i després l'operació que volem fer. Amb el sgüent:

```
> db.exemple.save( {msg:"Hola, què tal?"} )
```

Ens contestarà:

```
WriteResult({ "nInserted" : 1 })
```

Indicant que ha inserit un document en la col·lecció **exemple** (si no estava creada, la crearà).

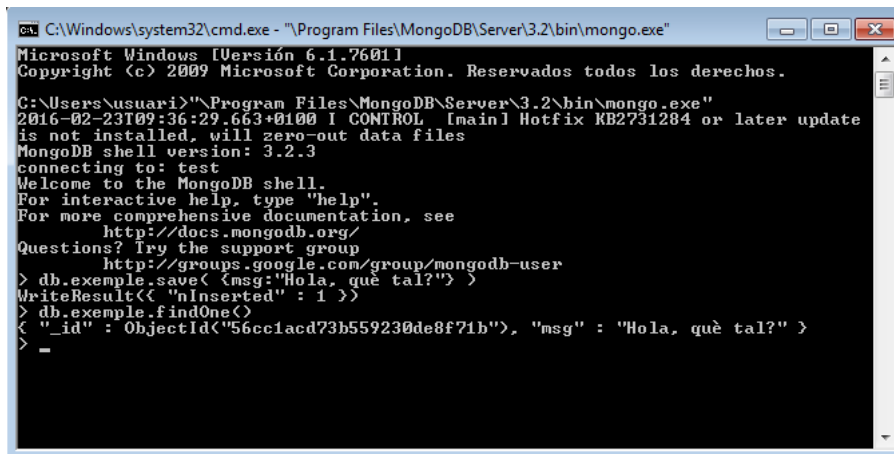
I amb el següent comando recuperem la informació:

```
> db.exemple.findOne()
```

Que ens tornarà:

```
{ "_id" : ObjectId("56cc1acd73b559230de8f71b"), "msg" : "Hola, què tal?" }
```

Tot ho fa en la mateixa terminal, i a cadascú de nosaltres ens donarà un número diferent en **ObjectId**. En la següent imatge es veuen les dues operacions:



```
C:\Windows\system32\cmd.exe - "Program Files\MongoDB\Server\3.2\bin\mongo.exe"
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\usuari>"Program Files\MongoDB\Server\3.2\bin\mongo.exe"
2016-02-23T09:36:29.663+0100 I CONTROL [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
MongoDB shell version: 3.2.3
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
> db.exemple.save( {msg:"Hola, què tal?"} )
WriteResult({ "nInserted" : 1 })
> db.exemple.findOne()
{ "_id" : ObjectId("56cc1acd73b559230de8f71b"), "msg" : "Hola, què tal?" }
>
```

En realitat estem connectats a una Base de Dades anomenada **test**. Podem crear i utilitzar més d'una Base de Dades, però en aquest curs tindràs més que suficient amb aquesta Base de Dades. Per a comprovar-ho podem executar la següent sentència, que ens torna el nom de la Base de Dades:

```
> db.getName()
test
```

Començarem la utilització de MongoDB des de la consola que havíem arrancat al final de la instal·lació.

Recordeu que tindrem dues terminals:

- Una amb el servidor en marxa (i que no hem de tancar): **mongod**
- Una altra amb el client que es connecta al servidor: **mongo**

En aquesta última consola del client podem utilitzar sentències del llenguatge **JavaScript**, però el que més ens interessarà, evidentment, són les sentències d'accés a dades. Del llenguatge Javascript pràcticament l'únic que utilitzarem són variables i algunes funcions.

Utilització de variables

Com comentàvem el que més utilitzarem del llenguatge **JavaScript** és la utilització de variables, que ens pot ser molt útil en algunes ocasions. Podrem utilitzar-les durant la sessió, però evidentment no perduraran d'una sessió a l'altra.

Per a definir una variable podem posar opcionalment davant la paraula reservada **var**, però no és necessari. Posarem el nom de la variable, el signe igual, i a continuació el valor de la variable, que pot ser una constant, o una expressió utilitzant constants, operadors, altres variables, funcions de Javascript, ...

Especialment interessant són les variables que poden contenir un document JSON.

Per exemple:

```
> a = 30
30
> b=a/4
7.5
> Math.sqrt(b)
2.7386127875258306
> doc = {camp1: "Hola", camp2: 45, camp3: new Date()}
{
  "camp1" : "Hola",
  "camp2" : 45,
  "camp3" : ISODate("2016-02-23T19:10:29.560Z")
}
>
```

Una variable de tipus JSON es podrà modificar molt fàcilment, tota ella, o algun dels elements. Per a arribar als elements posarem **nom_variable.nom_camp**:

```
> doc.camp4 = 3.141592
3.141592
```

```
> doc.camp5 = [ 2 , 4 , 6 , 8 ]
[ 2, 4, 6, 8 ]
```

I si ara intentem traure el contingut de la variable:

```
> doc
{
  "camp1" : "Hola",
  "camp2" : 45,
  "camp3" : ISODate("2016-02-24T22:31:12.724Z"),
  "camp4" : 3.141592,
  "camp5" : [
    2,
    4,
    6,
    8
  ]
}
>
```

També hem de fer constar que en un document, que serà de tipus JSON (pràcticament), serà un conjunt de parelles clau-valor, amb algunes restriccions:

- El document (que moltes vegades l'associarem a objecte de JSON) va entre claus ({ })
- Els elements d'un objecte van separats per comes, i són parelles clau-valor.
- La clau no pot ser nula, ni repetir-se en el mateix objecte (sí en diferents objectes, clar)
- Els valors són dels tipus que veurem en la pregunta 3.2.2
- Un document guardat ha de contenir obligatòriament un camp anomenat **_id**, i que contindrà un valor únic en la col·lecció i servirà per a identificar-lo. Si en guardar un document no li hem posat **camp_id**, el generarà automàticament MongoDB.

Els valors dels elements, és a dir de les parelles clau valor, poden ser d'una ampla gama de tipus. Fem un ràpid repàs.

En els exemples que van a continuació definim senzillament parelles clau-valor dels diferents tipus, o en tot cas ens ho guardem en variables, però no guardarem encara en la Base de Dades (ho farem en la següent pregunta).

Quan guardem en una variable es mostrarà el prompt, la definició de la variable i després el resultat d'haver guardat la variable. Utilitzarem requadres blancs. Els requadres grocs són únicament de la definició d'una clau-valor d'un determinat tipus

NULL

Més que un tipus de dades és un valor, millor dit, l'absència de valor

```
{ "x" : null }
```

BOOLEAN

El tipus booleà, que pot agafar els valors true o false.

```
{ "x" : true }
```

```
{ "y" : false }
```

NUMBER

Per defecte, el tiups de dades numèrics serà el de coma flotant (**float**), simple precisió. Si volem un altre tiups (enter, doble precisió, ...) ho haurem d'indicar expressament. Així els dos següents valors són float:

```
{ "x" : 3.14 }
```

```
{ "y" : 3 }
```

Si volem que siga estrictament enter, per exemple, haurem d'utilitzar una funció de conversió:

```
{ "x" : NumberDouble("3.14") }
```

```
{ "y" : NumberInt("3") }
```

STRING

Es pot guardar qualsevol cadena amb caràcters de la codificació UTF-8

```
{ x : "Hola, què tal?" }
```

DATE

Es guarda data i hora, i internament es guarden en milisegons des de l'any inicial. No es guara el *Time zone*, és a dir, la desviació respecte a l'hora internacional.

```
{ x : ISODate("2016-02-24T11:15:27.471Z") }
```

Normalment utilitzarem funcions de tractament de la data-hora. L'anterior era per a convertir el string en data-hora. La següent és per a obtenir la data-hora actual:

```
{ x : new Date() }
```

És a dir, que si no posem paràmetre, ens dona la data-hora actual. Però li podem posar com a paràmetre la data-hora que volem que genere. En aquest exemple, només posem data, per tant l'ho serà les 00:00:

```
> z = new Date("2016-08-01")  
ISODate("2016-08-01T00:00:00Z")
```

En aquest sí que posem una determinada hora, i observeu com hem depositar la T (Time) entre el dia i l'hora:

```
> z = new Date("2016-08-01T10:00")  
ISODate("2016-08-01T08:00:00Z")
```

És molt important que posem sempre **New Date()** per a generar una data-hora. Si posem únicament **Date()**, el que estem generant és un string (amb la data i hora, però un string):


```
> z = Date("2016-08-01")
Fri Feb 26 2016 08:30:13 GMT+0100 (CET)
```

ARRAY

És un conjunt d'elements, cadascun de qualsevol tipus, encara que el més habitual és que siguin del mateix tipus. Van entre claudàtors (`[]`) i els elements separats per comes.

```
{ x : [ 2 , 4 , 6 , 8 ] }
```

Com comentàvem, cada element de l'array pot ser de qualsevol tipus:

```
{ y : [ 2 , 3.14 , "Hola" , new Date() ] }
```

En MongoDB podem treballar molt bé amb arrays, i tindrem operacions per a poder buscar dins de l'array, modificar un element, crear índex, ...

DOCUMENTS (OBJECTES)

Els documents poden contenir com a elements uns altres documents (**objectes** en la terminologia JSON, però **documents** en la terminologia de MongoDB).

Com sempre van entre claus (`{ }`), i els elements que contindran van separats per comes i seran parelles clau-valor de qualsevol tipus (fins i tot altres documents).

```
{ x : { a : 1 , b : 2 } }
```

Posar documents dins d'uns altres documents (el que s'anomena *embedded document*) ens permet guardar la informació d'una manera més real, no tan plana. Així per exemple, les dades d'una persona les podríem definir de la següent manera. Les posarem en una variable, per veure després com podem accedir als diferents elements, encara que el més normal serà guardar-lo en la Base de Dades (amb **insert()** o **save()**). Si copiem el que va a continuació al terminal de Mongo, ens apareixerà amb un format estrany. És perquè la sentència d'assignació a la variable ocupa més d'una línia, i apareixeran 3 punts al principi per a indicar que continua la sentència. Però funcionarà perfectament :

```
doc = {
  nom: "Joan Martí",
  adreça: {
    carrer: "Major",
    número: 1,
    població: "Castelló"
  },
  telèfons : [964223344, 678345123]
}
```

Observeu com aquesta estructura que ha quedat tan clara, segurament en una Base de Dades Relacional ens hauria tocat guardar en 3 taules: la de persones, la d'adreces i la de telèfons.

Per a accedir als elements d'un document posàvem el punt. Doncs el mateix per als elements d'un document dins d'un document. I també podem accedir als elements d'un array, posant l'índex entre claudàtors.

```
> doc.nom
Joan Martí

> doc.adreça
{ "carrer" : "Major", "número" : 1, "població" : "Castelló" }

> doc.adreça.carrer
Major

> doc.telèfons
[ 964223344, 678345123 ]

> doc.telèfons[0]
964223344
```

OBJECT ID

És un tipus que defineix MongoDB per a poder obtenir valors únics. És el valor per defecte de l'element **_id**, necessari en tot document (atenció: en un document, no en un element de tipus document que hem dit equivalent a l'objecte de JSON). És un número long, és a dir que utilitza 24 bytes.

