

به نام خدا

# MongoDB

تهیه کننده :

محمدرضا خرمی

5	..... نصب MongoDB
5	..... نصب بر روی ویندوز :
8	..... نصب بر روی لینوکس (نسخه SUSE)
10	..... نصب بر روی لینوکس (Centos)
12	..... ایجاد سرویس MongoDB در ویندوز
	<b>Error! Bookmark not defined.</b> ..... ایجاد سرویس در لینوکس
21	..... استفاده از محیط Shell
28	..... عملیات CRUD (Create, Read, Update, Delete) در MongoDB
39	..... چارت مطابقت مفاهیم SQL به MongoDB
55	..... Aggregation
64	..... Text Search
67	..... Data Models
78	..... Indexes
78	..... ایندکس پیش فرض _id
79	..... ساخت ایندکس
79	..... انواع ایندکس
79	..... تک فیلد
79	..... ایندکس مرکب
80	..... ایندکس Multikey
80	..... ایندکس Geospatial

81	ایندکس Tesxt
81	ایندکس Hashe
82	Storage
82	WiredTiger
82	MMAPv1
84	Security
87	Administration
91	Storage Engine
92	RAID
92	MongoDB on Linux
92	Kernel and File Systems
93	Backup & Restore
93	بک آپ با Mongodump :
99	Monitoring
99	mongotop •
100	mongostat •
101	http://localhost:28017 •
115	Replication
130	Sharding
131	Configuration File
140	Performance
141	Locking Performance
142	Memory and the MMAPv1 Storage Engine
142	Memory Use
142	Number of Connections

142	..... Database Prtofiling
143	..... Mongobooster
143	..... ساخت دیتابیس با Mongobooster :
146	..... MongoDB در دیتابیس های موجود در
147	..... منابع

## نصب MongoDB

### نصب بر روی ویندوز :

برای نصب این دیتابیس روی ویندوز ابتدا باید از معماری ویندوزی که داریم آگاه شویم ( 32 بیتی یا 64 بیتی ) . لازم به ذکر است در معماری های 32 بیتی دیتابیسهای زیر 2GB پشتیبانی خواهند شد .

برای مشخص کردن ورژن مناسب برای ویندوز از کامندهای زیر استفاده می کنیم

کامند زیر نوع ویندوز را مشخص می کند:

```
wmic os get caption
```

این کامند نوع 32 یا 64 بیت بودن ویندوز را مشخص می کند:

```
wmic os get osarchitecture
```

سپس ورژن هماهنگ آن را از آدرس <https://www.mongodb.org/downloads> دانلود کرده و آن را EXTRACT می کنیم (فایل نصب نسخه ویندوزی در مسیر داکيومنت ها قرار دارد) .

یک دایرکتوری به اسم DATA و درون آن، یک دایرکتوری دیگر به اسم DB ایجاد می کنیم .

```
c:\>md data
```

```
c:\>md data
```

سپس

```
D:\آدرس\mongodb>cd bin
```

```
D:\آدرس\mongodb\bin>mongod.exe --dbpath "d:\آدرس\mongodb\data"
```

در این مرحله در خط آخر می بایست پیام ***waiting for connections*** را ببینیم تا مطمئن شویم دیتابیس بالا آمده است .

مشخصات هر کدام از سرویس ها mongodb و Utility ها به شرح زیر می باشد

Component Set	Binaries windows - Linux
Server	نقش سرور را در مونگو دی بی دارد. mongod.exe - mongod
Router	نقش Router را در مونگو دی بی دارد. mongos.exe - mongos
Client	برای اتصال به دیتابیس مونگو دی بی استفاده می شود. mongo.exe - mongo
MonitoringTools	ابزار های مانیتورینگ در مونگو دی بی می باشند. mongostat.exe, mongotop.exe - mongostat, mongotop
ImportExportTools	ابزار های برای بکاپ گیری در مونگو دی بی می باشند. mongodump.exe, mongorestore.exe, mongoexport.exe, mongoimport.exe - mongodump, mongorestore, mongoexport, mongoimport

MiscellaneousTools

bsondump<sup>1</sup>.exe, mongofiles.exe<sup>2</sup>, mongooplog<sup>3</sup>.exe, mongoperf<sup>4</sup>.exe - bsondump, mongofiles,

---

1 - bsondump برای گرفتن دامپ با فرمت bson استفاده می شود

2 - این یوتیلیتی برای تغییر فایل های حجیم با استفاده از Gridfs می باشد.

3 - mongooplog تونری ساده برای کپی کردن oplog از سرور ریموت به سرور لوکال می باشد

4 - mongoperf یوتیلیتی برای چک کردن وضعیت پرفورمنس دیسک می باشد

## نصب بر روی لینوکس (نسخه SUSE)

توجه: این راهنمای نصب فقط مختص سیستم های 64بیتی میباشد.

برای نصب ورژن 3.0 این دیتابیس بر روی سیستم عامل لینوکس به ترتیب زیر عمل میکنیم:

### 1) پیکربندی سیستم مدیریت پکیج (zypper):

برای بدست آوردن نسخه شماره 3.0 ، از کامند زیر استفاده میکنیم:

```
sudo zypper addrepo --no-gpgcheck https://repo.mongodb.org/zypper/suse/11/mongodb-org/3.0/x86_64/ mongodb
```

### 2) نصب پکیج mongodb و ابزارهای وابسته ی آن:

برای نصب این ورژن به همراه همه ابزارهای موجود، از کامند زیر استفاده میکنیم:

```
sudo zypper -n install mongodb-org
```

اما برای نصب این ورژن به همراه ابزارهای بخصوص (نه همه ابزارها) از کامندی مشابه زیر استفاده میکنیم:

```
sudo zypper install mongodb-org-3.0.13 mongodb-org-server-3.0.13 mongodb-org-shell-3.0.13 mongodb-org-mongos-3.0.13 mongodb-org-tools-3.0.13
```

**توجه کنید** که zypper به طور خودکار ورژن های جدیدتر را بعد از انتشار، دریافت کرده و نصب مینماید. برای جلوگیری از این آپگریدها باید به وسیله کامند زیر، پکیج را pin کنیم:

```
sudo zypper addlock mongodb-org-3.0.13 mongodb-org-server-3.0.13 mongodb-org-shell-3.0.13 mongodb-org-mongos-3.0.13 mongodb-org-tools-3.0.13
```

برای اجرای MongoDB به روش زیر عمل میکنیم:

-- mongodb به طور پیش فرض داده های خود را در مسیر /var/lib/mongo و فایل های لاگ را در مسیر /var/log/mongodb ذخیره میکند. اما میتوان این مسیرها را در /etc/mongod.conf تغییر داد.



بعلاوه اگر user ای که از mongodb استفاده میکند را تغییر داده باشیم، باید حق دسترسی را به مسیرهای `/var/lib/mongo` و `/var/log/mongodb` بدهیم تا یوزر اجازه دسترسی به منابع موجود در این مسیرها را پیدا کند.

روش اجرای `mongodb`:

1) برای استارت کردن `mongod` از کامند زیر استفاده میکنیم:

```
sudo service mongod start
```

2) برای بررسی موفقیت آمیز بودن استارت `mongodb` بوسیله مشاهده فهرست فایل لاگ در `/var/log/mongodb/mongod.log` از کامند زیر استفاده میکنیم:

```
[initandlisten] waiting for connections on port <port>
```

که `<port>` همان شماره پورت تعیین شده در فایل کانفیگ `/etc/mongod.conf` میباشد. همچنین میتوان بوسیله کامند زیر مطمئن شد که MongoDB بعد از ری بوت سیستم استارت خواهد شد:

```
sudo chkconfig mongod on
```

3) برای استاپ کردن `mongod` از کامند زیر استفاده میشود:

```
sudo service mongod stop
```

همچنین میتوان برای استاپ کردن `mongod` در حال اجرا، از ترکیب دکمه های `ctrl+C` در ترمینالی که `mongod` در حال اجراست استفاده کنید.

4) برای `restart` کردن `mogod` از کامند زیر استفاده میکنیم:

```
sudo service mongod restart
```

برای `uninstall` کردن `mongodb` و پاک کردن آن از سیستم، باید اپلیکیشن ها، فایل های کانفیگ و هر مسیری که شامل داده ها و لاگ های `mongodb` میشود را پاک کنیم. **توجه داشته باشید** که این روند برگشت ناپذیر بوده و قبل از هر اقدامی باید مطمئن شد که از داده ها و کانفیگ ها پشتیبان گیری انجام شده باشد.

(1) برای پاک کردن پکیج هایی که بر روی سیستم نصب شده است از کامند زیر استفاده میکنیم:

```
sudo zypper remove $(rpm -qa | grep mongodb-org)
```

(2) برای پاک کردن داده ها و فایل های لاگ از کامند زیر استفاده میشود:

```
sudo rm -r /var/log/mongodb  
sudo rm -r /var/lib/mongo
```

### نصب بر روی لینوکس (Centos)

پکج های نصب `mongodb` در مسیر زیر قرار دارند

[192.168.53.13\\Share\\Documents\\Mongodb\\Installation\\RPM\\Pckage](http://192.168.53.13\\Share\\Documents\\Mongodb\\Installation\\RPM\\Pckage)

برای نصب در حالت معمولی تنها پکیج ها نصب می شوند.

```
rpm -ivh mongo*
```

بعد از نصب فایل کانفیگ `mongo` که در مسیر زیر قرار دارد باید ویرایش شود و مسیر لاگ – مسیر دیتا فایل ها و شبکه و همچنین فعال کردن `authentication` تغییر یابد

قبل از تغییر سرویس دیتابیس استاپ شود

```
systemctl stop mongod
```

```
vim /etc/mongod.conf
```

مسیر لاگ طبق استاندارد گروه باید در پارتیشن /dbdata1/mongodLog قرار بگیرد

systemLog:

path: /dbdata1/mongodLog/mongod.log

مسیر ذخیره شدن دیتا در پارتیشن پیش فرض /dbdata1/mongoData قرار دارد

storage:

dbPath: /dbdata1/mongodata

برای امکان اتصال به سرور باید تنظیمات شبکه به صورت زیر باید تا هم امکان استفاده به صورت لوکال و آی پی فراهم گردد

net:

port: 27017

bindIp: 0.0.0.0

برای فعال سازی authentication تغییر زیر انجام شود

security:

authorization: enabled

## ایجاد سرویس MongoDB در ویندوز

می توان به جای هر بار استارت سرویس MongoDB می توان سرویس مربوطه را به عنوان یکی از سرویس های ویندوز قرارداد تا به صورت خودکار استارت شود.

1. اجرای cmd.exe با دسترسی Administrator

2. دایرکتوری های زیر را ایجاد می کنیم

```
mkdir c:\data\db
```

```
mkdir c:\data\log
```

3. ایجاد فایل کانفیگ در مسیر زیر

```
c:\mongodb\mongod.cfg
```

داخل فایل کانفیگ اطلاعات زیر را از طریق notepad وارد می کنیم

systemLog:

destination: file

path: c:\data\log\mongod.log

storage:

dbPath: c:\data\db

4. سرویس MongoDB را ایجاد می کنیم با استفاده از دستور زیر

```
sc.exe create MongoDB binPath="C:\mongodb\bin\mongod.exe --service --  
config=\"C:\mongodb\mongod.cfg\" DisplayName="MongoDB" start="auto"
```

(با فرض اینکه مسیر نصب mongod در C:\mongodb قرار دارد.)