Farem proves de la seua utilització en la seüent pregunta, en el moment d'inserir diferents documents.

En aquest punt anem a veure les operacions més bàsiques, per a poder treballar sobre exemples pràctics, i així disposar ja d'unes dades inicials per a practicar.

Inserció elemental

La funció **insert** afegirà documents a una col·lecció. En el paràmetre posem el document directament, o una variable que continga el document. Si la col·lecció no existia, la crearà i després afegirà el document. En la següent sentència estem treballant sobre la col·lecció **exemple**, que segurament ja existirà de quan vam la pregunta 3.1 d'instal·lació de MongoDB, que per a provar vam inserir un document. Però si no existia, la crearà sense problemes.

```
> db.exemple.insert({ msg2 : "Com va la cosa?" })
WriteResult({ "nInserted" : 1 })
```

Acabem d'inserir un nou document, i així ens ho avisa ({ "nInserted" : 1 }, s'ha inserit un document). Automàticament haurà creat un element **_id** de tipus **ObjectId**, ja que li fa falta per a identificar el document entre tots els altres de la col·lecció.

I en aquest exemple ens guardem el document en la variable **doc**, i després l'inserim

```
> doc = { msg3 : "Per ací no ens podem queixar ..."}
{ "msg3" : "Per ací no ens podem queixar ..." }
> db.exemple.insert(doc)
WriteResult({ "nInserted" : 1 })
```

També ens indica que ha inserit un document. I haurà creat també el camp **_id** com veurem en el següent punt

Lectura

Tenim dues funcions per a recuperar informació: **find** i **findOne**.

- **find()** : recuperarà tots els documents de la col·lecció, encara que podrem posar criteris per a que ens torne tots els documents que aconsegueixen aquestos criteris (ho veurem més avant).
- **findOne()** : només tornarà un document, en principi el primer. Pot ser sobre tots els documents (i per tant seria el primer document), o posar una condició, i tornaria el primer que complirà la condició.

Exemple de **find()**:

```
> db.exemple.find()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?" }
{ "_id" : ObjectId("56ce31f6c61e04ba81def50c"), "msg2" : "Com va la cosa?" }
{ "_id" : ObjectId("56ce3237c61e04ba81def50d"), "msg3" : "Per ací no ens podem queixar ..." }
```

Exemple de **findOne()**:

```
> db.exemple.findOne()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?" }
```

En tots els casos podem comprovar que és cert el que veníem afirmant, que ha creat automàticament l'element **_id** per a cada document guardat. Evidentment, cadascú de nosaltres tindrà una valors diferents.

Inserció especificant el id

Ara que ja sabem consultar els document de la col·lecció amb **find()** anem a continuar les insercions de documents, per veure les possibilitats que tenim.

En els document que hem inserit fins el moment, no hem especificat el camp **_id**, i Mongo l'ha generat automàticament de tipus **ObjectId**.

Però nosaltres podrem posar aquest camp **_id** amb el valor que vulguem. Això sí, haurem d'estar segurs que aquest valor no l'agafa cap altre document de la col·lecció, o ens donarà un error.

Així per exemple anem a inserir la informació d'uns alumnes. Els posarem en una col·lecció nova anomenada **alumnes**, i els intentarem posar un **_id** personal. Per exemple posarem els números 51, 52, 53, ...

```
> db.alumnes.insert ({_id: 51 , nom: "Rebeca" , cognoms: "Martí Peral"})
WriteResult({ "nInserted" : 1 })
```

Ha anat bé, i si mirem els documents que tenim en la col·lecció, comprovarem que ens ha respectat el **_id**:

```
> db.alumnes.find()
{ "_id" : 51, "nom" : "Rebeca", "cognoms" : "Martí Peral" }
>
```

Però si intentem inserir un altre document amb el mateix `_id` (51), ens donarà error:

```
> db.alumnes.insert ({_id: 51 , nom: "Raquel" , cognoms: "Gomis Arnau"})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: test.alumnes index: _id_ dup key: { :
51.0 }"
  }
})
>
```

Ens avisa que estem duplicant la *clau principal*, és a dir l'identificador.

Inserció múltiple

Quan els documents que volem inserir són senzills, podem inserir més d'un a la vegada, posant dis del `insert()` un **array** amb tots els elements. En el següent exemple creem uns quants nombres primers en la col·lecció del mateix nom:

```
> db.nombresprimers.insert( [ {_id:2} , {_id:3} , {_id:5} , {_id:7} , {_id:11} , {_id:13} ,
{ id:17} , { id:19} ] )
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 8,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```

Ens avisa que ha fet 8 insercions, i ací els tenim:

```
> db.nombresprimers.find()
{ "_id" : 2 }
{ "_id" : 3 }
{ "_id" : 5 }
{ "_id" : 7 }
{ "_id" : 11 }
{ "_id" : 13 }
{ "_id" : 17 }
{ "_id" : 19 }
>
```

Esborrat

Per a esborrar un document d 'una col·lecció utilitzarem la funció **remove**, passant-li com a paràmetre la condició del document o documents a esborrar.

```
> db.nombresprimers.remove( {"_id" : 19} )
WriteResult({ "nRemoved" : 1 })
>
```

Ens avisa que ha esborrat un document.

La condició no cal que siga sobre el camp `_id`. Pot ser sobre qualsevol camp, i esborrarà tots els que coincideixen.

```
> db.exemple.remove( {"msg3" : "Per ací no ens podem queixar ..."} )
WriteResult({ "nRemoved" : 1 })
>
```

També tenim la possibilitat d'esborrar tota una col·lecció amb la funció **drop()**. Pareu atenció perquè és molt senzilla d'eliminar, i per tant, potencialment molt perillosa.

```
> db.nombresprimers.drop()
true
>
```

Actualització

La funció **update** servirà per a actualitzar un document ja guardat. Tindrà dos paràmetres:

- El primer paràmetre serà la condició per a trobar el document que es va a actualitzar.
- El segon paràmetre serà el nou document que substituirà l'anterior

Per exemple, si mirem les dades actuals:

```
> db.exemple.find()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?" }
{ "_id" : ObjectId("56ce31f6c61e04ba81def50c"), "msg2" : "Com va la cosa?" }
```

Podem comprovar el contingut del segon document, el que té **msg2**. Anem a modificar-lo: en el primer paràmetre posem condició de búsqueda (només hi haurà un) i en el segon posem el nou document que substituirà l'anterior

```
> db.exemple.update( {msg2:"Com va la cosa?"} , {msg2:"Què? Com va la cosa?"} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Observeu que la contestació del **update()** és que ha fet **match** (hi ha hagut coincidència) amb un document, i que ha modificat un. Si no en troba cap, no donarà error, senzillament dirà que ha fet match amb 0 documents, i que ha modificat 0 documents. Mirem com efectivament ha canviat el segon document

```
> db.exemple.find()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?" }
{ "_id" : ObjectId("56ce31f6c61e04ba81def50c"), "msg2" : "Què? Com va la cosa?" }
```

Ens vindran molt bé les variables per a les actualitzacions, ja que en moltes ocasions serà modificar lleugerament el document, canviant o afegint algun element. Ho podem fer còmodament amb la variable: primer guardem el document a modificar en una variable; després modifiquem la variable; i per últim fem l'operació d'actualització. Evidentment si tenim alguna variable amb el contingut del document ens podríem estalviar el primer pas.

```
> doc1 = db.exemple.findOne()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?" }
```

```
> doc1.titol = "Missatge 1"
Missatge 1
```

```
> db.exemple.update( {msg:"Hola, què tal?"} , doc1)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.exemple.findOne()
{
  "_id" : ObjectId("56ce310bc61e04ba81def50b"),
  "msg" : "Hola, què tal?",
  "titol" : "Missatge 1"
}
>
```

Funció Save

Hem vist la manera d'inserir nous documents amb **insert()** i d'actualitzar documents existents amb **update()**. Però si intentem inserir un document ja existent (la manera de saber-ho és pel camp **_id**) ens donarà error. I si intentem actualitzar un document no existent, no donarà error, però no actualitzarà res.

La funció **save()** és una barreja dels dos: si el document que anem a salvar ja existeix, doncs el modificarà, i si no existeix el crearà.

Provem-ho amb un exemple, i aprofitem-nos que en la variable **doc1** tenim el contingut d'un document. Fem una modificació, per exemple afegint el camp destinatari:

```
> doc1.destinatari = "Ferran"
Ferran
```

```
> doc1
{
  "_id" : ObjectId("56ce310bc61e04ba81def50b"),
  "msg" : "Hola, què tal?",
  "titol" : "Missatge 1",
  "destinatari" : "Ferran"
}
```

Anem a guardar-lo amb **save()**:

```
> db.exemple.save(doc1)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Ja ens avisa que com ha trobat el document ("**nMatched : 1**") ha modificat un. Efectivament, mirem com ha modificat el document existent:

```
> db.exemple.find()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?", "titol" : "Missatge 1", "destinatari" : "Ferran" }
{ "_id" : ObjectId("56ce31f6c61e04ba81def50c"), "msg2" : "Què? Com va la cosa?" }
```

Anem a fer ara una modificació del camp **_id**. A tots els efectes serà un document nou, ja que la manera d'identificar un document és per aquest camp:

```
> doc1._id=100
100
```

```
> doc1
{
```

```
{
  "_id" : 100,
  "msg" : "Hola, què tal?",
  "titol" : "Missatge 1",
  "destinatari" : "Ferran"
}
```

I ara anem a fer el **save()** d'aquest document:

```
> db.exemple.save(doc1)
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 100 })
```

Observeu com ara avisa que no n'ha trobat cap igual, i el que fa és inserir-lo (ens diu **nUpserted**, que és una barreja de **Updated** i **Inserted**; més avant tornarem a aquesta paraula)

```
> db.exemple.find()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?", "titol" : "Missatge 1",
  "destinatari" : "Ferran" }
{ "_id" : ObjectId("56ce31f6c61e04ba81def50c"), "msg2" : "Què? Com va la cosa?" }
{ "_id" : 100, "msg" : "Hola, què tal?", "titol" : "Missatge 1", "destinatari" : "Ferran" }
```

Efectivament s'ha guardat com un nou document

3.2.3 - Operacions d'actualització avançada

Al final de la pregunta anterior vam veure l'actualització de documents ja existents a la Base de Dades. Aquesta actualització la fèiem modificant tot el document, encara que teníem la variant de guardar el document en una variable, modificar aquesta variable i després fer l'actualització amb aquesta variable. Però observeu que continua sent una modificació de tot el document, una substitució del document antic per un document nou.

En aquesta pregunta veurem la utilització d'uns modificadors (*modifiers*) de l'operació **update()**, que ens permetran modificar documents de forma potent: creant i eliminant claus (elements) d'un document, o canviant-los, i fins i tot afegir o eliminar elements d'un array.

El modificador **\$set** assigna un valor a un camp del document seleccionat de la Base de Dades. Si el camp ja existia, modificarà el valor, i si no existia el crearà.

La sintaxi del modificador **\$set** és la següent:

```
{ $set : { clau : valor } }
```

Però recordeu que és un modificador, i l'hem d'utilitzar dins d'una operació d'actualització. Anirà en el segon paràmetre del **update()**, i per tant amb aquestos modificadors ja no posem tot el document en el segon paràmetre, sinó únicament l'operador de modificació.

Mirem-ho millor en un exemple:

```
> db.alumnes.insert( {nom:"Abel", cognoms:"Bernat Carrera"} )
WriteResult({ "nInserted" : 1 })
>
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera"
}
>
```

Suposem ara que li volem afegir l'edat. Abans ho faríem guardant el document en una variable, i afegint el camp, per a guardar després. Ara ho tenim més fàcil:

```
> db.alumnes.update( {nom:"Abel"} , { $set: {edat:21} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Ha trobat un, i l'ha modificat. Evidentment, si hi haguera més d'un alumne a mb el nom Abel, els modificaria tots.

```
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 21
}
```

Es pot especificar més d'un camp amb els valor corresponents. Si no existien es crearan, i si ja existien es modificaran:

```
> db.alumnes.update( {nom:"Abel"} , { $set: {nota: 8.5 , edat:22} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : 8.5
}
```

I fins i tot es pot canviar el tipus d'un camp determinat, i utilitzar arrays, i objectes, ...

```
> db.alumnes.update( {nom:"Abel"} , { $set: {nota: [8.5,7.5,9] , adreça:
{carrer:"Major",numero:7,cp:"12001"} } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : [
    8.5,
    7.5,
    9
  ],
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12001"
  }
}
```

Podem fins i tot modificar ara només el valor d'un camp d'un objecte del document. Per exemple, anem a modificar el codi postal de l'anterior alumne. La manera d'arribar al codi postal serà **adreça.cp**, però haurem d'anar amb compte que vaja entre cometes per a que el trobe:

```
> db.alumnes.update( {nom:"Abel"} , { $set: {adreça.cp:"12502"} } )
2016-03-08T13:19:59.744+0100 E QUERY [thread1] SyntaxError: missing : after property id
@ (shell):1:49
```

```
> db.alumnes.update( {nom:"Abel"} , { $set: {"adreça.cp":"12502"} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : [
    8.5,
    7.5,
    9
  ],
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  }
}
```


El modificador **\$unset** servirà per a **eliminar** elements (camps) d'un o uns documents. Si el camp existia, l'eliminarà, i si no existia, no donarà error (avisarà que s'han modificat 0 documents).

La sintaxi és:

```
{ $unset : {camp : 1 } }
```

Haurem de posar un valor al camp que anem a esborrar per a mantenir la sintaxi correcta, i posem 1 que equival a true. També podríem posar -1, que equival a false, però aleshores no l'esborraria, i per tant no faríem res. Sempre posarem 1.

Mirem el següent exemple. Afegim un camp, que serà el número d'ordre, i després el llevarem.

```
> db.alumnes.update( {nom:"Abel"} , { $set: {num_ordre:10} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : [
    8.5,
    7.5,
    9
  ],
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "num_ordre" : 10
}

> db.alumnes.update( {nom:"Abel"} , { $unset: {num_ordre:1} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.update( {nom:"Abel"} , { $unset: {puntuacio:1} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

Hem afegit primer el camp **num_ordre**, i hem mostrat el document per comprovar que existeix. Després esborrem el camp **num_ordre** (i ens confirma que ha modificat un document). Després intentem esborrar un camp que no existeix, **puntuacio**. No dóna error, però ens avisa que ha modificat 0 documents. Podem comprovar al final com el document ha quedat com esperàvem.

```
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : [
    8.5,
    7.5,
    9
  ],
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  }
}
```

El modificador **\$rename** canviarà el nom d'un camp. Si no existia, no donarà error i senzillament no el modificarà. Hem de cuidar de posar el nou nom del camp entre cometes, per a que no done error.

La sintaxi és:

```
{ $rename : { camp1 : "nou_nom1" , camp2 : "nou_nom2" , ... } }
```

Per exemple, canviem el nom del camp **nota** a **notes**:

```
> db.alumnes.update( {nom:"Abel"} , { $rename: {nota:"notes"} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "notes" : [
    8.5,
    7.5,
    9
  ]
}
```

Observeu que l'ha canviat de lloc, cosa que ens fa pensar que en canviar de nom un camp, el que fa és tornar a crear-lo amb el nou nom, i esborrar el camp antic.

En aquest exemple tornem a canviar el nom a **nota**, i intentem canviar el nom a un camp inexistent, **camp1**. No donarà error.

```
> db.alumnes.update( {nom:"Abel"} , { $rename: {camp1: "camp2" , notes:"nota"} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    8.5,
    7.5,
    9
  ]
}
```

Com cabria esperar, el modificador **\$inc** servirà per a incrementar un camp numèric. Si el camp existia, l'incrementarà en la quantitat indicada. Si no existia, crearà el camp amb un valor inicial de 0, i incrementarà el valor amb la quantitat indicada. La quantitat pot ser positiva, negativa o fins i tot amb part fraccionària. Sempre funcionarà bé, excepte quan el camp a incrementar no siga numèric, que donarà error.

La sintaxi és aquesta:

```
{ $inc : {camp : quantitat } }
```

En els següents exemples, incrementem un camp nou (per tant el crearà amb el valor especificat), i després l'incrementem en quantitats positives, negatives i fraccionàries, concretament l'inicialitzem amb un **2**, i després l'incrementem en **5**, en **-4** i en **2.25**, per tant el resultat final serà **5.25**:

```
> db.alumnes.update( {nom:"Abel"} , { $inc: {puntuacio:2} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : [
    8.5,
    7.5,
    9
  ],
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "puntuacio" : 2
}
> db.alumnes.update( {nom:"Abel"} , { $inc: {puntuacio:5} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.update( {nom:"Abel"} , { $inc: {puntuacio:-4} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.update( {nom:"Abel"} , { $inc: {puntuacio:2.25} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnes.findOne()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "nota" : [
    8.5,
    7.5,
    9
  ],
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "puntuacio" : 5.25
}
```

3.2.3.5 - Elements d'un array

Per a accedir directament a un element d'un array d'un determinat document es pot utilitzar la següent sintaxi:

```
"array.index"
```

Hem de tenir present que el primer element de l'array és el de subíndex 0. I no us oblideu de tancar-ho tot entre comentes per a que ho puga trobar.

Si no existeix l'element amb el subíndex indicat, donarà error.

Per exemple, anem a pujar un punt la primera nota de l'alumne que estem utilitzant en tots els exemples :

```
>db.alumnes.update( {nom:"Abel"} , { $inc : { "nota.0" : 1 } } )
```

```
> db.alumnes.findOne()  
{  
  "_id" : ObjectId("56df11d778549bdfbf2125e3"),  
  "nom" : "Abel",  
  "cognoms" : "Bernat Carrera",  
  "edat" : 22,  
  "adreça" : {  
    "carrer" : "Major",  
    "numero" : 7,  
    "cp" : "12502"  
  },  
  "nota" : [  
    9.5,  
    7.5,  
    9  
  ]  
}
```

3.2.3.6 - Inserció en Arrays: \$push

La manera més senzilla d'introduir un element en un array és utilitzar **\$push** sense més. Si existia l'array, introduirà el o els nous elements al final. Si no existia l'array, el crearà amb aquest o aquests elements.

La sintaxi és:

```
{ $push : { clau : element } }
```

Per exemple anem a afegir una nota a l'alumne de sempre, i posem-la diferent per veure que s'introdueix al final:

```
> db.alumnes.update( {nom:"Abel"} , { $push : { nota : 7 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.alumnes.findOne()
{
  "_id" : ObjectId("56df11d778549bdfbf2125e3"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    9.5,
    7.5,
    9,
    7
  ]
}
```

Tambe hi ha manera d'introduir un element en una determinada posició que no siga al final, però es complica prou la cosa, ja que hem d'utilitzar per una banda el modificador **\$position** per a dir on s'ha d'inserir, i per una altra banda el modificador **\$each** per a poder especificar el o els valors que es volen inserir. Es posa a continuació únicament de forma il·lustrativa.

Per a inserir en una determinada posició hem d'utilitzar obligatòriament 2 modificadors més:

- **\$position** indicarà a partir de quina posició es farà l'acció (normalment d'inserir en l'array, és a dir, **\$push**)
- **\$each** ens permet especificar una sèrie de valors com un array, i vol dir que es farà l'operació per a cada valor de l'array

Els dos modificadors seguiran la sintaxi de sempre, de clau valor, per tant el conjunt de la sintaxi és:

```
{ $ push :
  { clau_del_array :
    { $position : posició ,
      $each : [ valors ]
    }
  }
}
```

Ací tenim un exemple on introduïm una nota en la primera posició:

```
> db.alumnes.update( {nom:"Abel"} , { $push : { nota : { $position : 0 , $each : [5] } } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.alumnes.findOne()
{
  "_id" : ObjectId("56df11d778549bdfbf2125e3"),
  "nom" : "Abel",
  "cognoms" : "Bernat Carrera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    5,
    9.5,
    7.5,
    9,
    7
  ]
}
```

3.2.3.7 - Eliminació en arrays: \$pop i \$pull

Hi ha més d'una manera d'eliminar elements d'un array.

\$pop

Si volem eliminar el primer element o l'últim, el modificador adequat és **\$pop**. La sintaxi és

```
{ $pop : { clau : posicio } }
```

On en posició podem posar:

- -1 , i esborrarà el primer element
- 1 , i esborrarà l'últim

En els següents exemples s'esborren primer l'últim element i després el primer.

```
> db.alumnes.findOne()  
{  
  "_id" : ObjectId("56df11d778549bdfbf2125e3"),  
  "nom" : "Abel",  
  "cognoms" : "Bernat Carrera",  
  "edat" : 22,  
  "adreça" : {  
    "carrer" : "Major",  
    "numero" : 7,  
    "cp" : "12502"  
  },  
  "nota" : [  
    5,  
    9.5,  
    7.5,  
    9,  
    7  
  ]  
}
```

```
> db.alumnes.update( {nom:"Abel"} , { $pop : { nota : 1 } } )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.alumnes.findOne()  
{  
  "_id" : ObjectId("56df11d778549bdfbf2125e3"),  
  "nom" : "Abel",  
  "cognoms" : "Bernat Carrera",  
  "edat" : 22,  
  "adreça" : {  
    "carrer" : "Major",  
    "numero" : 7,  
    "cp" : "12502"  
  },  
  "nota" : [  
    5,  
    9.5,  
    7.5,  
    9  
  ]  
}
```

```
> db.alumnes.update( {nom:"Abel"} , { $pop : { nota : -1 } } )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.alumnes.findOne()  
{  
  "_id" : ObjectId("56df11d778549bdfbf2125e3"),  
  "nom" : "Abel",  
  "cognoms" : "Bernat Carrera",  
  "edat" : 22,  
  "adreça" : {  
    "carrer" : "Major",  
    "numero" : 7,  
    "cp" : "12502"  
  },  
  "nota" : [  
    9.5,  
    7.5,  
    9  
  ]  
}
```

\$pull

Amb aquest modificador esborrarem els elements de l'array que coincideixen amb una condició, estiguen en la posició que estiguem. Observeu com es pot eliminar més d'un element.

Per a poder comprovar-lo bé, primer inserim un altre element al final de l'array, amb el valor **7.5** (si heu seguit els mateixos exemples que en

aquestos apunts, aquest valor ja es troba en la segona posició).

```
> db.alumnes.update( {nom:"Abel"} , { $push : { nota : 7.5 } } )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.alumnes.findOne()  
{  
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),  
  "nom" : "Abel",  
  "cognoms" : "Bernat Carrera",  
  "edat" : 22,  
  "adreça" : {  
    "carrer" : "Major",  
    "numero" : 7,  
    "cp" : "12502"  
  },  
  "nota" : [  
    9.5,  
    7.5,  
    9,  
    7.5  
  ]  
}
```

Ara anem a esborrar amb **\$pull** l'element de valor 7.5

```
> db.alumnes.update( {nom:"Abel"} , { $pull : { nota : 7.5 } } )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.alumnes.findOne()  
{  
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),  
  "nom" : "Abel",  
  "cognoms" : "Bernat Carrera",  
  "edat" : 22,  
  "adreça" : {  
    "carrer" : "Major",  
    "numero" : 7,  
    "cp" : "12502"  
  },  
  "nota" : [  
    9.5,  
    9  
  ]  
}
```

Aquesta paraula ja l'havíem comentada en un punt anterior.

En el **update()** normal, si la condició de búsqueda no donava cap resultat (parlant ràpid, si no feis *matching* amb cap document), doncs no actualitzava cap document i punt.

El **Upsert** és una variant de l'update, que quan no coincideixa cap document amb la condició, crearà un document nou que serà el resultat de combinar el criteri que s'ha utilitzat en la condició amb les operacions d'actualització fetes en el segon paràmetre

Per a que un **Update** actue d'aquesta manera, li hem de posar un tercer paràmetre amb el valor **true**:

```
update ( { ... } , { ... } , true )
```

Recordeu que el primer paràmetre era la condició, i el segon l'actualització.

Mirem-ho en l'exemple dels alumnes. Si anem a actualitzar els cognoms, i es troba el document, s'actualitzarà:

```
> db.alumnes.update( { nom:"Abel" } , { $set : { cognoms : "Bernat Cantera" } } , true )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Efectivament, ens diu que ha modificat un document.

Però si no es troba el document (per exemple perquè li hem posat el nom **Berta**):

```
> db.alumnes.update( { nom:"Berta" } , { $set : { cognoms : "Bernat Cantero" } } , true )
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("56dfdbd136d8b095cb6bd57a")
})
```

Ja ens avisa que no ha fet cap *matching*, i ha fet un **Upsert**. Ho podem comprovar mirant tots els document de la col·lecció:

```
> db.alumnes.find()
{ "_id" : ObjectId("56debe3017bf4ed437dc77c8"), "nom" : "Abel", "cognoms" : "Bernat Cantera",
"edat" : 22, "adreça" : { "carrer" : "Major", "numero" : 7, "cp" : "12502" }, "nota" : [ 9.5, 9 ]
}
{ "_id" : ObjectId("56dfdbd136d8b095cb6bd57a"), "nom" : "Berta", "cognoms" : "Bernat Cantero" }
>
```

El nou document tindrà els camps:

- **_id** , amb el que ens havia avisat que generaria
- Els camps de la condició, que en el nostre exemple és **{ nom:"Berta" }**
- Els camps de l'actualització, que en el nostre exemple eren els cognoms

En la pregunta anterior hem vist com introduir, eliminar i modificar documents. Les consultes de documents han segut molt senzilles, per a comprovar únicament els resultats.

En aquesta pregunta veurem en profunditat la consulta de documents.

- Funcions **find()** i **findOne()**, que són les que hem utilitzat fins ara. Veurem en profunditat la seua sintaxi i potència.
- Limitarem i ordenarem també els resultats
- Fins i tot podrem elaborar més els resultats, agrupant els resultats, utilitzant funcions d'agregació (o millor dir operadors d'agregació) i donant-los un aspecte diferent

3.3.1 - Paràmetres de les funcions find() i findOne()

Les funcions **find()** i **findOne()** són absolutament equivalents, amb l'única diferència que la primera torna tots els documents trobats, mentre que la segona només torna el primer document trobat.

Per una millor comprensió, utilitzarem únicament **find()**, per veure tots els resultats obtinguts.

La funció **find()** s'ha comparat tradicionalment amb la sentència SELECT de SQL. Sempre tornarà un conjunt de documents, que poden variar des de no tornar cap document, a tornar-los tots els de la col·lecció.

La funció **find()** pot tenir uns quants paràmetres.

- El primer indica una condició o criteri, i tornarà aquells documents de la col·lecció que acompleixen la condició o criteri. Aquesta condició ve donada en forma de document (o objecte) JSON, i és com l'havíem vist en la funció **update()**:

```
db.col_leccio1.find( { clau1 : valor1 } )
```

Tornarà tots els documents de la col·lecció **col_leccio1** que tinguen el camp **clau1** i que en ella tinguen el valor **valor1**. Aquest criteri pot ser el complicat que faça falta, formant-lo en JSON. En definitiva, tornarà aquells documents que facen *matching* amb el document del criteri, és a dir, funcionaria com un **and**

```
db.col_leccio1.find( { clau1 : valor1 , clau2 : valor2 } )
```

que tornaria aquells documents de la **col_leccio1** que tenen el camp **clau1** amb el valor **valor1** i que tenen el camp **clau2** amb el valor **valor2**

Si no volem posar cap criteri, per a que els torne tots, no posem res com a paràmetre, o encara millor, li passem un document (objecte) buit, de manera que tots els documents de la col·lecció faran *matching* amb ell.

```
db.col_leccio1.find( { } )
```

Tindrem açò present, sobretot quan ens toque utilitzar el segon paràmetre de **find**. Si no volem cap criteri, posarem el document buit com l'exemple anterior

- El segon paràmetre ens servirà per a delimitar els camps dels documents que es tornaran. També tindrà el format JSON d'un objecte al qual li posarem com a claus els diferents camps que volem que apareguen o no, i com a valor 1 per a que sí que apareguen i 0 per a que no apareguen.

Si posem algun camp a que sí que aparega (és a dir, amb el valor 1), els únics que apareixeran seran aquestos, a més del **_id** que per defecte sempre apareix.

```
> db.alumnes.find({}, {nom:1})
{ "_id" : ObjectId("56debe3017bf4ed437dc77c8"), "nom" : "Abel" }
{ "_id" : ObjectId("56dfdbd136d8b095cb6bd57a"), "nom" : "Berta" }
```

Per tant si no volem que aparega **_id** posarem:

```
> db.alumnes.find({}, {_id:0})
{ "nom" : "Abel", "cognoms" : "Bernat Cantera", "edat" : 22, "adreça" : { "carrer" : "Major",
"numero" : 7, "cp" : "12502" }, "nota" : [ 9.5, 9 ] }
{ "nom" : "Berta", "cognoms" : "Bernat Cantero" }
```

I si volem traure únicament el nom:

```
> db.alumnes.find({}, {nom:1, _id:0})
{ "nom" : "Abel" }
{ "nom" : "Berta" }
```

Per últim, com que a partir d'ara utilitzarem documents més complicats, si volem que ens apareguen els camps que retornem d'una forma un poc més elegant o bonica (*pretty*), posarem aquesta funció al final: **find().pretty()**

```
> db.alumnes.find().pretty()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Cantera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    9.5,
    9
  ]
}
{
  "_id" : ObjectId("56dfdbd136d8b095cb6bd57a"),
```

```
"nom" : "Berta",  
"cognoms" : "Bernat Cantero"  
}
```

3.3.2 - Operadors de les condicions

Abans de començar aquesta pregunta, anem a agafar unes dades de prova, que estan en el fitxer `libros_ejemplo.json`

Només heu de copiar el contingut del fitxer en la terminal del client de Mongo.

Posem ací el contingut per a que pugueu pegar-li una miradeta sense necessitat d'obrir-lo. Anirà bé per als exemples posteriors.

```
db.libro.save({
  "_id": "9788408117117",
  "titulo": "Circo Máximo",
  "autor": "Santiago Posteguillo",
  "editorial": "Planeta",
  "enstock": true,
  "paginas": 1100,
  "precio": 21.75,
  "fecha": new ISODate("2013-08-29T00:00:00Z"),
  "resumen": "Circo Máximo, de Santiago Posteguillo, que ha escrito otras obras de narrativa histórica c
})

db.libro.save({
  "_id": "9788401342158",
  "titulo": "El juego de Ripper",
  "autor": "Isabel Allende",
  "editorial": "Plaza & Janes",
  "enstock": true,
  "paginas": 480,
  "precio": 21.75,
  "fecha": new ISODate("2014-03-01T00:00:00Z"),
  "resumen": "Tal como predijo la astróloga más reputada de San Francisco, una oleada de crímenes comier
})

db.libro.save({
  "_id": "9788496208919",
  "titulo": "Juego de tronos: Canción de hielo y fuego 1",
  "autor": "George R.R. Martin",
  "editorial": "Gigamesh",
  "enstock": true,
  "paginas": 793,
  "precio": 9.5,
  "fecha": new ISODate("2011-11-24T00:00:00Z"),
  "resumen": "Tras el largo verano, el invierno se acerca a los Siete Reinos. Lord Eddars Stark, señor de I
})

db.libro.save({
  "_id": "9788499088075",
  "titulo": "La ladrona de libros",
  "autor": "Markus Zusak",
  "editorial": "Debolsillo",
  "enstock": false,
  "paginas": 544,
  "precio": 9.45,
  "fecha": new ISODate("2009-01-09T00:00:00Z"),
  "resumen": "En plena II Guerra Mundial, la pequeña Liesel hallará su salvación en la lectura. Una novela p
})

db.libro.save({
  "_id": "9788415140054",
  "titulo": "La princesa de hielo",
  "autor": "Camilla Lackberg",
  "editorial": "Embolsillo",
  "enstock": true,
  "precio": 11,
  "fecha": new ISODate("2012-10-30T00:00:00Z"),
  "resumen": "Misterio y secretos familiares en una emocionante novela de suspense Erica vuelve a su pueblo
})

db.libro.save({
  "_id": "9788408113331",
  "titulo": "Las carreras de Escorpio",
  "autor": "Maggie Stiefvater",
  "editorial": "Planeta",
  "enstock": false,
  "paginas": 290,
  "precio": 17.23,
  "fecha": new ISODate("2013-06-04T00:00:00Z"),
  "resumen": "En la pequeña isla de Thisby, cada noviembre los caballos de agua de la mitología celta emerge
})

db.libro.save({
  "_id": "9788468738895",
  "titulo": "Las reglas del juego",
  "autor": "Anna Casanovas",
  "enstock": true,
  "paginas": null,
  "precio": 15.90,
  "fecha": new ISODate("2014-02-06T00:00:00Z"),
  "resumen": "Susana Lobato tiene la vida perfectamente planeada y está a punto de conseguir todo lo que qui
})
```

Com podeu comprovar estan els comandos d'inserció (**save()**) i també es veu prou bé els camps de cada document.