بعد از اجرای موفق پیغام زیر داده می شود

```
[SC] CreateService SUCCESS
```

5. برای استارت کردن سرویس MongoDB از دستور زیر استفاده می کنیم

```
net start MongoDB
```

6. برای استاپ کردن MongoDB از دستور زیر استفاده نمایید

```
net stop MongoDB
```

7. در صورت نیاز به پاک کردن سرویس MongoDB از دستور زیر استفاده می شود.

```
sc.exe delete MongoDB
```

## تغییرات کرنل در لینوکس

```
cat >> /etc/sysctl.conf <<EOF
fs.file-max = 6815744
kernel.sem = 250 32000 100 128
kernel.shmni = 4096
kernel.shmall = 1073741824
kernel.shmmax = 4398046511104
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
fs.aio-max-nr = 1048576
net.ipv4.ip_local_port_range = 9000 65500
vm.swappiness = 1
vm.dirty_background_ratio = 5
vm.dirty_ratio = 80
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
kernel.panic_on_oops = 1
net.ipv4.conf.<NIC_NAME>.rp_filter = 2
EOF
```

note: NIC\_NAME is the name of the Network Interface Card. "ifconfig -a" will list your NICs.

```
cat >> /etc/security/limits.conf <<EOF
mongodb soft nproc 64000
mongodb hard nproc 64000
mongodb soft nofile 64000
mongodb hard nofile 64000
EOF
```

Append the following to the kernel command line in grub.conf:

```
cat >> /boot/grub2/grub.cfg <<EOF
transparent_hugepage=never
EOF
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

## Command To List Number Of Open File Descriptors

Use the following command command to display maximum number of open file descriptors:

```
cat /proc/sys/fs/file-max
```

75000 files normal user can have open in single login session. To see the hard and soft values, issue the command as follows:

```
# ulimit -Hn
# ulimit -Sn
```

To see the hard and soft values for httpd or oracle user, issue the command as follows:

```
# su - username
In this example, su to oracle user, enter:
# su - oracle
$ ulimit -Hn
$ ulimit -Sn
```

## etting the sysctl parameter aio-max-nr value to 1048576

The Linux kernel provides the Asynchronous non-blocking I/O (AIO) feature that allows a process to initiate multiple I/O operations simultaneously without having to wait for any of them to complete. This helps boost performance for applications that are able to overlap processing and I/O.

The performance can be tuned using the `/proc/sys/fs/aio-max-nr` virtual file in the `proc` file system. The `aio-max-nr` parameter determines the maximum number of allowable concurrent requests.

Another parameter, `/proc/sys/fs/aio-nr`, provides the current system-wide number of asynchronous requests.

Veritas recommends that you set the `aio-max-nr` value to 1048576. This helps HyperScale to perform optimally, in an environment that involves heavy I/O workloads.

**Perform the following steps on all HyperScale compute and data nodes:**

1. To set the `aio-max-nr` value, add the following line to the `/etc/sysctl.conf` file:

```
fs.aio-max-nr = 1048576
```

2. To activate the new setting, run the following command:

## CHANGING NETWORK KERNEL SETTINGS

e default setting in bytes of the socket receive buffer:

```
# sysctl -w net.core.rmem_default=262144
```

The default setting in bytes of the socket send buffer:

```
# sysctl -w net.core.wmem_default=262144
```

The maximum socket receive buffer size which may be set by using the SO\_RCVBUF socket option:

```
# sysctl -w net.core.rmem_max=262144
```

The maximum socket send buffer size which may be set by using the SO\_SNDBUF socket option:

```
# sysctl -w net.core.wmem_max=262144
```

To make the change permanent, add the following lines to the `/etc/sysctl.conf` file, which is used during the boot process:

```
net.core.rmem_default=262144
```

```
net.core.wmem_default=262144
```

```
net.core.rmem_max=262144
```

```
net.core.wmem_max=262144
```

To improve fail over performance in a RAC cluster, consider changing the following IP kernel parameters as well:

```
net.ipv4.tcp_keepalive_time
```

```
net.ipv4.tcp_keepalive_intvl
```

```
net.ipv4.tcp_retries2
```

```
net.ipv4.tcp_syn_retries
```

Changing these settings may be highly dependent on your system, network, and other applications. For suggestions, see Metalink Note:249213.1 and Note:265194.1.



On Red Hat Enterprise Linux systems the default range of IP port numbers that are allowed for TCP and UDP traffic on the server is too low for 9i and 10g systems. Oracle recommends the following port range:

```
# sysctl -w net.ipv4.ip_local_port_range="1024 65000"
```

To make the change permanent, add the following line to the `/etc/sysctl.conf` file, which is used during the boot process:

```
net.ipv4.ip_local_port_range=1024 65000
```

The first number is the first local port allowed for TCP and UDP traffic, and the second number is the last port number.

## pdflush

Type the following command to see current wake up time of pdflush:  
# `sysctl vm.dirty_background_ratio`

Sample outputs:

```
sysctl vm.dirty_background_ratio = 10
```

`vm.dirty_background_ratio` contains 10, which is a percentage of total system memory, the number of pages at which the pdflush background writeback daemon will start writing out dirty data. However, for fast RAID based disk system this may cause large flushes of dirty memory pages. If you increase this value from 10 to 20 (a large value) will result into less frequent flushes:  
# `sysctl -w vm.dirty_background_ratio=20`

## swappiness

Type the following command to see current default value:  
# `sysctl vm.swappiness`

Sample outputs:

```
vm.swappiness = 60
```

The value 60 defines how aggressively memory pages are swapped to disk. If you do not want swapping, than lower this value. However, if your system process sleeps for a long time you may benefit with an aggressive swapping behavior by increasing this value. For example, you can change swappiness behavior by increasing or decreasing the value:

```
# sysctl -w vm.swappiness=100
```

### dirty\_ratio

Type the following command:  
# sysctl vm.dirty\_ratio  
Sample outputs:

```
vm.dirty_ratio = 40
```

The value 40 is a percentage of total system memory, the number of pages at which a process which is generating disk writes will itself start writing out dirty data. This is nothing but the ratio at which dirty pages created by application disk writes will be flushed out to disk. A value of 40 mean that data will be written into system memory until the file system cache has a size of 40% of the server's RAM. So if you've 12GB ram, data will be written into system memory until the file system cache has a size of 4.8G. You change the dirty ratio as follows:  
# sysctl -w vm.dirty\_ratio=25

### kernel.panic\_on\_oops

This parameter controls the kernel's behaviour when an oops or bug is encountered:

- 0: try to continue operation
  - 1: panic immediately. If the 'panic' sysctl is also non-zero then the machine will be rebooted.
- OOPS is a deviation from correct behavior of the Linux kernel, one that produces a certain error log. The better-known kernel panic condition results from many kinds of oops, but other instances of an oops event may allow continued operation with compromised reliability.

This is recommended in a system where we want to have node evicted in case of any hardware failure or any other issue.

note: NIC\_NAME is the name of the Network Interface Card. "ifconfig -a" will list your NICs.

```
cat >> /etc/security/limits.conf <<EOF
mongod    soft    nproc     64000
mongod    hard    nproc     64000
mongod    soft    nofile    64000
mongod    hard    nofile    64000
EOF
```

Append the following to the kernel command line in grub.conf:

```
cat >> /boot/grub2/grub.cfg <<EOF
transparent_hugepage=never
EOF
```

## توضیح sellinux

```
[root@selinux]# cat /etc/selinux/config | grep SELINUX=
# SELINUX= can take one of these three values:
SELINUX=enforcing
[root@selinux]# sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
[root@selinux]# cat /etc/selinux/config | grep SELINUX=
# SELINUX= can take one of these three values:
SELINUX=disabled
```

**Reboot the Server**

<https://admin-docs.com/databases/mongodb/mongodb-installation/install-mongodb-on-centos-7-64-bit/>

## استفاده از محیط Shell

Mongo shell یک اینترفیس جاوا اسکریپتی تعاملی برای MongoDB است. شما میتوانید از mongo shell برای نوشتن کوئری ها ، آپدیت کردن داده ها و انجام دادن کارهای ادمین استفاده کنید.

استارت کردن mongo shell:

- قبل از شروع کار با mongo shell باید مطمئن شوید که MongoDB در حال اجرا میباشد.  
برای استارت کردن mongo shell و متصل شدن به MongoDB ای که بر روی پورت پیش فرض localhost در حال اجراست:

1- از طریق cmd به مسیر نصب MongoDB خود بروید

```
cd <mongodb installation dir>
```

2- برای استارت mongo کامند زیر را تایپ کنید

```
./bin/mongo
```

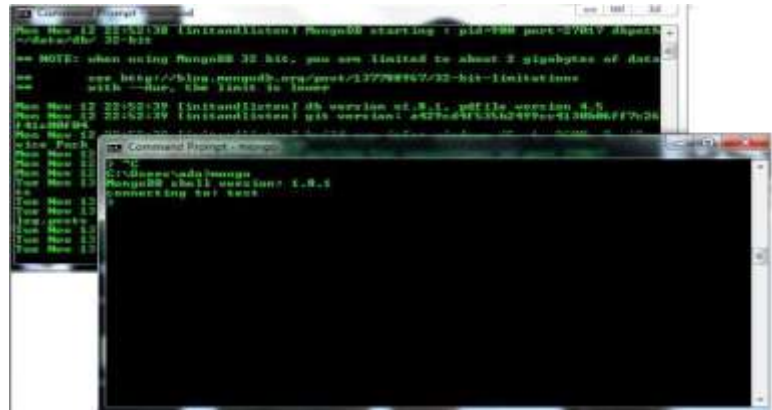
وقتی شما mongo را بدون هیچ آرگومانی اجرا میکنید، mongo shell به MongoDB ای که بر روی سرور پیش فرض localhost (یعنی 27017) در حال اجراست، وصل میشود. اما برای اتصال به صورت ریموت یا اتصال به MongoDB ای بر روی پورت غیر استاندارد باید از کامندهای زیر استفاده کنید:

```
mongo --username <user> --password <pass> --host <host> --port 28015
```

یا بطور خلاصه

```
mongo -u <user> -p <pass> --host <host> --port 28015
```

شما باید بجای موارد <pass>, <user> و <host> مشخصات مورد نیاز خود را وارد کنید. همچنین بعد از port - شماره پورت مناسب خود را قرار بدهید.



### - فایل mongorc.js :

هنگام شروع، mongo دایرکتوری HOME کاربر را برای پیدا کردن فایل جاوا اسکریپتی mongorc.js جستجو میکند. اگر این فایل را پیدا کند، قبل از نمایش دادن cmd، آن را یک بار محاسبه میکند. اگر شما از shell برای ارزیابی یک فایل جاوا اسکریپت استفاده میکنید، mongo بعد از پایان پردازش جاوا اسکریپت، فایل mongorc.js را میخواند. به هر حال شما میتوانید از آپشن -norc برای جلوگیری از لود شدن این فایل استفاده کنید.

### - کار با mongo shell :

برای دیدن دیتابیس‌هایی که در حال استفاده از آن هستید، بنویسید: db

```
db
```

این کار باعث به نمایش درآمدن دیتابیس پیش فرض test میشود. برای سوئیچ کردن بین دیتابیس‌ها از (db) use استفاده کنید. (مانند مثال زیر)

```
use <database>
```

برای دیدن لیست دیتابیس‌های موجود، از show dbs استفاده کنید. وقتی شما اولین بار دیتا را در دیتابیس ذخیره میکنید، مانند ساختن یک کالکشن، MongoDB یک دیتابیس را ایجاد میکند. برای مثال کوئری زیر به طور همزمان دیتابیس myNewDatabase و کالکشن MyCollection را هنگام عملیات اینسرت ایجاد میکند.

```
use myNewDatabase
db.myCollection.insert( { x: 1 } );
```

اگر mongo shell به نام کالکشن دسترسی نداشته باشد، برای مثال اگر نام کالکشن شامل space باشد یا با عدد شروع شود، میتوانید از alternate syntax برای اشاره به کالکشن استفاده کنید

```
db["3test"].find()

db.getCollection("3test").find()
```

### – فرمت دادن به خروجی جستجوها:

برای شکل دادن و مرتب کردن خروجی جستجوها میتوانید `pretty()` را در کامند خود اضافه کنید

```
db.myCollection.find().pretty()
```

بعلاوه شما میتوانید از متدهای زیر در mongo shell استفاده کنید:

- `print()` برای پرینت بدون فرمت بندی
- `print(tojson(<obj>))` برای پرینت با فرمت JSON

### – دستورات multi-line در mongo shell

برای انجام دستورات چند خطی کافیسٹ پرانتز، آکولاد یا براکتی را باز کرده و تا انتهای دستورات آنها را نبندید. Mongo shell تا زمانی که یکی از کاراکترهای نام برده باز باشد، اجرای کد را آغاز نمیکند. مانند کوثری زیر:

```
> if ( x > 0 ) {
... count++;
... print (x);
... }
```

### – Keyboard Shortcuts

Mongo shell از شورتکات های کیبورد پشتیبانی میکند؛ برای مثال:

- میتوانید از دکمه های up/down arrow برای پیمایش کامند هیستوری استفاده کنید.

- میتوانید از (tab) برای کامل کردن خودکار یا لیست کردن امکانات تکمیل کننده استفاده کنید. مثلاً در مثال زیر از (tab) برای کامل کردن نام متدی که با حرف "c" شروع میشود استفاده شده است

```
db.myCollection.c<Tab>
```

البته به دلیل اینکه تعداد متدهایی که با حرف "c" شروع میشوند زیاد است، (tab) یک لیست از تمام آن متدها را برای شما نمایش میدهد.  
لیست زیر تمام شورتکات های موجود را به شما نشان میدهد:

Keybinding	Function
Up arrow	Retrieve previous command from history
Down-arrow	Retrieve next command from history
Home	Go to beginning of the line
End	Go to end of the line
Tab	Autocomplete method/command
Left-arrow	Go backward one character
Right-arrow	Go forward one character
Ctrl-left-arrow	Go backward one word
Ctrl-right-arrow	Go forward one word
Meta-left-arrow	Go backward one word



Keybinding	Function
Meta-right-arrow	Go forward one word
Ctrl-A	Go to the beginning of the line
Ctrl-B	Go backward one character
Ctrl-C	Exit the <code>mongo</code> shell
Ctrl-D	Delete a char (or exit the <code>mongo</code> shell)
Ctrl-E	Go to the end of the line
Ctrl-F	Go forward one character
Ctrl-G	Abort
Ctrl-J	Accept/evaluate the line
Ctrl-K	Kill/erase the line
Ctrl-L or type <code>cls</code>	Clear the screen
Ctrl-M	Accept/evaluate the line
Ctrl-N	Retrieve next command from history
Ctrl-P	Retrieve previous command from history
Ctrl-R	Reverse-search command history
Ctrl-S	Forward-search command history

Keybinding	Function
Ctrl-T	Transpose characters
Ctrl-U	Perform Unix line-discard
Ctrl-W	Perform Unix word-rubout
Ctrl-Y	Yank
Ctrl-Z	Suspend (job control works in linux)
Ctrl-H	Backward-delete a character
Ctrl-I	Complete, same as Tab
Meta-B	Go backward one word
Meta-C	Capitalize word
Meta-D	Kill word
Meta-F	Go forward one word
Meta-L	Change word to lowercase
Meta-U	Change word to uppercase
Meta-Y	Yank-pop
Meta-Backspace	Backward-kill word
Meta-<	Retrieve the first command in command history

Keybinding	Function
Meta->	Retrieve the last command in command history

### - خروج از shell

برای خارج شدن از shell از متد `quit()` یا شورتکات `<ctrl-c>` استفاده کنید.






## عملیات CRUD (Create, Read, Update, Delete) در MongoDB

### Create operation

اَعمال create یا insert، داکيومنت های جدیدی را به کالکشن ها اضافه میکنند. اگر یک کالکشن وجود نداشته باشد، عمل insert باعث ساخت آن خواهد شد.

MongoDB متدهای زیر را برای اینسرت داکيومنتها به داخل کالکشن ها فراهم کرده است:

- `db.collection.insert()`
- `db.collection.insertOne()` در نسخه های 3.2 به بعد
- `db.collection.insertMany()` در نسخه های 3.2 به بعد

<code>Db.users.insert (</code>		Collection		document
<code>{</code>				
<code>  Name: "sue",</code>		field:value		
<code>  Age: 26,</code>		field:value		
<code>  Status: "A"</code>		field:value		
<code>}</code>				
<code>)</code>				

```

db.users.insert (
  {
    name: "sue",
    age: 26,
    status: "A"
  }
)

```

← collection

← field: value

← field: value

← field: value

} document

توضیحات بیشتر در مورد هر یک از این متدها را از [اینجا](#) بخوانید.

### Read operation

اَعمال read، داکيومنت ها را از کالکشن ها فراخوانی میکند. MongoDB متد زیر را برای خواندن داکيومنت ها از کالکشن ها فراهم کرده است:

- db.collection.find()

```

db.users.find(
  { age: { $gt: 18 } },
  { name: 1, address: 1 }
).limit(5)

```

← collection

← query criteria

← projection

← cursor modifier

در مورد این متد در ادامه توضیحات بیشتری ارائه خواهد شد.

### Update operation

اَعمال update، داکيومنت های موجود را در کالکشن ها تغییر میدهد. MongoDB متدهای زیر را برای آپدیت داکيومنت های موجود در کالکشن ها فراهم کرده است:

- db.collection.update()
- db.collection.updateOne() در نسخه های 3.2 به بعد
- db.collection.updateMany() در نسخه های 3.2 به بعد
- db.collection.replaceOne() در نسخه های 3.2 به بعد

```
db.users.update(  
  { age: { $gt: 18 } },  
  { $set: { status: "A" } },  
  { multi: true }  
)
```

← collection  
← update criteria  
← update action  
← update option

توضیحات بیشتر در مورد این متدها را از [اینجا](#) بخوانید.

## Delete operation

اعمال delete داکيومنت ها را از کالکشن ها حذف میکند. MongoDB متدهای زیر را برای حذف کردن کالکشن ها از داکيومنت ها ارائه داده است:

- db.collection.remove()
- db.collection.deleteOne()
- db.collection.deleteMany()

```
db.users.remove(  
  { status: "D" }  
)
```

← collection  
← remove criteria

توضیحات بیشتر در مورد این متدها را از [اینجا](#) بخوانید.

## ■ متدهای insert

همانطور که گفته شد MongoDB سه متد زیر را برای اینسرت داکيومنت ها فراهم کرده است:

- db.collection.insert()
- db.collection.insertOne() در نسخه های 3.2 به بعد
- db.collection.insertMany() در نسخه های 3.2 به بعد

بطور کلی در MongoDB اگر یک کالکشن در دیتابیس وجود نداشته باشد عمل insert باعث ساخته شدن آن کالکشن خواهد شد.

= فیلد \_id : در MongoDB هر داکيومنت به همراه یک فیلد یکتا به نام \_id درون کالکشن ذخیره میشود که این فیلد مانند primary key عمل میکند. اگر در هنگام اینسرت، این فیلد در داکيومنت قرار داده نشود، MongoDB به طور خودکار \_id را به آن داکيومنت اضافه میکند

### 1 db.collection.insertOne

این متد یک تک داکيومنت را به کالکشن اضافه میکند. مثال زیر یک داکيومنت را به کالکشن users اضافه میکند:

```
db.products.insertOne( { _id: 10, "item" : "packing peanuts", "qty" : 200 } );
```

### 2 db.collection.insertMany

این متد قابلیت اضافه کردن چند داکيومنت را به کالکشن مورد نظر دارد. در مثال زیر سه داکيومنت به کالکشن users اضافه شده است:

```
db.products.insertMany( [
  { item: "card", qty: 15 },
  { item: "envelope", qty: 20 },
  { item: "stamps" , qty: 30 }
] );
```

### 3 db.collection.insert

این متد میتواند از یک تا چند داکيومنت را به کالکشن اضافه کند. برای افزودن یک داکيومنت، آن را به متد پاس بدهید و برای افزودن چندین داکيومنت، یک آرایه از داکيومنت ها را به متد بفرستید. مثالهای این متد مشابه مثال دو متد قبلی است.

```
db.products.insert( { _id: 10, item: "box", qty: 20 } )

db.products.insert(
  [
```

```
{ _id: 11, item: "pencil", qty: 50, type: "no.2" },  
{ item: "pen", qty: 20 },  
{ item: "eraser", qty: 25 }  
]  
)
```

### ■ متدهای Update

به طور کلی متدهای `updateOne()` و `updateMany()` و `update()` برای بروز رسانی اطلاعات داکيومنت ها استفاده میشود. تفاوت دو مورد اول در آپدیت کردن یکی یا گروهی از داکيومنت هایی است که با شرط قید شده در آپدیت هماهنگ هستند. اما بجای هر دوی این متدها میتوان از مورد سوم (یعنی `update()`) استفاده کرد. این متد بصورت پیش فرض تنها یک داکيومنت را آپدیت میکند. اما برای بروز رسانی بیش از یک داکيومنت کافیسیت از آپشن `multi` درون متد استفاده کنید. در زیر نمونه ای از هر متد را خواهید دید:

```
db.myColl.updateOne(  
  { category: "cafe" },  
  { $set: { status: "Updated" } },  
  { collation: { locale: "fr", strength: 1 } }  
);
```

```
db.myColl.updateMany(  
  { category: "cafe" },  
  { $set: { status: "Updated" } },  
  { collation: { locale: "fr", strength: 1 } }  
);
```

```
db.books.update(  
  { _id: 1 },  
  {  
    $inc: { stock: 5 },  
    $set: {  
      item: "ABC123",  
      "info.publisher": "2222",
```



```
tags: [ "software" ],
"ratings.1": { by: "xyz", rating: 3 }
}
}
)
```

```
db.books.update(
  { stock: { $lte: 10 } },
  { $set: { reorder: true } },
  {
    multi: true,
    writeConcern: { w: "majority", wtimeout: 5000 }
  }
)
```

در مورد متد `replaceOne()` باید بدانید که این متد مانند `updateOne()` عمل میکند با این تفاوت که فیلد `_id` را نمیتوان با این متد بروز رسانی کرد. یعنی برای جایگزین کردن مورد جدید بجای داکيومنت قبلی با این متد، یا نباید فیلد `_id` را ذکر کرد، یا باید با همان مقدار قبلی آن را بکار برد.

```
db.restaurant.replaceOne(
  { "name" : "Central Perk Cafe" },
  { "name" : "Central Pork Cafe", "Borough" : "Manhattan" }
);
```

### ■ متدهای delete

متدهای `deleteMany()` و `remove()` برای حذف کردن چندین داکيومنت از کالکشن ها استفاده میشود. درحالی که متد `deleteOne()` تنها یک داکيومنت را از کالکشن مورد نظر حذف میکند. در زیر نمونه هایی از این سه متد با قید شرط یا بدون قید شرط برای آنها نشان میدهد:

- حذف همه داکيومنت های کالکشن:

```
db.orders.deleteMany();
```

```
db.products.remove()
```

- حذف همه داکيومنت های کالکشن که شامل فیلتر ذکر شده هستند:

```
db.orders.deleteMany( { "client" : "Crude Traders Inc." } );
```

```
db.products.remove( { qty: { $gt: 20 } } )
```

- حذف اولین داکيومنت کالکشن که با فیلتر ذکر شده همخوانی دارد:

```
db.orders.deleteOne( { "_id" : ObjectId("563237a41a4d68582c2509da") } );
```

```
db.products.remove( { qty: { $gt: 20 }, 1 } )
```

نکته اینکه برای حذف یک داکيومنت با متد `remove()` باید پارامتر دوم متد را با عدد 1 مشخص کنید.

### ■ اعمال Bulk Write

MongoDB به کاربرانش این امکان را داده که اعمال نوشتن در دیتابیس ها را در حجم بالا انجام دهند. این اعمال فقط بر روی یک کالکشن اثر میگذارد. MongoDB به اپلیکیشن ها این امکان را میدهد که حجم و سطح قابل قبول برای اعمال `bulk write` را تعیین کنند. متد `db.collection.bulkwrite()` توانایی انجام اعمال `bulk insert`، `bulk update`، `bulk remove` را فراهم میکند. همچنین MongoDB عمل `bulk insert` را در متد `db.collection.insertMany()` هم انجام میدهد.

### • اعمال Ordered یا Unordered

اعمال `bulk write` میتوانند مرتب شده یا مرتب نشده باشند.

با لیست اعمال مرتب شده (`ordered`)، MongoDB اعمال را بطور سریالی انجام میدهد، اما با لیست اعمال مرتب نشده (`unordered`)، MongoDB اعمال را به شکل موازی انجام خواهد داد. مزیت لیست `ordered` این است که اگر MongoDB در هنگام کار با خطایی

مواجهه شود، بدون انجام باقی اعمال لیست، آن خطا را برمیگرداند. ولی لیست `unordered` در صورت برخورد به خطا، به انجام اعمال دیگر در لیست ادامه خواهد داد. اما مزیت لیست های `unordered` در سرعت بیشتر انجام اعمال است. توجه داشته باشید که متد `bulkWrite()` به صورت پیش فرض اعمال `ordered` را انجام میدهد. برای ایجاد لیست `unordered` باید آپشن `ordered` را در متد، برابر `false` قرار دهید:

```
db.collection.bulkWrite(
  [
    { insertOne : <document> },
    { updateOne : <document> },
    { updateMany : <document> },
    { replaceOne : <document> },
    { deleteOne : <document> },
    { deleteMany : <document> }
  ],
  { ordered : false }
)
```

### • متدهای `bulkWrite()`

هر یک از عملیات زیر میتوانند به عنوان یک آرایه به `bulkWrite()` فرستاده شوند:

#### `insertOne()`

```
db.collection.bulkWrite( [
  { insertOne : { "document" : <document> } }
] )
```

#### `updateOne()` و `updateMany()`

متد `updateOne()` تنها یک داکيومنت را بروز رسانی میکند. اگر چندین داکيومنت با فیلتر موجود همخوانی داشته باشند، این متد اولین داکيومنت را آپدیت میکند.

اما متد `updateMany()` همه داکيومنت هايي که با فيلتر موجود همخواني داشته باشند را بروز رساني ميکند. به یک مثال از هر کدام از اين متدها توجه کنيد:

```
db.collection.bulkWrite( [
  { updateOne :
    {
      "filter" : <document>,
      "update" : <document>,
      "upsert" : <boolean>
    }
  }
] )
```

```
db.collection.bulkWrite( [
  { updateMany :
    {
      "filter" : <document>,
      "update" : <document>,
      "upsert" : <boolean>
    }
  }
] )
```

### **: replaceOne()**

اين متد یک داکيومنت را با داکيومنت ديگري که با فيلتر قيد شده همخواني داشته باشد، جايگزين ميکند:

```
db.collection.bulkWrite([
  { replaceOne :
    {
      "filter" : <document>,
      "replacement" : <document>,
      "upsert" : <boolean>
    }
  }
])
```

```
    }  
  }  
] )
```

### : deleteMany() و deleteOne()

deleteOne() تنها یک داکيومنت و deleteMany() همه داکيومنتهایی که با فیلتر قید شده برابری کند، حذف خواهد کرد:

```
db.collection.bulkWrite([  
  { deleteOne : { "filter" : <document> } }  
] )
```

```
db.collection.bulkWrite([  
  { deleteMany : { "filter" : <document> } }  
] )
```

مثال زیر از این متدها برای عمل bulkWrite() استفاده کرده است:

```
{ "_id" : 1, "char" : "Brisbane", "class" : "monk", "lvl" : 4 },  
{ "_id" : 2, "char" : "Eldon", "class" : "alchemist", "lvl" : 3 },  
{ "_id" : 3, "char" : "Meldane", "class" : "ranger", "lvl" : 3 }
```

```
db.characters.bulkWrite(  
  [  
    { insertOne :  
      {  
        "document" :  
          {  
            "_id" : 4, "char" : "Dithras", "class" : "barbarian",  
            "lvl" : 4  
          }  
        }  
      }  
  ],
```