Podeu comprovar que hi ha 7 documents en la nova col·lecció **libro**:

```
> db.libro.count()
7
```

I també podem consultar els títols de forma còmoda:

```
> db.libro.find( {}, {titulo:1} )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo" }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1" }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros" }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

I un altre exemple, on consultem els llibres que estan en stock (hi ha un camp booleà que ho diu: **enstock**), mostrant títol, editorial i preu

```
> db.libro.find( {enstock: true} , {titulo:1 , editorial:1 , precio:1} )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes",
"precio" : 21.75 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial"
: "Gigamesh", "precio" : 9.5 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo",
"precio" : 11 }
```

I un últim exemple, on consultem els llibres que estan en stock i tenen un preu de 21.75 €, mostrant tot excepte el `_id` i el resum

```
> db.libro.find( {enstock: true , precio: 21.75} , {titulo:1 , editorial:1 , precio:1} )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes",
"precio" : 21.75 }
```

Anem a mirar ara operadors que ens serviran per fer millor les consultes.

Operadors de comparació

Fins ara en totes les condicions hem utilitzat la igualtat, si un determinat camp era igual a un determinat valor. Però hi ha infinitat de consultes en les quals voldrem altres operacions de comparació: major, major o igual, menor, ...

Aquestos són els operadors de comparació:

- **\$lt** (*less than*) **menor**
- **\$lte** (*less than or equal*) **menor o igual**
- **\$gt** (*greater than*) **major**
- **\$gte** (*greater than or equal*) **major o igual**
- **\$ne** (*not equal*) **distint**
- **\$eq** (*equal*) **igual** (però aquest quasi que no caldria, perquè en no posar res es refereix a la igualtat com fins ara)

La sintaxi per a la seua utilització és, com sempre, acoplar-se a la sintaxi JSON:

```
clau : { $operador : valor [, ... ] }
```

Així per exemple, per a buscar els llibres de més de 10 €:

```
> db.libro.find( { precio : { $gt : 10 } } , { titulo:1 , precio:1 } )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "precio" : 21.75 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "precio" : 11 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "precio" : 17.23 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9 }
```

I per a buscar els llibres entre 10 i 20 €:

```
> db.libro.find( { precio : { $gt : 10 , $lt:20 } } , { titulo:1 , precio:1 } )
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "precio" : 11 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "precio" : 17.23 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9 }
```

És especialment útil per a les dates, ja que difícilment trobarem una data (i hora) exacta, i voldrem quasi sempre els documents anteriors a una data, o posteriors, o entre dues dates. Haurem d'anar amb compte pel tractament especial de les dates: hem de comparar coses del mateix tipus, i per tant la data amb la qual volem comparar l'haurem de tenir en forma de data:

```
> var d = new ISODate("2013-01-01T00:00:00Z")
> db.libro.find( {fecha:{ $gte:d } } , {fecha:1} )
{ "_id" : "9788408117117", "fecha" : ISODate("2013-08-29T00:00:00Z") }
{ "_id" : "9788401342158", "fecha" : ISODate("2014-03-01T00:00:00Z") }
{ "_id" : "9788408113331", "fecha" : ISODate("2013-06-04T00:00:00Z") }
{ "_id" : "9788468738895", "fecha" : ISODate("2014-02-06T00:00:00Z") }
```

\$in

Servirà per a comprovar si el valor d'un camp està entre els d'una llista, proporcionada com un array. La sintaxi és:

```
clau : { $in : [valor1 , valor2 , ... , valorN] }
```

I ací tenim un exemple, els llibre de les editorials Planeta i Debolsillo:

```
> db.libro.find( { editorial: { $in : ["Planeta" , "Debolsillo"] } } , { titulo:1 , editorial:1 } )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta" }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta" }
```

\$nin

És el contrari, traura els que no estan en la llista.

```
> db.libro.find( { editorial: { $nin : ["Planeta" , "Debolsillo"] } } , { titulo:1 , editorial:1 } )
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial" : "Gigamesh" }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

Observeu com també trau els llibres que no tenen editorial, com és el cas de l'últim llibre, Las reglas del juego

\$or

L'operador anterior, \$in, ja feia una espècie de OR, però sempre sobre el mateix camp. Si l'operació OR la volem fer sobre camps distints, haurem d'utilitzar l'operador \$or. La seua sintaxi ha de jugar amb la possibilitat de posar molts elements, i per tant convé l'array:

\$or : [{clau1:valor1} , {clau2:valor2} , ... , {clauN:valorN}]

Serà cert si s'acompleix alguna de les condicions. Per exemple, traure els llibres que no estan en stock o que no tenen editorial:

```
> db.libro.find( { $or : [ {enstock:false} , {editorial:null} ] } , { titulo:1 , enstock:1 , editorial:1 } )
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo", "enstock" : false }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta", "enstock" : false }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "enstock" : true }
```

\$not

Serveix per a negar una altra condició.

```
$not : { condició }
```

Per exemple els llibres que no són de l'editorial Planeta (observeu que seria més senzill utilitzar l'operador \$ne, però és per a mostrar el seu funcionament:

```
> db.libro.find( { editorial: { $not : { $eq:"Planeta" } } } , { titulo:1 , editorial:1 } )
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial" : "Gigamesh" }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo" }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

\$exists

Servirà per a saber els documents que tenen un determinat camp

```
clau : { $exists : boolean }
```

Depenent del valor *boolean*, el funcionament serà:

- **true**: torna els documents en els quals existeix el camp, encara que el seu valor siga nul
- **false**: torna els documents que no tenen el camp.

Anem a traure els llibres que tenen el camp **paginas**:

```
> db.libro.find( { paginas: { $exists:true } } , { titulo:1 , paginas:1 } )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "paginas" : 1100 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "paginas" : 480 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "paginas" :
```

```
793 }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "paginas" : 544 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "paginas" : 290 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "paginas" : null }
```

Observeu com ens apareix també l'últim llibre, que té el camp **paginas** amb el valor **nul**. En canvi si haguérem fet la consulta preguntant pels que són diferent de nul, no apareixeria aquest últim llibre:

```
> db.libro.find( { paginas: {$ne:null} } , {titulo:1 , paginas:1} )
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "paginas" : 1100 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "paginas" : 480 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "paginas" : 793 }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "paginas" : 544 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "paginas" : 290 }
```

I si posem **false** al valor en el **\$exists**, únicament ens apareixerà el llibre que no té el camp:

```
> db.libro.find( { paginas: {$exists:false} } , {titulo:1 , paginas:1} )
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
```

I per la mateixa raó que abans, si traiem els que tenen **paginas** a nul, ens eixirà tant qui no té el camp, com qui el té però amb valor nul:

```
> db.libro.find( { paginas: null } , {titulo:1 , paginas:1} )
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "paginas" : null }
```

Per tant, per a segons quines coses, ens interessa l'operador **\$exists**, en compte de jugar amb el nul.

Expressions regulars

Mongo accepta les expressions regulars de forma nativa, cosa que dóna molta potència per a poder buscar informació diversa.

Les expressions regulars en Mongo tenen la mateixa sintaxi que en Perl, i que és molt molt pareguda a la major part de llenguatges de programació.

Mirem alguns exemples. Els llibres dins dels quals està la paraula **juego**:

```
> db.libro.find( { titulo: /juego/ } , {titulo:1} )
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

Ara que tenen la paraula **juego** sense importar majúscules o minúscules:

```
> db.libro.find( { titulo: /juego/i } , {titulo:1} )
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

I ara que tenen la paraula **juego** només al principi.

```
> db.libro.find( { titulo: /^juego/i } , {titulo:1} )
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1" }
```

I ara els llibres que en el resum (**resumen**) que tenen la paraula **amiga** o **amigo**, és a dir **amig** seguit d'una **a** o una **o**:

```
> db.libro.find( { resumen: /amig[ao]/i } , {titulo:1} )
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

Arrays

Les consultes dins d'arrays de Mongo són molt senzilles.

La més senzilla és com quan busquem un valor d'un tipus senzill, i en aquest cas el que farà Mongo és buscar en tot l'array per si està aquest valor. És a dir, exactament igual que el que hem fet fins ara.

```
db.col.lecciol.find ( { clau_array : valor } )
```

Mirem-ho en un exemple. Anem a crear dos documents que tinguen un array cadascun, per exemple de colors. El creem en una col·lecció nova, anomenada **colorins**, en dos documents amb el mateix camp de tipus array, **color**, però amb dades diferents:

```
> db.colorins.insert({color: ["roig","blau","groc"]})
WriteResult({ "nInserted" : 1 })
```

```
> db.colorins.insert({color: ["negre","blanc","roig"]})
WriteResult({ "nInserted" : 1 })
```

```
> db.colorins.find();
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
```

Com es veu en la sintaxi, triar els documents que tenen un camp (en aquest cas d'array) que continga un valor, és igual de senzill que quan es tracta d'un camp de tipus string, per exemple:

```
> db.colorins.find({color:"roig"})
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
```

També podem utilitzar qualsevol dels operadors vistos fins el moment, com per exemple l'operador **\$in**, que mirarà els documents que tenen algun dels colors que s'especifica a continuació:

```
> db.colorins.find({color: {$in: ["groc","lila"]}})
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
```

O per exemple també utilitzar **expressions regulars**:

```
> db.colorins.find({color: /bl/ })
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
```

\$all

L'operador **\$all** el podem utilitzar quan vulguem seleccionar els documents que en l'array tiguin **tots** els elements especificats.

Per exemple, anem a buscar els document que tenen el color roig i blau.

```
> db.colorins.find({color: { $all: ["roig","blau"]}})
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
```

Subíndex

Si volem mirar exactament una determinada posició de l'array, podem especificar la posició immediatament després de la clau, **separada per un punt**. Recordeu que la primera posició és la **0**. Hem de posar entre cometes la clau i la posició, sinó no sabrà trobar-la.