```
{ insertOne :
  {
    "document" :
    {
      "_id" : 5, "char" : "Taeln", "class" : "fighter", "lvl"
: 3
    }
  },
  { updateOne :
    {
      "filter" : { "char" : "Eldon" },
      "update" : { $set : { "status" : "Critical Injury" } }
    }
  },
  { deleteOne :
    { "filter" : { "char" : "Brisbane" } }
  },
  { replaceOne :
    {
      "filter" : { "char" : "Meldane" },
      "replacement" : { "char" : "Tanys", "class" : "oracle",
"lvl" : 4 }
    }
  }
}
];
```

این اعمال، نتیجه زیر را برمیگرداند:

```
{
  "acknowledged" : true,
  "deletedCount" : 1,
  "insertedCount" : 2,
  "matchedCount" : 2,
  "upsertedCount" : 0,
```

```
"insertedIds" : {
  "0" : 4,
  "1" : 5
},
"upsertedIds" : {

}
}
```

### چارت مطابقت مفاهیم SQL به MongoDB

- جدول زیر، مفاهیم و عبارات مربوط به SQL و معادل آن مفاهیم در MongoDB را نشان میدهد:

SQL Terms/Concepts	MongoDB Terms/Concepts
database	<a href="#">database</a>
table	<a href="#">collection</a>
row	<a href="#">document</a> or <a href="#">BSON document</a>
column	<a href="#">field</a>
index	<a href="#">index</a>
table joins	embedded documents and linking
primary key Specify any unique column or column combination as primary key.	<a href="#">primary key</a> In MongoDB, the primary key is automatically set to the <a href="#">_id</a> field.
aggregation (e.g. group by)	aggregation pipeline

- جدول زیر برخی دیتابیس های اجرایی و معادل آنها در MongoDB را نشان میدهد:

	MongoDB	MySQL	Oracle	Informix	DB2
Database Server	<code>mongod</code>	<code>mysqld</code>	<code>oracle</code>	IDS	DB2 Server
Database Client	<code>mongo</code>	<code>mysql</code>	<code>sqlplus</code>	DB-Access	DB2 Client

- جدول زیر، برخی از توابع و مفاهیم SQL و معادل آنها در MongoDB را نشان میدهد:

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	<code>\$match</code>
GROUP BY	<code>\$group</code>
HAVING	<code>\$match</code>
SELECT	<code>\$project</code>
ORDER BY	<code>\$sort</code>
LIMIT	<code>\$limit</code>
SUM()	<code>\$sum</code>
COUNT()	<code>\$sum</code>
join	<code>\$lookup</code>



در زیر، مثال هایی از [جدول یاد شده](#) نشان داده میشود. توجه کنید که مثال های ذکر شده دارای شرایط زیر هستند:

- مثال های SQL از جدولی با نام `users` فرض شده اند.
- مثال های MongoDB از کالکشنی با نام `users` فرض شده اند که دارای داکيومنت فرضی زیر است:

```
{
  _id: ObjectId("509a8fb2f3f4948bd2f983a0"),
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

: Alter و Create

### SQL Schema Statements

```
CREATE TABLE users (
  id MEDIUMINT NOT NULL
    AUTO_INCREMENT,
  user_id Varchar(30),
  age Number,
  status char(1),
  PRIMARY KEY (id)
)
```

### MongoDB Schema Statements

Implicitly created on first `insert()` operation. The primary key `_id` is automatically added if `_id` field is not specified.

```
db.users.insert( {
  user_id: "abc123",
  age: 55,
  status: "A"
} )
```

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

## SQL Schema Statements

```
ALTER TABLE users
ADD join_date DATETIME
```

## MongoDB Schema Statements

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `update()` operations can add fields to existing documents using the `$set` operator.

```
db.users.update(
  { },
  { $set: { join_date: new Date() } },
  { multi: true }
)
```

```
ALTER TABLE users
DROP COLUMN join_date
```

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `update()` operations can remove fields from documents using the `$unset` operator.

```
db.users.update(
  { },
  { $unset: { join_date: "" } },
  { multi: true }
)
```

```
CREATE INDEX
idx_user_id_asc
```

```
db.users.createIndex( { user_id: 1 } )
```

SQL Schema Statements	MongoDB Schema Statements
<code>ON users (user_id)</code>	
<b>CREATE INDEX</b>  <code>idx_user_id_asc_age_desc</code> <code>ON users (user_id, age</code> <code>DESC)</code>	<code>db.users.createIndex( { user_id: 1, age: -1 } )</code>
<b>DROP TABLE</b> users	<code>db.users.drop()</code>

: Insert

SQL INSERT Statements	MongoDB insert() Statements
<code>INSERT INTO users (user_id,</code> <code>age,</code> <code>status)</code>  <code>VALUES ("bcd001",</code> <code>45,</code> <code>"A")</code>	<code>db.users.insert(</code> <code>{ user_id: "bcd001", age: 45, status: "A" }</code> <code>)</code>

: Select

SQL SELECT Statements	MongoDB find() Statements
<b>SELECT</b> * <b>FROM</b> users	<code>db.users.find()</code>
<b>SELECT</b> id, user_id, status <b>FROM</b> users	<code>db.users.find(     { },     { user_id: 1, status: 1 } )</code>
<b>SELECT</b> user_id, status <b>FROM</b> users	<code>db.users.find(     { },     { user_id: 1, status: 1, _id: 0 } )</code>
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A"	<code>db.users.find(     { status: "A" } )</code>
<b>SELECT</b> user_id, status <b>FROM</b> users <b>WHERE</b> status = "A"	<code>db.users.find(     { status: "A" },     { user_id: 1, status: 1, _id: 0 } )</code>
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status != "A"	<code>db.users.find(     { status: { \$ne: "A" } } )</code>
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A"	<code>db.users.find(     { status: "A",       age: 50 } )</code>

SQL SELECT Statements	MongoDB find() Statements
<b>AND</b> age = 50	)
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A" <b>OR</b> age = 50	db.users.find( { \$or: [ { status: "A" } , { age: 50 } ] } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> age > 25	db.users.find( { age: { \$gt: 25 } } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> age < 25	db.users.find( { age: { \$lt: 25 } } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> age > 25 <b>AND</b> age <= 50	db.users.find( { age: { \$gt: 25, \$lte: 50 } } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> user_id <b>like</b> "%bc%"	db.users.find( { user_id: /bc/ } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> user_id <b>like</b> "bc%"	db.users.find( { user_id: /^bc/ } )

SQL SELECT Statements	MongoDB find() Statements
<pre>SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC</pre>	<pre>db.users.find( { status: "A" } ).sort( { user_id: 1 } )</pre>
<pre>SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC</pre>	<pre>db.users.find( { status: "A" } ).sort( { user_id: -1 } )</pre>
<pre>SELECT COUNT(*) FROM users</pre>	<pre>db.users.count()  or  db.users.find().count()</pre>
<pre>SELECT COUNT(user_id) FROM users</pre>	<pre>db.users.count( { user_id: { \$exists: true } } )  or  db.users.find( { user_id: { \$exists: true } } ) .count()</pre>
<pre>SELECT COUNT(*) FROM users WHERE age &gt; 30</pre>	<pre>db.users.count( { age: { \$gt: 30 } } )  or  db.users.find( { age: { \$gt: 30 } } ).count()</pre>

SQL SELECT Statements	MongoDB find() Statements
<b>SELECT</b> <b>DISTINCT</b> (status) <b>FROM</b> users	<b>db.users.distinct( "status" )</b>
<b>SELECT</b> * <b>FROM</b> users <b>LIMIT</b> 1	<b>db.users.findOne()</b>  or  <b>db.users.find().limit(1)</b>
<b>SELECT</b> * <b>FROM</b> users <b>LIMIT</b> 5 <b>SKIP</b> 10	<b>db.users.find().limit(5).skip(10)</b>
<b>EXPLAIN SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A"	<b>db.users.find( { status: "A" } ).explain()</b>

: Update

SQL Update Statements	MongoDB update() Statements
-----------------------	-----------------------------

## SQL Update Statements

```
UPDATE users
SET status = "C"
WHERE age > 25
```

```
UPDATE users
SET age = age + 3
WHERE status = "A"
```

## MongoDB update() Statements

```
db.users.update (
  { age: { $gt: 25 } },
  { $set: { status: "C" } },
  { multi: true }
)
```

```
db.users.update (
  { status: "A" } ,
  { $inc: { age: 3 } },
  { multi: true }
)
```

: Delete

## SQL Delete Statements

```
DELETE FROM users
WHERE status = "D"
```

```
DELETE FROM users
```

## MongoDB remove() Statements

```
db.users.remove( { status: "D" } )
```

```
db.users.remove({})
```

در زیر مثال هایی از [جدول یاد شده](#) نشان داده میشود. توجه کنید که این مثال ها دارای شرایط زیر میباشند:

- مثال های SQL از دو جدول orders و order\_lineitem که از طریق ستون های orders.id و order\_lineitem.order\_id با هم جوین هستند، فرض شده است.



- مثال های MongoDB از کالکشن orders با داکيومنت زیر فرض شده است:

```
{
  cust_id: "abc123",
  ord_date: ISODate("2012-11-02T17:04:11.102Z"),
  status: 'A',
  price: 50,
  items: [ { sku: "xxx", qty: 25, price: 1 },
            { sku: "yyy", qty: 25, price: 1 } ]
}
```

SQL Example	MongoDB Example	Description
<pre>SELECT COUNT(*) AS count FROM orders</pre>	<pre>db.orders.aggregate( [   {     \$group: {       _id: null,       count: { \$sum: 1 }     }   } ] )</pre>	Count all records from orders
<pre>SELECT SUM(price) AS total FROM orders</pre>	<pre>db.orders.aggregate( [   {     \$group: {       _id: null,       total: { \$sum: "\$price" }     }   } ] )</pre>	Sum the price field from orders
<pre>SELECT cust_id,</pre>	<pre>db.orders.aggregate( [</pre>	For each unique cust_id, sum

SQL Example	MongoDB Example	Description
<pre> SUM(price) AS total FROM orders GROUP BY cust_id </pre>	<pre> {   \$group: {     _id: "\$cust_id",     total: { \$sum: "\$price" }   } } ] ) </pre>	<p>the price field.</p>
<pre> SELECT cust_id, SUM(price) AS total FROM orders GROUP BY cust_id ORDER BY total </pre>	<pre> db.orders.aggregate( [ {   \$group: {     _id: "\$cust_id",     total: { \$sum: "\$price" }   }, { \$sort: { total: 1 } } ] ) </pre>	<p>For each unique cust_id, sum the price field, results sorted by sum.</p>
<pre> SELECT cust_id, ord_date, SUM(price) AS total FROM orders GROUP BY cust_id, ord_date </pre>	<pre> db.orders.aggregate( [ {   \$group: {     _id: {       cust_id: "\$cust_id",       ord_date: {         month: { \$month: "\$ord_date" },         day: { </pre>	<p>For each unique cust_id, ord_date grouping, sum the price field. Excludes the time portion of the date.</p>

SQL Example	MongoDB Example	Description
	<pre> \$dayOfMonth: "\$ord_date" },     year: { \$year: "\$ord_date"}     }     },     total: { \$sum: "\$price" }     } } ] ) </pre>	
<pre> SELECT cust_id,        count(*) FROM orders GROUP BY cust_id HAVING count(*) &gt; 1 </pre>	<pre> db.orders.aggregate( [ {   \$group: {     _id: "\$cust_id",     count: { \$sum: 1 }   },   { \$match: { count: { \$gt: 1 } } } } ] ) </pre>	For cust_id with multiple records, return the cust_id and the corresponding record count.
<pre> SELECT cust_id,        ord_date,        SUM(price) AS total FROM orders GROUP BY cust_id,          ord_date HAVING total &gt; 250 </pre>	<pre> db.orders.aggregate( [ {   \$group: {     _id: {       cust_id: "\$cust_id",       ord_date: {         month: { </pre>	For each unique cust_id, ord_date grouping, sum the price field and return only where the sum is greater than 250. Excludes the time portion of the date.

SQL Example	MongoDB Example	Description
	<pre> \$month: "\$ord_date" },     day: { \$dayOfMonth: "\$ord_date" },     year: { \$year: "\$ord_date"}     }     },     total: { \$sum: "\$price" }     }     },     { \$match: { total: { \$gt: 250 } } } ] ) </pre>	
<pre> SELECT cust_id,       SUM(price) as total FROM orders WHERE status = 'A' GROUP BY cust_id </pre>	<pre> db.orders.aggregate( [   { \$match: { status: 'A' } },   {     \$group: {       _id: "\$cust_id",       total: { \$sum: "\$price" }     }   } ] ) </pre>	For each uniquecust_idwith status A, sum theprice field.
<pre> SELECT cust_id,       SUM(price) as total </pre>	<pre> db.orders.aggregate( [   { \$match: { status: 'A' } }, </pre>	For each uniquecust_idwith status A, sum theprice field and return only where the sum is

SQL Example	MongoDB Example	Description
<pre>FROM orders WHERE status = 'A' GROUP BY cust_id HAVING total &gt; 250</pre>	<pre>{   \$group: {     _id: "\$cust_id",     total: { \$sum: "\$price" }   } }, { \$match: { total: { \$gt: 250 } } } ] )</pre>	<p>greater than 250.</p>
<pre>SELECT cust_id,       SUM(li.qty) as qty FROM orders o, order_lineitem li WHERE li.order_id = o.id GROUP BY cust_id</pre>	<pre>db.orders.aggregate( [   { \$unwind: "\$items" },   {     \$group: {       _id: "\$cust_id",       qty: { \$sum: "\$items.qty" }     }   } ] )</pre>	<p>For each uniquecust_id, sum the corresponding line item qtyfields associated with the orders.</p>
<pre>SELECT COUNT(*) FROM (SELECT cust_id, ord_date FROM orders GROUP BY cust_id,</pre>	<pre>db.orders.aggregate( [   {     \$group: {       _id: {         cust_id: "\$cust_id",         ord_date: {           month: {</pre>	<p>Count the number of distinctcust_id,ord_dategroup ings. Excludes the time portion of the date.</p>

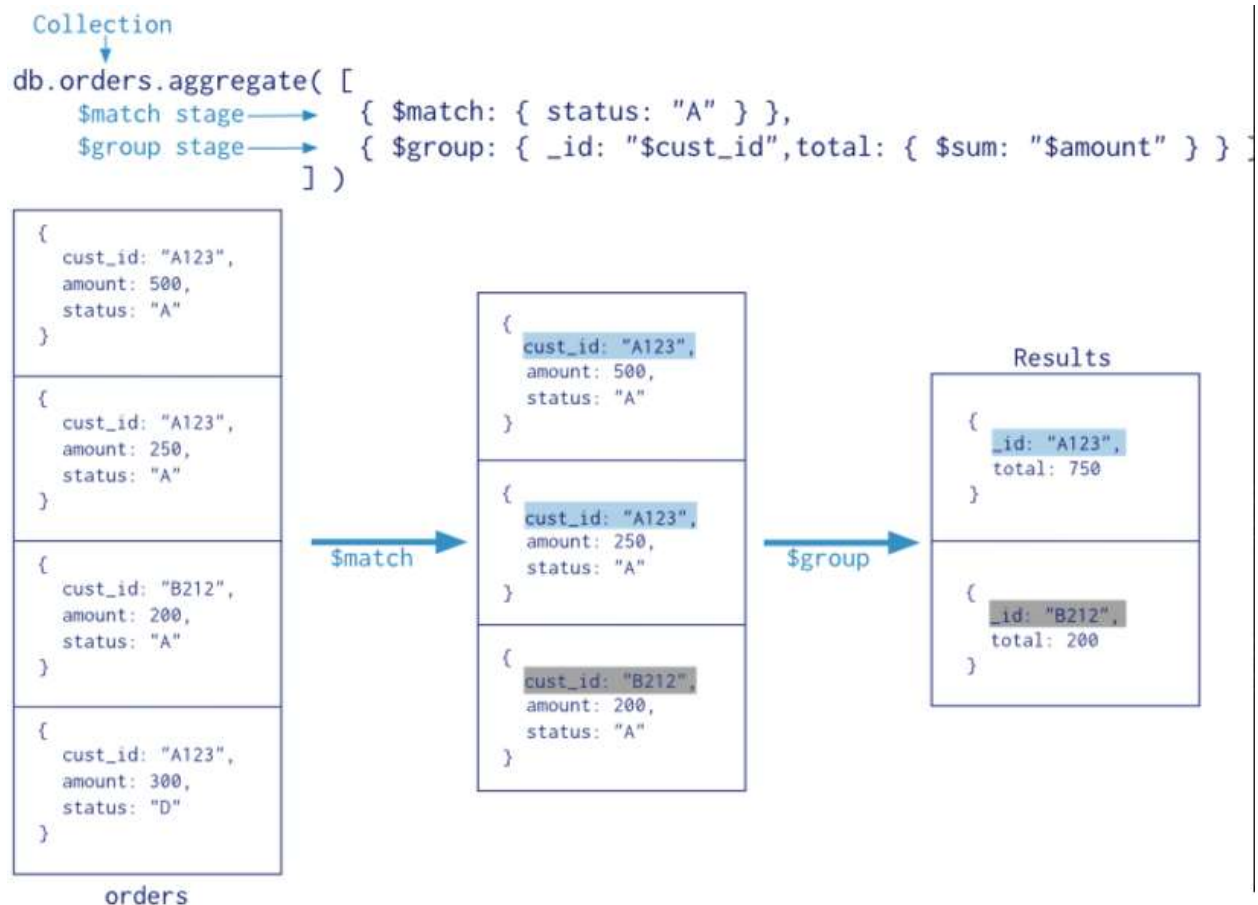
SQL Example	MongoDB Example	Description
<pre>ord_date)     as DerivedTable</pre>	<pre>\$month: "\$ord_date" },     day: { \$dayOfMonth: "\$ord_date" },     year: { \$year: "\$ord_date"}     }     } }, {     \$group: {         _id: null,         count: { \$sum: 1 }     } } ] )</pre>	

## Aggregation

عملیات aggregation اعمالی هستند که بر روی داده ها پردازش خاصی را انجام میدهند و نتایج این پردازش را برمیگردانند. این عملیات میتوانند مقادیری از چند داکيومنت را گروه بندی کرده و پردازش های مختلفی را بر روی این داده های گروه بندی شده انجام بدهند تا یک جواب واحد از این پردازش بگیرند. MongoDB سه روش برای انجام aggregation فراهم کرده است:

- aggregation pipeline
- map-reduce function
- single purpose aggregation methods

**aggregation pipeline** یک فریم ورک مدل شده بر روی مفهوم مسیرهای پردازش داده است. در این روش داکيومنت ها به یک مسیر multi-stage وارد میشوند که این مسیر داکيومنت ها را به نتایج aggregate شده (تجميع شده) هدایت میکند. برای فهم بهتر موضوع به مثال زیر توجه کنید:



در این مثال تعدادی داکيومنت از کالکشن `orders` وارد مسير `aggregation` میشود و این مسير با متد `$match` آنها را فیلتر میکند و در انتها با متد `$group` خروجی ها را دسته بندی میکند.

در واقع بیشتر `pipeline` های پایه در این روش، فیلترهایی را شبیه به `query` ها برای هدایت داکيومنت ها به سمت خروجی مورد نظر فراهم میکنند.

`pipeline` های دیگر، ابزار هایی را برای گروه بندی یا طبقه بندی داکيومنت ها با یک یا چند فیلد مشخص فراهم میکنند. بعلاوه `pipeline stage` ها میتوانند از عملیات مختلفی مانند محاسبه های ریاضی یا اتصال رشته ها برای تسک ها استفاده کنند. این `pipeline` ها متدهای بسیار خوبی برای تجميع داده ها در MongoDB دارا میباشند.

`aggregation pipeline` میتواند از ایندکس ها برای بهبود نمایش خروجی ها استفاده کند. بعلاوه این روش دارای یک وجهه بهینه سازی درونی است. برای مشاهده انواع مختلف این `pipeline optimization` ها به این صفحه مراجعه کنید: <https://docs.mongodb.com/manual/core/aggregation-pipeline-optimization>

متد استفاده از `aggrigation pipeline` به صورت زیر است:

```
db.collection.aggregate(pipeline,options)
```

در این متد، بخش `pipeline` شامل لیست زیر میباشد:

Name	Description
<code>\$collStats</code>	Returns statistics regarding a collection or view.
<code>\$project</code>	Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.
<code>\$match</code>	Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. <code>\$match</code> uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no



Name	Description
	match).
<code>\$redact</code>	Reshapes each document in the stream by restricting the content for each document based on information stored in the documents themselves. Incorporates the functionality of <code>\$project</code> and <code>\$match</code> . Can be used to implement field level redaction. For each input document, outputs either one or zero documents.
<code>\$limit</code>	Passes the first $n$ documents unmodified to the pipeline where $n$ is the specified limit. For each input document, outputs either one document (for the first $n$ documents) or zero documents (after the first $n$ documents).
<code>\$skip</code>	Skips the first $n$ documents where $n$ is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first $n$ documents) or one document (if after the first $n$ documents).
<code>\$unwind</code>	Deconstructs an array field from the input documents to output a document for <i>each</i> element. Each output document replaces the array with an element value. For each input document, outputs $n$ documents where $n$ is the number of array elements and can be zero for an empty array.
<code>\$group</code>	Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.
<code>\$sample</code>	Randomly selects the specified number of documents from its input.
<code>\$sort</code>	Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.

Name	Description
<code>\$geoNear</code>	Returns an ordered stream of documents based on the proximity to a geospatial point. Incorporates the functionality of <code>\$match</code> , <code>\$sort</code> , and <code>\$limit</code> for geospatial data. The output documents include an additional distance field and can include a location identifier field.
<code>\$lookup</code>	Performs a left outer join to another collection in the <i>same</i> database to filter in documents from the “joined” collection for processing.
<code>\$out</code>	Writes the resulting documents of the aggregation pipeline to a collection. To use the <code>\$out</code> stage, it must be the last stage in the pipeline.
<code>\$indexStats</code>	Returns statistics regarding the use of each index for the collection.
<code>\$facet</code>	Processes multiple aggregation pipelines within a single stage on the same set of input documents. Enables the creation of multi-faceted aggregations capable of characterizing data across multiple dimensions, or facets, in a single stage.
<code>\$bucket</code>	Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.
<code>\$bucketAuto</code>	Categorizes incoming documents into a specific number of groups, called buckets, based on a specified expression. Bucket boundaries are automatically determined in an attempt to evenly distribute the documents into the specified number of buckets.
<code>\$sortByCount</code>	Groups incoming documents based on the value of a specified expression, then computes the count of documents in each distinct group.
<code>\$addFields</code>	Adds new fields to documents. Outputs documents that contain all existing fields from the input documents and newly added fields.

Name	Description
<code>\$replaceRoot</code>	Replaces a document with the specified embedded document. The operation replaces all existing fields in the input document, including the <code>_id</code> field. Specify a document embedded in the input document to promote the embedded document to the top level.
<code>\$count</code>	Returns a count of the number of documents at this stage of the aggregation pipeline.
<code>\$graphLookup</code>	Performs a recursive search on a collection. To each output document, adds a new array field that contains the traversal results of the recursive search for that document.

**map-reduce operation** ها دارای دو وجهه هستند: یک `map stage` که هر داکيومنت را پردازش میکند و میتواند یک یا چند آبجکت برای هر داکيومنت منتشر کند، و یک وجهه `reduce` که خروجی های `map` را با هم ترکیب میکند. بطور اختیاری، `map-reduce` میتواند دارای یک `finalize stage` برای ساختن تغییرات نهایی در خروجی باشد.

`map-reduce` از توابع `javaScript` برای انجام اعمال `map` و `reduce` استفاده میکند. به طور عمومی زمانی که `javaScript` کاربر دارای انعطاف پذیری کافی باشد، `map-reduce` از روش قبلی ( `aggregation pipeline` ) پیچیده تر بوده و بهره وری کمتری دارد. برای فهم بهتر این روش، به مثال زیر توجه نمایید:

```

Collection
↓
Db.orders.mapReduce(
    map    —————> Function() { emit( this.cust_id, this.amount ); },
    reduce —————> Function(key, values) {return array.sum( values ) },
    {

```

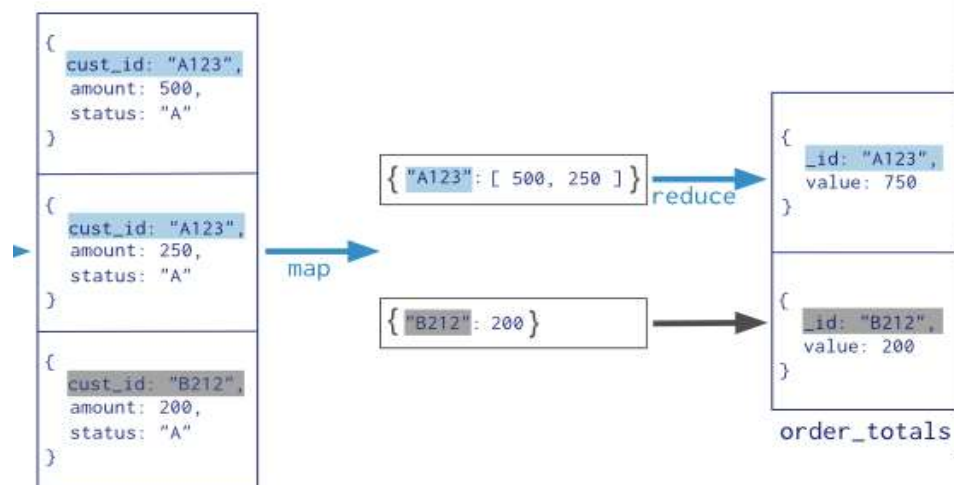
query → Query: { status:"A" },

output → Out: "order\_totals"

}

)


```
function() { emit( this.cust_id, this.amount ); },
function(key, values) { return Array.sum( values ) },
{
  query: { status: "A" },
  out: "order_totals"
}
```



**single purpose aggregation operation** : برای این منظور MongoDB دو متد زیر را ایجاد کرده

است:

• **db.collection.count(query,options)**

**query:** از نوع داکيومنت بوده و در آن ضوابط مشخص میشوند.  
**options:** از نوع داکيومنت بوده و برای اصلاح شمارش بکار میرود.  
 برای مثال:

```
db.orders.count( { ord_dt: { $gt: new Date('01/01/2012') } } )
```