Per exemple, busquem els documents que tenen el roig en la primera posició.

```
> db.colorins.find({"color.0" : "roig" })
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
```

Nota

Accedir a una determinada posició és fàcil, però no és tan fàcil accedir a una posició calculada, per exemple a l'última posició. Ja fa falta coneixements un poc més avançats de JavaScript, per a posar dins del **find()** una funció en JavaScript, i actuar dins d'aquesta.

Únicament de manera il·lustrativa, posem ací la manera de traure els documents, l'últim color dels quals és el roig. En ella ens creem una variable amb l'últim element de l'array (amb **pop()**), i el comparem amb el color roig, tornant true en cas de que sí que siguin iguals:

```
> db.colorins.find(function() { var a =this.color.pop(); return (a=="roig") } )
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
```

També hi ha una forma alternativa de fer-ho, que és utilitzant l'operador **\$where**, que ens permet crear condicions amb sintaxi JavaScript:

```
> db.colorins.find({$where:"this.color[this.color.length - 1]=='roig'"})
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
```

\$size

L'operador **\$size** ens servirà per a fer condicions sobre el número d'elements d'un array.

Incorporem 2 documents nous, amb 2 i 4 elements respectivament, per a poder comprovar-lo:

```
> db.colorins.insert({color: ["negre","blanc"]})
WriteResult({ "nInserted" : 1 })
```

```
> db.colorins.insert({color: ["taronja","gris","lila","verd"]})
WriteResult({ "nInserted" : 1 })
```

```
> db.colorins.find()
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
{ "_id" : ObjectId("56e16972aa3c92aaed389da6"), "color" : [ "negre", "blanc" ] }
{ "_id" : ObjectId("56e16990aa3c92aaed389da7"), "color" : [ "taronja", "gris", "lila", "verd" ] }
```

Ara anem a seleccionar els documents que tenen 4 colors


```
> db.colorins.find({color:{size:4}})
{ "_id" : ObjectId("56e16990aa3c92aaed389da7"), "color" : [ "taronja", "gris", "lila", "verd" ] }
```

Nota

L'operador **\$size** només admet un valor numèric, i no es poden concatenar expressions amb altres operadors, com per exemple intentar la condició que la grandària de l'array siga menor o igual a un determinat valor. Es pot tornar a esquivar la qüestió amb l'operador **\$where**, i posar la condició en JavaScript. Així la consulta dels documents que tenen 3 o menys colors la podríem traure d'aquesta manera:

```
> db.colorins.find({$where:"this.color.length<=3"})
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau", "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc", "roig" ] }
{ "_id" : ObjectId("56e16972aa3c92aaed389da6"), "color" : [ "negre", "blanc" ] }
```

\$slice

L'operador **\$slice** no és un operador que es pugui posar en les condicions (criteris), sinó que servirà per a extraure determinats elements de l'array, pel número d'ordre d'aquests elements en e l'array. Només el podrem posar, per tant, en el segon paràmetre del **find()**.

La sintaxi és:

```
clau : {$slice : x }
```

Els valors que pot agafar **x** són:

- Números positius: serà el número d'elements del principi (per l'esquerra)
- Números negatius: serà el número d'elements del final (per la dreta)
- Un array de 2 elements (**[x,y]**): traurà a partir de la posició **x** (0 és el primer), tants elements com indique **y**

Per exemple, anem a traure els dos primers colors de cada document:

```
> db.colorins.find({}, {color:{$slice:2} })
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "roig", "blau" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "negre", "blanc" ] }
{ "_id" : ObjectId("56e16972aa3c92aaed389da6"), "color" : [ "negre", "blanc" ] }
{ "_id" : ObjectId("56e16990aa3c92aaed389da7"), "color" : [ "taronja", "gris" ] }
```

O traure l'últim color:

```
> db.colorins.find({}, {color:{$slice:-1} })
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "roig" ] }
{ "_id" : ObjectId("56e16972aa3c92aaed389da6"), "color" : [ "blanc" ] }
{ "_id" : ObjectId("56e16990aa3c92aaed389da7"), "color" : [ "verd" ] }
```

O traure el tercer element, tinguem els que tinguem. Recordeu que el segon element, és el de la posició 2, i en volem traure 1.

```
> db.colorins.find({}, {color:{$slice:[2,1]} })
{ "_id" : ObjectId("56e1438ff6663c8169030e09"), "color" : [ "groc" ] }
{ "_id" : ObjectId("56e14398f6663c8169030e0a"), "color" : [ "roig" ] }
{ "_id" : ObjectId("56e16972aa3c92aaed389da6"), "color" : [ ] }
{ "_id" : ObjectId("56e16990aa3c92aaed389da7"), "color" : [ "lila" ] }
```

Recerques en objectes

Per a fer recerques en camps que a la seua vegada són objectes (o documents dins de documents, en la terminologia de Mongo), només hem de posar la ruta de les claus separant per mig de punts, i cuidar de posar-les entre cometes.

Així, per exemple, anem a fer una consulta sobre la col·lecció d'alumnes, que eren uns documents en els quals hi havia algun camp de tipus objecte.

```
> db.alumnes.find().pretty()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Cantera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    9.5,
    9
  ]
}
{
  "_id" : ObjectId("56dfdbd136d8b095cb6bd57a"),
```

```
{
  "nom" : "Berta",
  "cognoms" : "Bernat Cantero"
}
```

Es podrien traure els documents (els alumnes) que viuen en el codi postal 12502. Ens ha d'eixir l'únic alumne del qual tenim l'adreça, que justament té aquest codi postal. Recordeu que en la clau (realment clau.subclau), ha d'anar entre cometes. Hem posat al final **pretty()** per a una millor lectura, però evidentment no és necessari.

```
> db.alumnes.find({"adreça.cp": "12502"}).pretty()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Cantera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    9.5,
    9
  ]
}
```

I funcionaria igual amb qualsevol número de subnivells, és a dir, documents que tenen objectes, els quals tenen objectes, ... I també amb altres tipus d'operadors, o expressions regulars, ...

Per exemple, tots els alumnes de Castelló (el codi postal ha de començar per 12 i contenir 3 xifres més, és a dir, caràcter del 0 al 9, i 3 vegades.

```
> db.alumnes.find({"adreça.cp": /^12[0-9]{3}/}).pretty()
{
  "_id" : ObjectId("56debe3017bf4ed437dc77c8"),
  "nom" : "Abel",
  "cognoms" : "Bernat Cantera",
  "edat" : 22,
  "adreça" : {
    "carrer" : "Major",
    "numero" : 7,
    "cp" : "12502"
  },
  "nota" : [
    9.5,
    9
  ]
}
```

Limit, Skip i Sort

Una vegada tenim feta una consulta, podem limitar el nombre de documents que ens ha de tornar, o ordenar-los.

Per a això hi ha uns mètodes que apliquem al final del **find()**, és a dir, a continuació del **find()**, separats per un punt.

Ho aplicarem als llibres, que és on tenim més documents. I no mostrem tots els camps, per a una millor lectura:

```
> db.libro.find({}, {titulo:1, precio:1, editorial:1})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes",
"precio" : 21.75 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial"
: "Gigamesh", "precio" : 9.5 }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo",
"precio" : 9.45 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo",
"precio" : 11 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta",
"precio" : 17.23 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9 }
```

limit(n)

Limita el número de documents tornats a *n* documents.

```
> db.libro.find({}, {titulo:1, precio:1, editorial:1}).limit(3)
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes",
"precio" : 21.75 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial"
: "Gigamesh", "precio" : 9.5 }
```

Si el número de documents que fa la consulta és menor que *n*, doncs se'n tornaran menys. Així per exemple, de l'editorial Planeta només hi ha dos llibres. Encara que posem **limit(3)**, se'n tornaran 2.

```
> db.libro.find({editorial:"Planeta"}, {titulo:1, precio:1, editorial:1}).limit(3)
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta",
"precio" : 17.23 }
```

skip(n)

Se saltaran els primers *n* documents. Si hi haguera menys documents dels que se salten, doncs no se'n mostraria cap.

```
> db.libro.find({}, {titulo:1 , precio:1 , editorial:1}).skip(2)
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial" : "Gigamesh", "precio" : 9.5 }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo", "precio" : 9.45 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo", "precio" : 11 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta", "precio" : 17.23 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9 }
```

sort()

Serveix per a ordenar. Com a paràmetre se li passarà un objecte JSON amb les claus per a ordenar, i els valors seran:

- 1: ordre ascendent
- -1: ordre descendent

Si posem més d'una clau, s'ordenarà pel primer, en cas d'empat pel segon, ...

En aquest exemple ordenem pel preu

```
> db.libro.find({}, {titulo:1 , precio:1 , editorial:1}).sort({precio:1})
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo", "precio" : 9.45 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial" : "Gigamesh", "precio" : 9.5 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo", "precio" : 11 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta", "precio" : 17.23 }
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes", "precio" : 21.75 }
```

I com déiem, es pot posar més d'un camp d'ordenació. Per exemple, per editorial en ordre ascendent, i per preu en ordre descendent

```
> db.libro.find({}, {titulo:1 , precio:1 , editorial:1}).sort({editorial:1 , precio:-1})
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9 }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo", "precio" : 9.45 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo", "precio" : 11 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial" : "Gigamesh", "precio" : 9.5 }
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta", "precio" : 21.75 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta", "precio" : 17.23 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes", "precio" : 21.75 }
```

Observeu com el primer és el que no té editorial (equivalent a null). I com que hi ha dos de l'editorial Planeta, apareix primer el més car, i després el més barat.

I evidentment, es poden combinar els mètodes limit, skip i sort.

En aquest exemple traurem el segon i tercer llibre més car. Per a això ordenem per preu de forma descendent, saltam un i limitem a 2. No importa l'ordre com col·locar skip, limit i sort.

```
> db.libro.find({}, {titulo:1 , precio:1 , editorial:1}).sort({precio:-1}).skip(1).limit(2)
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes", "precio" : 21.75 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta", "precio" : 17.23 }
```

L'agregació ens permetrà fer consultes molt avançades. És un procés un poc complicat però molt potent. Ens donarà una potència quasi com la del SQL quan comencem a utilitzar el GROUP BY i HAVING.

La tècnica que s'utilitza és la del **pipeline**, és a dir fer una s'erie de comandos, cadascun agafa les dades que proporciona l'anterior i a la seua vegada proporciona les dades al següent comando. D'aquesta manera es tractarà un conjunt de documents i es faran "operacions" sobre ells seqüencialment en blocs: filtrat, projecció, agrupacions, ordenació, limitació i *skipping* (saltar alguns).