• **db.collection.distinct(field,query,options)**

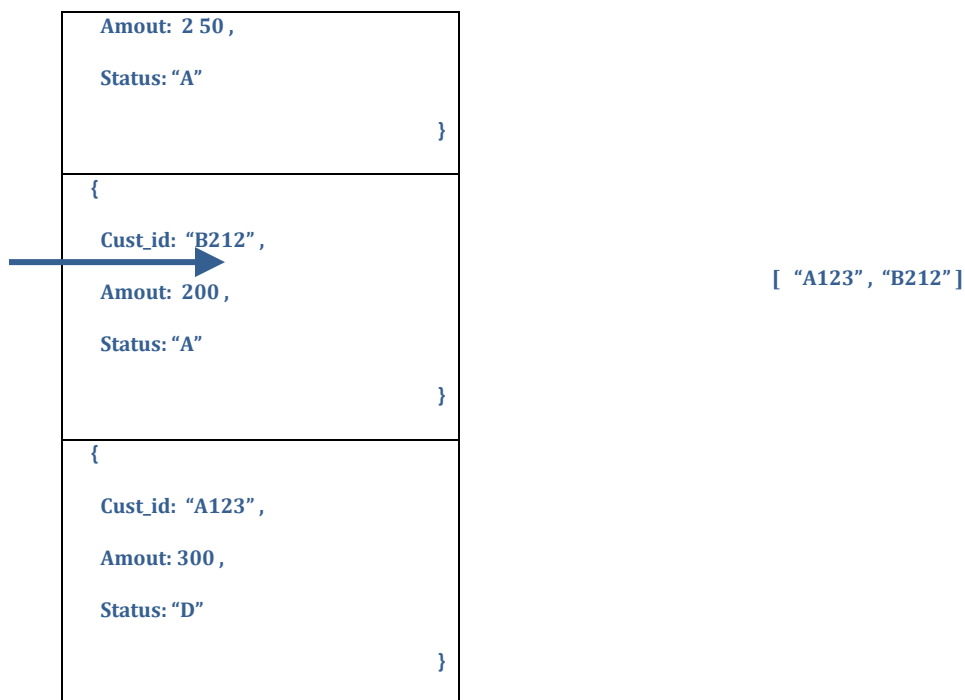
**field:** از نوع رشته ای بوده و برای تعیین اینکه کدام مقادیر متمایز باید برگردانده شوند استفاده میشود  
**query:** از نوع داکيومنت بوده و داکيومنت هایی را که مقادیر متمایز باید از آنها برگردانده شوند را مشخص میکند  
**options:** از نوع داکيومنت بوده و آپشن های اضافه ای را مشخص میکند.  
 برای مثال:

```
db.inventory.distinct( "item.sku", { dept: "A" } )
```

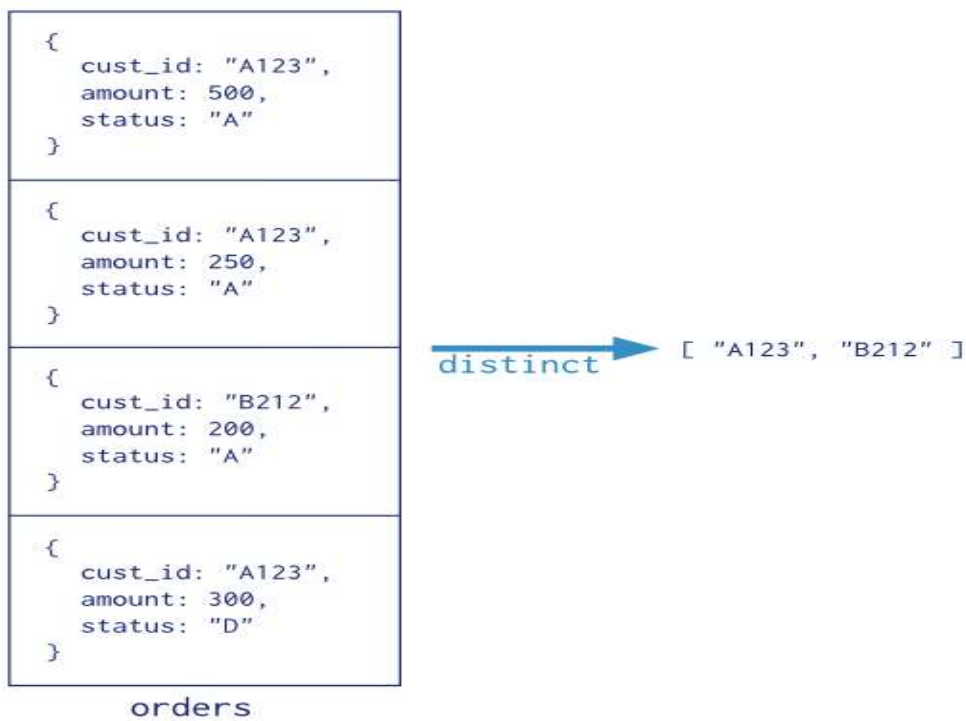
هر دوی این اعمال داکيومنت ها را فقط از یک کالکشن تجميع میکنند. در مورد این دو متد باید گفت که در مقایسه با دو روش قبلی (aggregation pipline و map-reduce function)، این روش انعطاف پذیری و قابلیت اطمینان کمتری دارد. برای فهم بهتر این روش به مثال زیر توجه کنید:

Collection  
 ↓  
**db.orders.distinct( "cust\_id" )**

<pre>{   Cust_id: "A123",   Amout: 500,   Status: "A" }</pre>
<pre>{   Cust_id: "A123",</pre>



Collection  
↓  
`db.orders.distinct( "cust_id" )`





## Text Search

mongodb از کوئری هایی که برای اجرای جستجو در رشته های متنی استفاده میشود، پشتیبانی میکند؛ برای جستجوی متنی، از Text index و عملگر \$text استفاده میشود.

-- مثال های ذکر شده در این قسمت همگی بر روی کالکشن فرضی زیر اجرا میشوند:

```
db.stores.insert(
  [
    { _id: 1, name: "Java Hut", description: "Coffee and cakes" },
    { _id: 2, name: "Burger Buns", description: "Gourmet hamburgers" },
    { _id: 3, name: "Coffee Shop", description: "Just coffee" },
    { _id: 4, name: "Clothes Clothes Clothes", description: "Discount clothing" },
    { _id: 5, name: "Java Shopping", description: "Indonesian goods" }
  ]
)
```

## Text Indexes

تکست ایندکس ها برای پشتیبانی از کوئری های جستجوی متنی در ذخایر رشته ای فراهم شده اند. برای اجرای کوئری های جستجوی متنی، شما باید یک text index بر روی کالکشن خود داشته باشید. هر کالکشن میتواند تنها یک تکست ایندکس داشته باشد، اما آن ایندکس میتواند شامل چندین فیلد بشود.

برای ساخت text index بر روی کالکشن از نمونه کوئری زیر استفاده میشود:

```
db.stores.createIndex( { name: "text", description: "text" } )
```

با این کوئری از این پس میتوانیم جستجوهای متنی را بر روی فیلدهای name و description انجام دهیم.

## Text Search Operators



-- از عملگر `$text` برای اجرای جستجوهای متنی بر روی یک کالکشن با `text index` استفاده میشود. برای مثال میتوان از کوئری زیر برای پیدا کردن تمام ذخایری که شامل عبارات "coffee" و "shop" و "java" میباشد استفاده کرد:

```
db.stores.find( { $text: { $search: "java coffee shop" } } )
```

و یا برای پیدا کردن مواردی که شامل عبارات "java" یا "coffee shop" میشود از کوئری زیر استفاده میکنیم:

```
db.stores.find( { $text: { $search: "java \"coffee shop\"" } } )
```

اما برای پیدا کردن مواردی که شامل عبارات "java" یا "shop" بغیر از "coffee" هستند از کوئری زیر استفاده میشود:

```
db.stores.find( { $text: { $search: "java shop -coffee" } } )
```

-- تکست سرچ در `Aggregation Pipeline` از طریق عملگر `$text` در `$meta` فراهم میشود. اما در این مورد محدودیت های زیر وجود دارد:

- استیج `$meta` ای که شامل عملگر `$text` شود، حتما باید اولین استیج در `pipeline` باشد.
- یک عملگر متنی تنها یکبار میتواند در استیج اتفاق بیافتد
- عملگر متنی نمیتواند در `$or` و `$not` ظاهر شود.
- جستجوی متنی بطور پیشفرض نتایج را نامرتب برمیگرداند. برای مرتب سازی باید از `$meta` در استیج `$sort` استفاده کرد.

عملگر `$text` به هر داکيومنتی که شامل عبارات جستجو شده در فیلدهای ایندکس گذاری شده باشد، امتیازی اختصاص میدهد. این امتیاز میتواند بخشی از مشخصات `$sort Pipeline` باشد. عبارت `{ "textScore": $meta: { اطلاعاتی را برای پردازش عملیات $text فراهم میکند.`

`metadata` فقط بعد از استیج `$match` که شامل عملگر `$text` باشد، قابل استفاده است.

گفته شد **mongodb** نتایج را به طور پیش فرض به صورت نامرتب برمیگرداند. برای مرتب کردن نتایج به ترتیب امتیاز ارتباط (**relevance score**) باید از عملگر **\$meta** بر روی فیلد **textScore** استفاده کرد:

برای مثال برای سفارش لیستی از کافی شاپ ها بر اساس **relevance score** از کوئری زیر استفاده میشود.

```
db.stores.find(
  { $text: { $search: "coffee shop cake" } },
  { score: { $meta: "textScore" } }
).sort( { score: { $meta: "textScore" } } )
```

مثال زیر، یک **text index** بر روی فیلد **subject** کالکشن **articles** ایجاد میکند:

```
db.articles.createIndex( { subject: "text" } )
```

اکنون در **aggregation** زیر عبارت **cake** در استیج **\$match** جستجو شده و **total view** هایی برای داکيومنت های هماهنگ در استیج **\$group** محاسبه میشود:

```
db.articles.aggregate(
  [
    { $match: { $text: { $search: "cake" } } },
    { $group: { _id: null, views: { $sum: "$views" } } }
  ]
)
```

اکنون نتایج بدست آمده با امتیاز **text search** مرتب سازی شده و برگردانده میشود:

```
db.articles.aggregate(
  [
    { $match: { $text: { $search: "cake tea" } } },
    { $sort: { score: { $meta: "textScore" } } },
    { $project: { title: 1, _id: 0 } }
  ]
)
```

## Data Models

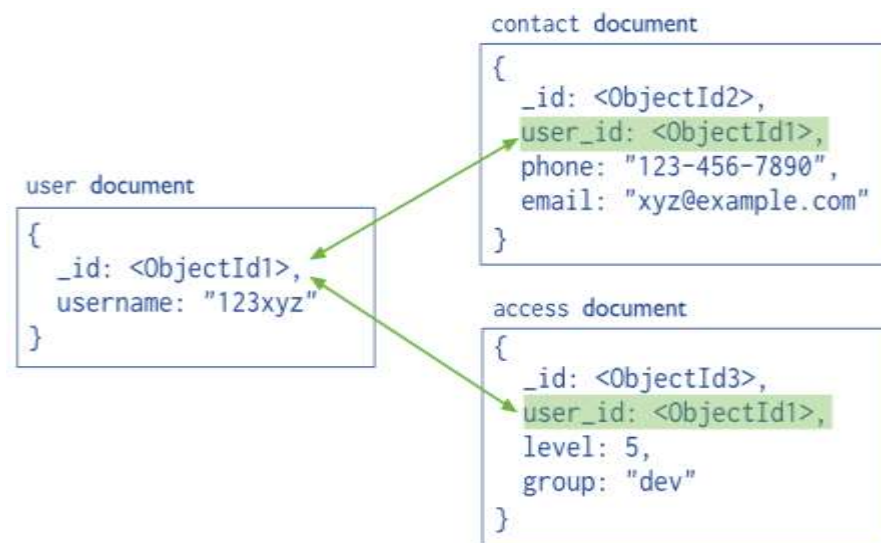
داده ها در MongoDB دارای طرح و الگوی انعطاف پذیری هستند. برخلاف دیتابیس های SQL که باید شمای جداول را قبل از insert کردن داده ها تعریف و اعلام کنیم، کالکشن های MongoDB به ساختار داکيومنت وابسته نیستند.

### :document structure

key decision در طراحی مدل های داده برای اپلیکیشن های MongoDB، حول محور ساختار داکيومنت ها و اینکه اپلیکیشن ها چگونه روابط بین داده ها را نشان میدهند، میگردد. دو ابزار برای اپلیکیشن ها وجود دارد که روابط سه گانه را نشان دهند: references و embeded documents

#### References •

رفرنس ها، رابطه بین داده ها را به وسیله links یا references از یک داکيومنت به داکيومنتی دیگر ذخیره میکنند. اپلیکیشن ها میتوانند از این رفرنس ها برای دسترسی به داده های مربوطه استفاده کنند. این مدل داده به طور گسترده ای نرمالیزه شده است.



#### Embeded Data •

embeded data، رابطه بین داده ها را با قرار دادن و مرتب سازی داده های مرتبط، در ساختار یک داکيومنت واحد ایجاد میکند. این مدل داده ی نرمالیزه نشده، به اپلیکیشن ها اجازه میدهد داده های مربوطه را بازیابی کنند و آنها را بوسیله عملگرهای موجود دستکاری کنند.



### Atomicity of Write Operations

در مونگو دی بی عملیات write در سطح داکيومنت، atomic است. این به آن معناست که هیچ عملیات write ای نمیتواند بر روی بیش از یک داکيومنت یا کالکشن به طور atomically اثر بگذارد. مدل‌های داده ی نرمالیزه نشده، همه داده های مرتبط را در یک داکيومنت واحد ترکیب میکنند

### Document Growth

بعضی آپدیت ها، مانند وارد کردن یک ایمان در آرایه یا اضافه کردن یک فیلد، سایز داکيومنت را افزایش میدهد. در موتورهای ذخیره سازی MMAPv1 اگر سایز داکيومنت از حد مشخص شده تجاوز کند، MongoDB داکيومنت مربوطه را در دیسک جابجا میکند.

### Data Use & Performance

هنگام طراحی مدل داده ها باید تصمیم بگیریم که اپلیکیشن ها چگونه از دیتابیس ما استفاده میکنند؟ برای مثال، اگر اپلیکیشن ما تنها از داده های وارد شده جدید استفاده میکند، باید از Capped Collection استفاده کنیم. یا اگر نیازهای اپلیکیشن ما بر اساس استفاده از عملگرها بر روی کالکشن هاست، باید از ایندکس ها برای افزایش کارایی در کوئری های رایج استفاده کنیم.

### :Document Validation

MongoDB امکان معتبر سازی داکيومنت ها را حین insert و update داده ها فراهم میکند. این امر به وسیله عملگر validator و بر اساس قوانین per-collection امکان پذیر میباشد.

اضافه کردن اعتبارسنج داکيومنت به دو طریق امکان پذیر است. اول در هنگام ساختن یک کالکشن و استفاده از آپشن validator، و دوم استفاده از کامند collMod به همراه آپشن validator در یک کالکشن موجود.

در مثال زیر یک کالکشن جدید به نام contacts به همراه یک validator ایجاد شده است:

```
db.createCollection( "contacts",
  { validator: { $or:
    [
      { phone: { $type: "string" } },
      { email: { $regex: /@mongodb\.com$/ } },
      { status: { $in: [ "Unknown", "Incomplete" ] } }
    ]
  }
} )
```

در این مثال، کالکشن contacts دارای شروطی برای insert کردن داکيومنت هاست. به زبان ساده تر، از این به بعد نمیتوان هر نوع داکيومنتی را بدون رعایت قوانین اعمال شده، وارد کالکشن کرد؛ داکيومنت های ورودی در صورت داشتن فیلدهایی نظیر phone، email و status حتما باید قوانین ذکر شده را رعایت کنند.

همچنین MongoDB دو آپشن validationLevel و validationAction را در اختیار کاربران قرار داده است.

validationLevel مشخص میکند که MongoDB چگونه قوانین اعتبارسنج را هنگام آپدیت یک داکيومنت به کار میبرد. اما validationAction مشخص میکند که MongoDB در هنگام نقض قوانین validation باید مانع از insert و update شود یا فقط به نوشتن هشدار در log بسنده کرده و اجازه insert و update را به داکيومنت ناقض شروط بدهد؟ در ادامه به توضیح بیشتری از این آپشن ها میپردازیم.

-- توجه داشته باشید که شروط validation هنگام insert و update اتفاق می افتد. وقتی ما validation را به کالکشن خود اضافه کنیم، شروط تنها شامل داکيومنت های جدید شده و داکيومنت های موجود در آن محدود به آن قوانین نمیشوند.

اما با استفاده از آپشن **validationLevel** میتوان مشخص کرد که MongoDB در قبال داکيومنت های فعلی که از قبل در کالکشن موجود بودند چگونه عمل کند؟ این آپشن به طور پیشفرض در حالت **strict** قرار دارد و MongoDB قوانین **validation** را برای همه **insert** ها و **update** ها به کار میگیرد. اما با تنظیم آن به حالت **moderate**، مونگو دی بی شروط **validation** را هنگام **insert** و **update** به داکيومنت هایی اعمال میکند که همه شاخص های اعتبارسنج را بر طرف کنند. در حقیقت در حالت **moderate**، در زمان آپدیت داده ها، داکيومنت هایی که همه شروط را نداشته باشند شامل آپدیت نمیشوند.

برای مثال به داکيومنت های زیر در کالکشن **contacts** توجه کنید:

```
{
  "_id": "125876",
  "name": "Anne",
  "phone": "+1 555 123 456",
  "city": "London",
  "status": "Complete"
},
{
  "_id": "860000",
  "name": "Ivan",
  "city": "Vancouver"
}
```

اکنون با کامند زیر، یک **validator** را به **contacts** اضافه میکنیم:

```
db.runCommand( {
  collMod: "contacts",
  validator: { $or: [ { phone: { $exists: true } }, { email: { $exists: true } } ] },
  validationLevel: "moderate"
} )
```

در اینجا **validationLevel** بر روی **moderate** تنظیم شده است. اکنون اگر ما بخواهیم داکيومنت با آی دی 125876 را آپدیت کنیم، MongoDB تا زمانی که شروط موجود در اعتبارسنج، در این داکيومنت رعایت شود آن را آپدیت میکند. ولی در همین هنگام داکيومنت با آی دی 860000 اصلاً به روز رسانی نمیشود چون هیچ کدام از قوانین **validation** در آن رعایت نشده است.

برای غیر فعال کردن validation، باید validationLevel را در حالت off قرار دهیم.

آپشن **validationAction** بطور پیش فرض بر روی گزینه error قرار دارد. این حالت باعث میشود MongoDB تمام insert ها و update هایی که شروط validation را رعایت نمیکنند را رد کند. اما وقتی این آپشن را بر روی گزینه warn قرار دهیم، MongoDB همه تخطی ها را در log ثبت میکند اما اجازه پردازش insert و update ها را میدهد.

مثال زیر کالکشن contacts را با اعتبارسازی با قوانین زیر ایجاد میکند:

- فیلد phone باید از نوع رشته ای باشد
- فیلد email باید با شرط ذکر شده در کد مطابقت داشته باشد
- فیلد status باید شامل یکی از دو گزینه Unknown یا Incomplete باشد.

```
db.createCollection( "contacts",
  {
    validator: { $or:
      [
        { phone: { $type: "string" } },
        { email: { $regex: /@mongodb\.com$/ } },
        { status: { $in: [ "Unknown", "Incomplete" ] } }
      ]
    },
    validationAction: "warn"
  }
)
```

با وجود validator قرار داده شده، کامند زیر قوانین موجود را نقض میکند؛ اما تا زمانی که validationAction در حالت warn باشد، این نقض در log ذخیره شده و عملیات insert با موفقیت انجام میشود.

```
db.contacts.insert( { name: "Amanda", status: "Updated" } )
```

در فایل log نام کامل تبیل اسپیس و کالکشنی که داکيومنت در آن ناقض شروط validator است به همراه زمان ثبت آن نوشته میشود:

```
2015-10-15T11:20:44.260-0400 W STORAGE [conn3] Document would fail
validation collection: example.contacts doc: { _id:
ObjectId('561fc44c067a5d85b96274e4'), name: "Amanda", status: "Updated" }
```

تذکر: (1) validator را نمیتوان در دیتابیس های Admin، Local و Config تعریف کرد.

(2) validator را نمیتوان برای کالکشن های \*system تعریف نمود.

گفتیم که MongoDB شمای انعطاف پذیری دارد و کالکشن ها محدود به ساختار داکيومنت ها نیستند. در واقع با تشخیص و تصمیم ما، data model میتواند بر روی اپلیکیشن ها تاثیر گذاشته و کارایی آنها را تحت تاثیر قرار دهد.

در ادامه به data model هایی میپردازیم که از داکيومنت های embeded استفاده میکنند تا ارتباط بین داده ها را نشان دهد.

**الگوها:** به مثالهای زیر که روابط بین patron و address را ترسیم میکند توجه کنید:

**رابطه one-to-one با داده های embeded:** در این مثال که رابطه بین داده های patron و address را نشان میدهد، فیلد address متعلق به patron است. در data model نرمال، داکيومنت address دارای یک reference یا مرجع به داکيومنت patron است.

```
{
  _id: "joe",
  name: "Joe Bookreader"
}

{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```



در اینجا اگر داده های address مکررا با اطلاعات name بازیابی شود، اپلیکیشن ما نیاز به چندین query برای برطرف کردن منبع دارد. اما data model بهتر، قرار دادن داده های address درون داده های patron خواهد بود:

```
{
  _id: "joe",
  name: "Joe Bookreader",
  address: {
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
  }
}
```

با embedded data model اپلیکیشن ما میتواند اطلاعات کاملی از patron را تنها با یک کوئری بازیابی کند.

رابطه **one-to-many** با داده های **embedded**: در این مثال که رابطه بین داده های patron و address را نشان میدهد، patron دارای چندین ماهیت address است. در data model نرمال، داکيومنت address دارای یک reference یا مرجع به داکيومنت patron است.

```
{
  _id: "joe",
  name: "Joe Bookreader"
}

{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```

```
{
  patron_id: "joe",
  street: "1 Some Other Street",
  city: "Boston",
  state: "MA",
  zip: "12345"
}
```

در اینجا اگر داده های address مکررا با اطلاعات name بازیابی شود، اپلیکیشن ما نیاز به چندین query برای برطرف کردن منبع دارد. اما data model بهینه تر، قرار دادن داده های address درون داده های patron خواهد بود:

```
{
  _id: "joe",
  name: "Joe Bookreader",
  addresses: [
    {
      street: "123 Fake Street",
      city: "Faketon",
      state: "MA",
      zip: "12345"
    },
    {
      street: "1 Some Other Street",
      city: "Boston",
      state: "MA",
      zip: "12345"
    }
  ]
}
```

با embeded data model اپلیکیشن ما میتواند اطلاعات کاملی از patron را تنها با یک کوئری بازیابی کند.

رابطه **one-to-many** با داده های **references**: در این data model از داکيومنت های دارای reference استفاده شده است. به مثالهای زیر که روابط بین publisher و book را ترسیم میکند و مزیت های referencing در مقابل embedding نشان میدهد، توجه کنید:

```
{
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher: {
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
  }
}

{
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher: {
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
  }
}
```

در اینجا مشاهده میکنیم که عمل embedding داکيومنت publisher درون داکيومنت book، منجر به تکرار داده های publisher میشود. برای جلوگیری از این تکرار باید از reference ها و نگهداری اطلاعات publisher در یک کالکشن جداگانه استفاده کرد.

```
{
```

```
name: "O'Reilly Media",
founded: 1980,
location: "CA",
books: [123456789, 234567890, ...]
}

{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English"
}

{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English"
}
```

برای جلوگیری از ناپایداری و رشد آرایه ها، باید publisher reference را درون داکيومنت book ذخیره کنیم:

```
{
  _id: "oreilly",
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA"
}

{
  _id: 123456789,
```

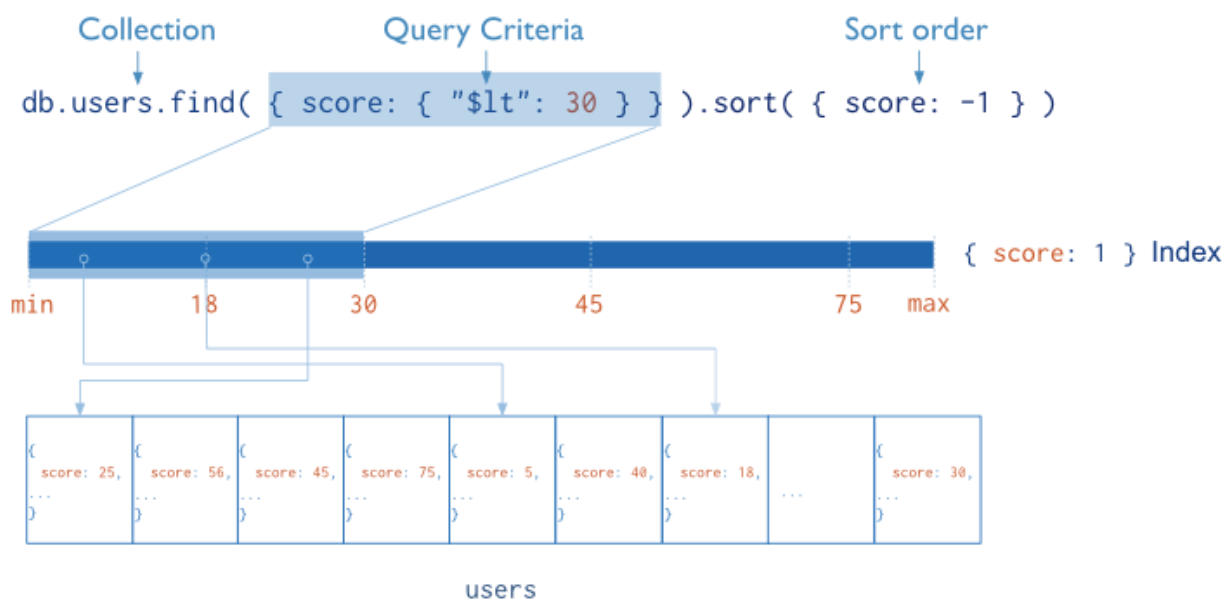
```
title: "MongoDB: The Definitive Guide",
author: [ "Kristina Chodorow", "Mike Dirolf" ],
published_date: ISODate("2010-09-24"),
pages: 216,
language: "English",
publisher_id: "oreilly"
}

{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher_id: "oreilly"
}
```

## Indexes

برای جستجوی موثر در MongoDB باید از ایندکس استفاده نمود. بدون ایندکس ها MongoDB تمامی داکيومنت های کوئری اجرا شده را برای رسیدن به مقدار مورد نظر اسکن می کند. در صورتی که ایندکس مناسبی برای کوئری وجود داشته باشد MongoDB با استفاده از ایندکس داکيومنت های کمتری را جستجو می کند.