La sintaxi serà:

```
db.col_lecciol.aggregate ( operador $match , operador $project , operador $group , operador $sort , operador $limit , operador $skip )
```

L'ordre dels operadors pot canviar, però hem de tenir en compte que els comandos s'executen en el ordre en què els posem (d'esquerra a dreta). Així, per exemple, pot ser molt convenient posar el primer operador el \$match, que és el de seleccionar documents, així les altres operacions es faran sobre menys documents i aniran més ràpides.

Cada paràmetre del aggregate, és a dir, cada operador tindrà format JSON, i per tant sempre serà de l'estil:

```
{ $operador : { clau:valor , ... } }
```

Abans de veure els comandos d'agrupació, mirem un exemple per a poder adonar-nos de la potència

Operador \$match

Servirà per a filtrar els documents. Aleshores, l'agregació només afectarà als documents seleccionats. Es poden utilitzar tots els operadors que hem anat estudiant.

El següent exemple selecciona (per a aggregate, però com no fem res més senzillament selecciona els documents) els documents de l'editorial Planeta.

```
> db.libro.aggregate({$match:{editorial:"Planeta"}})
```

En el següent exemple, a més de seleccionar els de l'editorial Planeta després apliquem una projecció sobre els camps títol i editorial, per a poder visualitzar millor el resultat.

```
> db.libro.aggregate({$match:{editorial:"Planeta"}},{ $project:{titulo:1,editorial:1}})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta" }
```

Operador \$project

Ens permet projectar sobre determinats camps del document, però és molt més complet que en la projecció "normal" que havíem fet fins ara, ja que permet també renombrar camps, fer càlculs, etc.

Projecció

La manera més senzilla, evidentment és projectar sobre alguns camps dels existents, i el funcionament és idèntic al de l'altra vegada (valors 1 per a que apareguen, 0 per a que no apareguen; per defecte `_id` sempre apareix):

```
> db.libro.aggregate({$project:{titulo:1,editorial:1}})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" : "Planeta" }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" : "Plaza & Janes" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "editorial" : "Gigamesh" }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "editorial" : "Debolsillo" }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" : "Embolsillo" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "editorial" : "Planeta" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

Renomenar

\$project també ens permet renombrar camps existents (després veurem que també càlculs). La manera serà posar d'aquesta manera:

```
{ $project : { "nom_nou" : "$camp_vell" } }
```

El secret està en el dòlar que va davant del camp vell, ja que d'aquesta manera ens referim al valor d'aquest camp. Així per exemple renomnem el camp **enstock** a **disponible**, a banda de traure el títol:

```
> db.libro.aggregate({$project:{titulo:1 , disponible:"$enstock"}})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "disponible" : true }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "disponible" : true }
```

```
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "disponible" : true }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "disponible" : false }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "disponible" : true }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "disponible" : false }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "disponible" : true }
```

Camps calculats

Amb aquest nom genèric ens referirem a tots els càlculs, expressions i més coses que podrem posar per a transformar el que ja tenim. Com veiem, açò és molt més potent que la projecció normal.

- **Expressions matemàtiques:** Podrem aplicar fórmules per a sumar (\$add), restar (\$subtract), multiplicar (\$multiply), dividir (\$divide) i més coses (potència, arrel quadrada, valor absolut, mòdul, ...). Cada operació té el seu operador que serà una paraula precedida pel dòlar, i amb la sintaxi de JSON, on posarem els operands en un array.

Per exemple, traurem títol del llibre, preu i preu en pessetes (multiplicant per 166.386)

```
> db.libro.aggregate({$project:{titulo:1 , precio:1 , preu_pessetes:{$multiply:[$precio" , 166.386]}}})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "precio" : 21.75, "preu_pessetes" : 3618.8955 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "precio" : 21.75, "preu_pessetes" : 3618.8955 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y fuego 1", "precio" : 9.5, "preu_pessetes" : 1580.667 }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros", "precio" : 9.45, "preu_pessetes" : 1572.3476999999998 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "precio" : 11, "preu_pessetes" : 1830.2459999999999 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio", "precio" : 17.23, "preu_pessetes" : 2866.83078 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" : 15.9, "preu_pessetes" : 2645.5374 }
```

- **Expressions de dates:** Ja veurem i ja podem anar intuïnt que moltes agregacions estaran basades en el temps, per a poder fer consultes de documents de la setmana passada, o el mes passat, ... Per a poder fer aquestes agregacions, hi ha un conjunt d'expressions que permeten extraure fàcilment d'una data el seu dia, mes, any, ... en forma de número

Són les expressions: \$year, \$month, \$week, \$dayOfMonth, \$DayOfWeek, \$dayOfYear, \$hour, \$minute i \$second.

En el següent exemple traurem tots els documents, projectant per la data, any i mes:

```
> db.libro.aggregate({$project : {fecha:1 , año:{$year:"$fecha"} , mes:{$month:"$fecha"}}})
{ "_id" : "9788408117117", "fecha" : ISODate("2013-08-29T00:00:00Z"), "año" : 2013, "mes" : 8 }
{ "_id" : "9788401342158", "fecha" : ISODate("2014-03-01T00:00:00Z"), "año" : 2014, "mes" : 3 }
{ "_id" : "9788496208919", "fecha" : ISODate("2011-11-24T00:00:00Z"), "año" : 2011, "mes" : 11 }
{ "_id" : "9788499088075", "fecha" : ISODate("2009-01-09T00:00:00Z"), "año" : 2009, "mes" : 1 }
{ "_id" : "9788415140054", "fecha" : ISODate("2012-10-30T00:00:00Z"), "año" : 2012, "mes" : 10 }
{ "_id" : "9788408113331", "fecha" : ISODate("2013-06-04T00:00:00Z"), "año" : 2013, "mes" : 6 }
{ "_id" : "9788468738895", "fecha" : ISODate("2014-02-06T00:00:00Z"), "año" : 2014, "mes" : 2 }
```

- **Expressions de strings:** Ens permeten manipular els strings per a extraure subcadena, concatenar, passar a majúscules o minúscules. Aquestes són algunes de les funcions:

- \$substr: [exp , inici , llargària] : extrau una subcadena del string del primer paràmetre, des de la posició que indica el segon paràmetre (o primer caràcter) i tants caràcters com el tercer paràmetre
- \$concat : [exp1 , exp2 , ...] : concatena les expressions que hi ha en l'array
- \$toLower : exp i \$toUpper : exp : converteixen l'expressió a majúscules i minúscules respectivament

Per exemple, anem a traure el títol dels llibres amb l'autor entre parèntesis:

```
> db.libro.aggregate({$project: { "Llibre:" : {$concat : ["$titulo" , " (" , "$autor" , ")"]}}} )
{ "_id" : "9788408117117", "Llibre:" : "Circo Máximo (Santiago Posteguillo)" }
{ "_id" : "9788401342158", "Llibre:" : "El juego de Ripper (Isabel Allende)" }
{ "_id" : "9788496208919", "Llibre:" : "Juego de tronos: Canción de hielo y fuego 1 (George R.R. Martin)" }
{ "_id" : "9788499088075", "Llibre:" : "La ladrona de libros (Markus Zusak)" }
{ "_id" : "9788415140054", "Llibre:" : "La princesa de hielo (Camilla Lackberg)" }
{ "_id" : "9788408113331", "Llibre:" : "Las carreras de Escorpio (Maggie Stiefvater)" }
{ "_id" : "9788468738895", "Llibre:" : "Las reglas del juego (Anna Casanovas)" }
```

I ara el mateix, però amb el títol en majúscules:

```
> db.libro.aggregate({$project: { "Llibre:" : {$concat : [{"toUpper:"$titulo"} , " (" , "$autor" , ")"]}}} )
{ "_id" : "9788408117117", "Llibre:" : "CIRCO MÁXIMO (Santiago Posteguillo)" }
{ "_id" : "9788401342158", "Llibre:" : "EL JUEGO DE RIPPER (Isabel Allende)" }
{ "_id" : "9788496208919", "Llibre:" : "JUEGO DE TRONOS: CANCIÓN DE HIELO Y FUEGO 1 (George R.R. Martin)" }
```

```
R.R. Martin)" }
{ "_id" : "9788499088075", "Llibre:" : "LA LADRONA DE LIBROS (Markus Zusak)" }
{ "_id" : "9788415140054", "Llibre:" : "LA PRINCESA DE HIELO (Camilla Lackberg)" }
{ "_id" : "9788408113331", "Llibre:" : "LAS CARRERAS DE ESCORPIO (Maggie Stiefvater)" }
{ "_id" : "9788468738895", "Llibre:" : "LAS REGLAS DEL JUEGO (Anna Casanovas)" }
```

Operador \$group

Realitza grups sobre els documents seleccionats prèviament, per a valors iguals del camp o expressions que determinem. Posteriorment, amb els grups, podem realitzar operacions, com sumar o traure la mitjana d'alguna quantitat dels documents del grup, o el màxim o mínim, ...

Per a poder agrupar, haurem de definir com a `_id` del grup el camp o camps pels valors dels quals volem agrupar. Per exemple, si volem agrupar els llibres per l'editorial, haurem de definir el `_id` del grup el camp editorial

```
$group : { "_id" : camp o camps }
```

Si agrupem per un únic camp, senzillament el posem amb un dòlar davant i entre cometes. Si és més d'un camp, els posem com un objecte. Per exemple, agrupem per editorial:

```
> db.libro.aggregate( { $group : { "_id" : "$editorial" } } )
{ "_id" : "Debolsillo" }
{ "_id" : null }
{ "_id" : "Gigamesh" }
{ "_id" : "Embolsillo" }
{ "_id" : "Plaza & Janes" }
{ "_id" : "Planeta" }
```

Podem observar com hi ha algun llibre que no té editorial.

Ara agrupem per any de publicació (l'extraurem del camp `fecha`):

```
> db.libro.aggregate( { $group : { "_id" : { "any" : { $year : "$fecha" } } } } )
{ "_id" : { "any" : 2012 } }
{ "_id" : { "any" : 2009 } }
{ "_id" : { "any" : 2011 } }
{ "_id" : { "any" : 2014 } }
{ "_id" : { "any" : 2013 } }
```

I ara agrupem per editorial i any de publicació (els dos llibres de Planeta són del 2013)

```
> db.libro.aggregate( { $group : { "_id" : { "Editorial" : "$editorial" , "any" : { $year : "$fecha" } } } } )
{ "_id" : { "Editorial" : "Embolsillo", "any" : 2012 } }
{ "_id" : { "any" : 2014 } }
{ "_id" : { "Editorial" : "Debolsillo", "any" : 2009 } }
{ "_id" : { "Editorial" : "Gigamesh", "any" : 2011 } }
{ "_id" : { "Editorial" : "Plaza & Janes", "any" : 2014 } }
{ "_id" : { "Editorial" : "Planeta", "any" : 2013 } }
```

Operadors d'agrupació

Ens permetran fer alguna operació sobre els documents del grup. Es posen com a segon paràmetre del grup (després de la definició del `_id`).

- **\$sum : valor** : sumarà el valor de tots els documents del grup. El valor pot ser un camp numèric, o alguna altra cosa més complicada.
- **\$avg : valor** : calcularà la mitjana dels valors per als documents del grup
- **\$max : valor** : màxim
- **\$min : valor** : mínim
- **\$first : exp** : agafarà el primer valor de l'expressió del grup, ignorant les altres del grup
- **\$last : exp** : agafarà l'últim

Per exemple, la suma dels preus dels llibres de cada editorial:

```
> db.libro.aggregate( { $group : { "_id" : "$editorial" , "suma_preus" : { $sum : "$precio" } } } )
{ "_id" : "Debolsillo", "suma_preus" : 9.45 }
{ "_id" : null, "suma_preus" : 15.9 }
{ "_id" : "Gigamesh", "suma_preus" : 9.5 }
{ "_id" : "Embolsillo", "suma_preus" : 11 }
{ "_id" : "Plaza & Janes", "suma_preus" : 21.75 }
{ "_id" : "Planeta", "suma_preus" : 38.980000000000004 }
```

O la mitjana dels preus de cada any:

```
> db.libro.aggregate( { $group : { "_id" : { "any" : { $year : "$fecha" } } , "mitjana_preus" : { $avg : "$precio" } } } )
{ "_id" : { "any" : 2012 } , "mitjana_preus" : 11 }
{ "_id" : { "any" : 2009 } , "mitjana_preus" : 9.45 }
{ "_id" : { "any" : 2011 } , "mitjana_preus" : 9.5 }
{ "_id" : { "any" : 2014 } , "mitjana_preus" : 18.825 }
{ "_id" : { "any" : 2013 } , "mitjana_preus" : 19.490000000000002 }
```

Operador \$sort

Serveix per a ordenar i segueix la mateixa sintàxi que en les consultes normal (1: ordre ascendent; -1: ordre descendent). Podem ordenar pels camps normals o per camps calculats.

Per exemple ordenem per la suma de preus de cada editorial:

```
> db.libro.aggregate( { $group : { "_id" : "$editorial" , "suma_preus" : { $sum : "$precio" } } } , { $sort : { suma_preus : 1 } } )
{ "_id" : "Debolsillo", "suma_preus" : 9.45 }
{ "_id" : "Gigamesh", "suma_preus" : 9.5 }
{ "_id" : "Embolillo", "suma_preus" : 11 }
{ "_id" : null, "suma_preus" : 15.9 }
{ "_id" : "Plaza & Janes", "suma_preus" : 21.75 }
{ "_id" : "Planeta", "suma_preus" : 38.980000000000004 }
```

I ara ordenem de forma descendent per la mitjana de preus de cada any:

```
> db.libro.aggregate( { $group : { "_id" : {"any":{"year":"$fecha"}} , "mitjana_preus":{$avg:"$precio"} } } , {$sort:{"mitjana_preus":-1}} )
{ "_id" : { "any" : 2013 }, "mitjana_preus" : 19.490000000000002 }
{ "_id" : { "any" : 2014 }, "mitjana_preus" : 18.825 }
{ "_id" : { "any" : 2012 }, "mitjana_preus" : 11 }
{ "_id" : { "any" : 2011 }, "mitjana_preus" : 9.5 }
{ "_id" : { "any" : 2009 }, "mitjana_preus" : 9.45 }
```

Operador \$limit

Limita el resultat del aggregate al número indicat.

Per exemple, els tres anys de mitjana de preus més cara. És com l'últim exemple, afegint el límit:

```
> db.libro.aggregate({$group:{"_id":{"any":{"year":"$fecha"}}, "mitjana_preus":{"avg":"$precio"}}} , {$sort:{"mitjana_preus":-1}} , {$limit:3})
{ "_id" : { "any" : 2013 }, "mitjana_preus" : 19.490000000000002 }
{ "_id" : { "any" : 2014 }, "mitjana_preus" : 18.825 }
{ "_id" : { "any" : 2012 }, "mitjana_preus" : 11 }
```

Operador \$skip

Salta el número indicat

En l'exemple anterior, ara saltem els 3 primers:

```
> db.libro.aggregate({$group:{"_id":{"any":{"year":"$fecha"}}, "mitjana_preus":{"avg":"$precio"}}} , {$sort:{"mitjana_preus":-1}} , {$skip:3})
{ "_id" : { "any" : 2011 }, "mitjana_preus" : 9.5 }
{ "_id" : { "any" : 2009 }, "mitjana_preus" : 9.45 }
```


Per a poder connectar des de Java ens serà suficient amb un driver, que haurem d'incorporar al projecte. En el moment de fer aquests apunts, l'últim driver disponible és el següent:

<https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongo-java-driver/3.2.2/mongo-java-driver-3.2.2.jar>

Per a separar les proves i exercicis de la part de **Redis**, en creem en el projecte **Tema8** un paquet anomenat **provesMongo**.

Connexió

La connexió és tan senzilla com el següent:

```
MongoClient con = new MongoClient("localhost" , 27017);
MongoDatabase bd = con.getDatabase("test");
```

És a dir, obtenim un objecte **MongoClient** passant-li al constructor l'adreça del servidor i el port de connexió (que per defecte és 27017).

Posteriorment hem de connectar amb la Base de Dades. Ja havíem comentat en la instal·lació de Mongo que nosaltres només utilitzàrem una Base de Dades ja creada anomenada **test**. Obtenim un objecte **MongoDatabase** que farà referència a la Base de Dades, i és l'objecte que utilitzarem a partir d'ara. Evidentment ho podríem haver fet en una única línia.

Si el servidor no el tenim en la mateixa màquina, només haurem de substituir **localhost** per l'adreça on estiga el servidor.

Per a tancar la connexió:

```
con.close();
```

Inserció de documents

Des de Java podrem inserir documents amb la mateixa facilitat que des de la consola. Només haurem de crear un objecte **Document** de **BSON** (recordeu que és el format intern de Mongo, absolutament similar a **JSON**). La manera d'anar posant parelles calu valor en aquest document és per mig del mètode **put**. Fem un exemple molt senzill on senzillament guardem un document amb una parella clau-valor d'un missatge:

```
import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;


public class Proval {

    public static void main(String[] args) {
        MongoClient con = new MongoClient("localhost" , 27017);
        MongoDatabase bd = con.getDatabase("test");

        Document doc = new Document();
        doc.put("msg4", "Missatge inserit des de Java");
        bd.getCollection("exemple").insertOne(doc);

        con.close();
    }
}
```

Segurament traurà avisos en la consola, però només són avisos. Podem comprovar en la terminal com s'ha inserit el document:



```
alvar@Ubuntet: ~
alvar@Ubuntet: ~
alvar@Ubuntet:~$ ./mongodb-linux-x86_64-ubuntu1404-3.2.1/bin/mongo
MongoDB shell version: 3.2.1
connecting to: test
Server has startup warnings:
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten]
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'alw
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten]
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'alwa
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-03-13T17:44:54.219+0100 I CONTROL [initandlisten]
> db.exemple.find()
{ "_id" : ObjectId("56ce310bc61e04ba81def50b"), "msg" : "Hola, què tal?", "titol" : "Missatge 1", "destinatari" : "Ferran" }
{ "_id" : ObjectId("56ce31f6c61e04ba81def50c"), "msg2" : "Què? Com va la cosa?" }
{ "_id" : 100, "msg" : "Hola, què tal?", "titol" : "Missatge 1", "destinatari" : "Ferran" }
{ "_id" : ObjectId("56e5b920bf0fc416f78bfa9"), "msg4" : "Missatge inserit des de Java" }
>
```


Consultes

Tenim el mètode **find()** per a fer consultes, i li podem posar un document com a paràmetre per a seleccionar determinats documents o traure determinada informació.

```
import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoDatabase;

public class Prova2 {

    public static void main(String[] args) {
        MongoClient con = new MongoClient("localhost" , 27017);
        MongoDatabase bd = con.getDatabase("test");

        FindIterable<Document> llibres = bd.getCollection("libro").find();

        for (Document llibre : llibres)
            System.out.println(llibre.get("titulo"));

        con.close();
    }
}
```

I com comentàvem podem posar com a paràmetres en el find() per a seleccionar determinats documents, ordenar, etc. Només hem de cuidar que ho hem de posar en **JSON** (millor dit **BSON**), i per tant haurem de crear un document per a això.

```
import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoDatabase;

public class Prova3 {

    public static void main(String[] args) {
        MongoClient con = new MongoClient("localhost" , 27017);
        MongoDatabase bd = con.getDatabase("test");

        Document ordenar = new Document();
        ordenar.put("precio", -1);

        FindIterable<Document> llibres = bd.getCollection("libro").find().sort(ordenar);

        for (Document llibre : llibres)
            System.out.println("Titol:" + llibre.get("titulo") + ". Preu:" +
llibre.get("precio"));

        con.close();
    }
}
```



Exercici 8.1

Sobre la Base de Dades del Servidor local fer les següents operacions, tant per a guardar una sèrie de dades, com per a recuperar-les. Sempre posarem a les claus el **prefix 999x_**, on heu de substituir 999 per les 3 últimes xifres del vostre DNI, i la x per la lletra del NIF.

Copia-les en un únic fitxer de text, de forma numerada. És aquest fitxer el que hauràs de pujar.

- a. Crea la clau **999x_Nom** amb el teu nom
- b. Crea la clau **999x_Cognoms** amb els teus cognoms. Una de les dues almenys, nom o cognoms, ha de constar de més d'una paraula.
- c. Mostra totes les claus teues, i únicament les teues.
- d. Crea la clau **999x_Adreca**, de tipus Hash, amb els subcamp **carrer**, **numero** i **cp**. No importa que les dades siguin falses. Pots fer-lo en una sentència o en més d'una.
- e. Afegeix a l'anterior el subcamp **poblacio**
- f. Mostra tota la informació de la teua adreça (només la informació, no les subclaus)
- g. Crea la clau **999x_Moduls_DAW**, de tipus Set, amb tots els mòduls del DAW, que són: SI, BD, PR, LM, ED, FOL, DWEC, DWES, DAW, DIW, EIE, PROJ i FCT Pots fer-lo en una o en més d'una sentència.
- h. Crea la clau **999x_Moduls_meus**, de tipus Set, amb tots els mòduls als quals estàs matriculat. Pots fer-lo en una o en més d'una sentència.
- i. Guarda en la clau **999x_Moduls_altres** els mòduls en els quals no estàs matriculat actualment. Ha de ser per mig d'operacions de conjunts. Pots comprovar que el resultat és correcte amb **smembers**
- j. Crea una llista amb el nom **999x_Notes_BD** amb la nota de 4 exercicis de BD. Les notes seran: 7, 9, 6,10. Han de quedar en aquest ordre (no en ordre invers)