شکل زیر مثال روشنی برای select و مرتب کردن داکيومنت هایی که با ایندکس همخوانی دارد می باشد.



اساساً ایندکس در MongoDB مانند ایندکس در سایر سیستم ها می باشد. در MongoDB می توان روی collection ها و یا حتی زیر مجموعه های آنها ایندکس ایجاد کرد.

### ایندکس پیش فرض \_id

MongoDB زمانی که یک collection ایجاد می شود به صورت پیش فرض اندکس unique روی فیلد \_id می سازد. این ایندکس باعث می شود داکيومنتی با \_id تکراری در collection وارد نشود. این ایندکس drop نمی شود.

### ساخت ایندکس

برای ساخت ایندکس از `db.collection.createIndex()` استفاده می شود دستور ساخت ان به شرح زیر می باشد

```
db.collection.createIndex( <key and index type specification>, <options> )
```

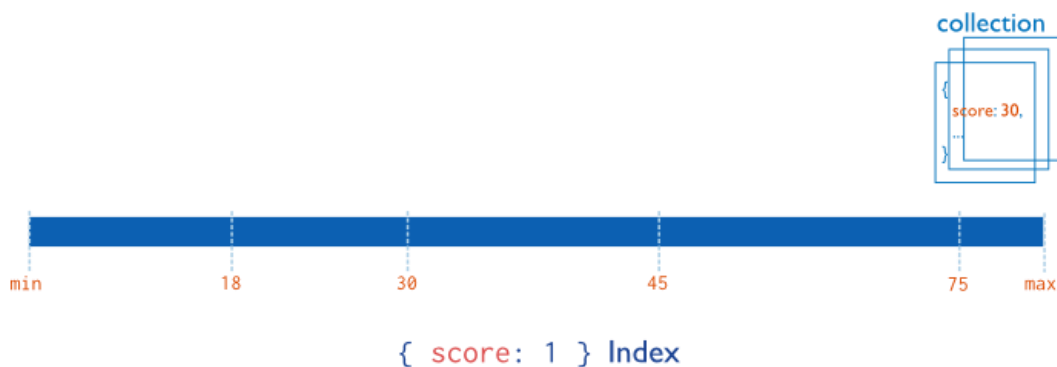
از این متد تنها برای ساخت ایندکس استفاده می شود در صورتی که ایندکس مشابهی وجود نداشته باشد.

### انواع ایندکس

Mongodb از انواع مختلف ایندکس ها پشتیبانی می کند

#### تک فیلد

مانند ایندکس `_id` می باشد. امکان ساخت ایندکس روی هر فیلدی به صورت `Asc/Desc` توسط کاربران وجود دارد.



دراین نوع ایندکس (ایندکس روی تک فیلد) زیاد اهمیتی از دید Mongodb ندارد زیرا mongodb از هر دو جهت ایندکس را پیمایش کند.

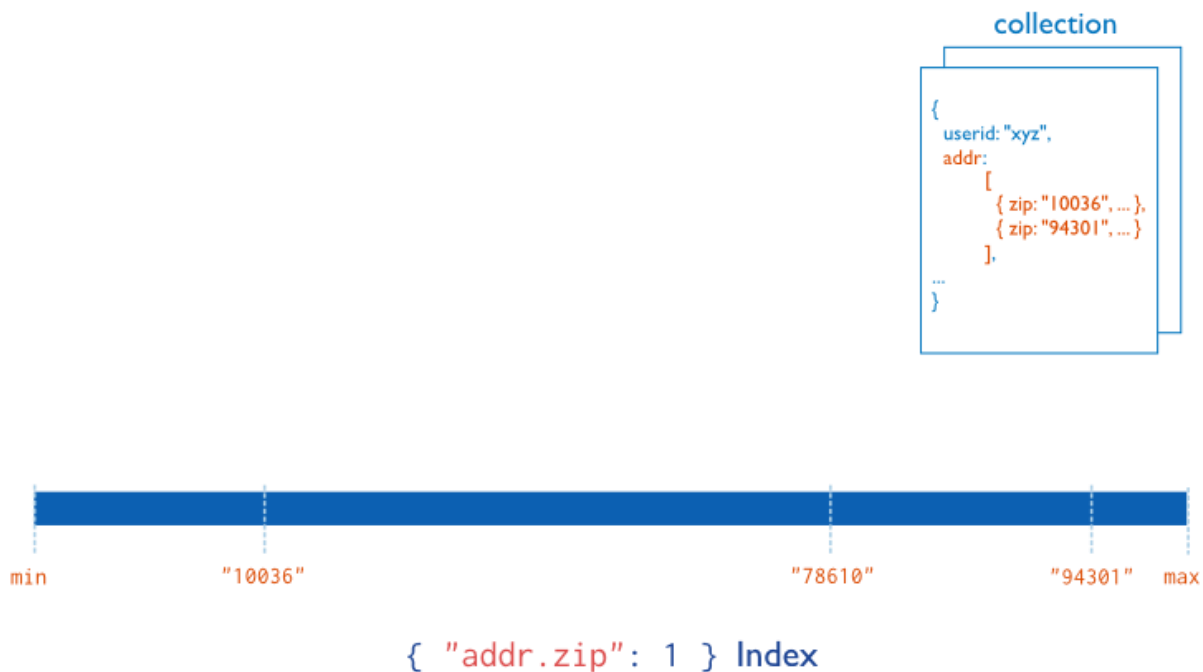
#### ایندکس مرکب

Mongodb از ایندکس هایی شامل چند فیلد که توسط کاربر ساخته می شود پشتیبانی می کند.

مرتب سازی در ایندکس های چند فیلدی با مفهوم می باشد به عنوان مثال اگر ایندکس ترکیبی شامل فیلد های {user\_id: 1, score: -1} باشد ایندکس ابتدا با userid و بعد هر userid با score مرتب می شود.

### ایندکس Multikey

Mongodb از multikey ایندکس برای محتویات داخل یک کالکشن استفاده می کند. اگر ایندکسی روی یکی از فیلد های داخل آرایه ایجاد کنیم Mongodb برای هریک از المان ها ایندکس جداگانه ای می سازد



### ایندکس Geospatial

برای پشتیبانی از کوئری هایی که روی دادهای مکانی (geospatial) اجرا می شوند Mongodb دو نوع ایندکس را پشتیبانی می کند ، 2d ایندکس که از هندسه دوجهی و یا از 2dsphere ایندکس که از هندسه کروی استفاده می کند برای بازگرداندن نتیجه استفاده می شود.



### ایندکس Tesxt

Mongodb از ایندکس text را برای جستجوی متن درون collection ها پشتیبانی می کند. این ایندکس کلمات اصلی را ذخیره می کند و کلماتی چون ( "the", "a" , "or" ) در ایندکس ذخیره نمی شوند.

### ایندکس Hashe

برای پشتیبانی hash در sharding ، Mongodb از ایندکس hash پشتیبانی می کند. این ایندکس تنها از کوئری های با مقدار یکسان پشتیبانی می کند و کوئری های رنجی پشتیبانی نمی کند.

## Storage

Storage engine مدیریت دیتا در MongoDB را بر عهده دارد. MongoDB از engine های ذخیره ساز های متنوعی پشتیبانی می کند با توجه به اپلیکشین می توان از engine مناسب استفاده کرد.

Journal لاگی می باشد که برای ریکاور کردن دیتابیس در صورت خاموش شدن ناگهانی استفاده میشود. Option های زیادی برای برقراری توازن بین کارایی و قابلیت اطمینان وجود دارد.

GridFS سیستم مدیریت چند وجهی برای فایل های با حجم بالا می باشد و برای داکيومنت هایی که بیشتر 16MB می باشد.

Storage engine جزئی از دیتابیس می باشد که مسئولیت مدیریت ذخیره سازی دیتا را بر عهده دارد هم در دیسک هم در مموری (ram). MongoDB از چند engine برای بالا بردن کارایی در حجم کار پشتیبانی می کند. با توجه به بیزینس باید از engine مناسب برای ذخیره سازی استفاده کنیم.

### WiredTiger

Storage engine پیش فرض برای mongodb از ورژن 3.2 می باشد. مناسب برای اغلب برنامه ها می باشد به خصوص پروژه های جدید (دیتابیس های اولیه). WireTriger از همزمانی دسترسی به داکيومنت ها و ایجاد checkpoint و زیپ کردن داده ها پشتیبانی می کند.

### MMAPv1

Storage engine اصلی برای MongoDB قبل از ورژن 3.2 می باشد مناسب برای خواندن و نوشتن زیاد با update های زیاد می باشد.

Storage engine In-memory در ورژن MongoDB Enterprise می باشد. که داکيومنت ها را علاوه بر دیسک در memory برای پاخگویی و پیش بینی بهتر نگهداری می کند.

برای تغییر Storage engine از دستور زیر استفاده می کنیم

```
mongod --storageEngine mmapv1
```

یا در کانفیگ دیتابیس storage Engine را مشخص می کنیم.

```
storage:  
  
  engine: mmapv1
```

یا از دستور زیر استفاده می کنیم.

```
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
```

## Security

جهت فعال سازی authentication باید در فایل کانفیگ تغییراتی ایجاد کنیم و این آپشن را فعال نماییم به صورت پیش فرض mongod بدون نیاز به نام کاربری و کلمه عبور قابل استفاده می باشد. مراحل زیر جهت فعال کردن authentication می باشد.

1- connect to mongo

2- use dbname

3- نام کاربری و کلمه عبور را با توجه به دیتابیس مورد نظر تعریف نمایید

```
db.createUser(  
  {  
    user: "Your_user_name",  
    pwd: "Your_password",  
    roles: [ { role: "userAdminAnyDatabase", db: "Your_db_name" },  
             "readWriteAnyDatabase" ]  
  }  
)
```

exit

4- فایل کانفیگ را تغییر دهید

```
vi /etc/mongod.conf
```

```
security:
```

```
  authorization: enabled
```

5- ریستارت کردن دیتابیس

```
sudo systemctl restart mongod
```

برای تعریف Role و تغییر Role یا ایجاد کاربر باید دسترسی کاربر Admin باشد. این کاربر می تواند کاربران و Role ها را مدیریت کند ساخت کاربر اختصاص Role یا گرفتن آن از کاربر و یا تعریف Role در جزو وظایف کاربر Admin می باشد.

برای ساخت کاربر از اسکریپت زیر استفاده می کنیم.

```
db.createUser(user:"", pwd:"", roles:[role:"read", db:"Reporter"])
```

به عنوان مثال برای ساخت کاربر با نام **reportuser** و کلمه عبور **12345678** با دسترسی به دیتابیس های **products.reporting** و **sales** از دستور زیر استفاده می کنیم.

```
db.createUser({user:"reportsUser", pwd: "12345678", roles:[{role: "read", db: "reporting"},{ role: "read", db: "products"},{ role: "read", db: "sales"},{ role: "readWrite", db: "accounts"}]})
```

جدول **admin.system.roles** شامل تمامی **Role** های تعریف شده در سیستم می باشد

**Role** زیر برای ایجاد دسترسی مانیتورینگ یا **Mongostat** می باشد

```
db.createRole({role:"mongostatRole", privileges:[{resource:{cluster:true}, actions:["serverStatus"]}], roles:[]})
```

برای مشاهده **Role** های تعریف شده از کامند زیر استفاده می کنیم

```
db.getUser("mongostatRole")
```

**mongostatRole** نام **role** تعریف شده می باشد

برای گرفتن **Role** کاربری از دستور زیر استفاده می کنیم

```
db.revokeRolesFromUser("reportsUser",[{role: "readWrite", db: "reporter"}])
```

لیست **role** های موجود به شرح زیر است

```
read
readWrite
dbAdmin
userAdmin
```

```
clusterAdmin
readAnyDatabase
readWriteAnyDatabase
userAdminAnyDatabase
dbAdminAnyDatabase
```

برای دادن دسترسی **Role** به کاربری از دستور زیر استفاده می کنیم

```
db.grantRolesToUser("app", [{role: "read", db: "reporter"}])
```

برای تغییر پسورد کاربری از دستور زیر استفاده می کنیم

```
db.changeUserPassword("reporting", "Tt123456")
```

آپدیت کاربر :

```
db.updateUser("user",{
  roles: [
    { role: "readWrite", db: "database" }
  ]
})
```

دراپ کاربر :

```
db.removeUser("user")
or
db.dropUser("user")
```

مشاهده کاربران:

```
db.getUsers();
```

## Administration

### system database

#### Local

هر instance دیتابیس mongo دارای یک دیتابیس local است، که دادهای replication را ذخیره می کند. در replication دیتای این دیتابیس منتقل نمی شود. دیتابیس local دادهای داخلی (برای هر instance) replication را ذخیره می کند. دیتابیس local شامل collection های زیر می باشد

`local.startup_log`

On startup, each `mongod` instance inserts a document into `startup_log` with diagnostic information about the `mongod` instance itself and host information. `startup_log` is a capped collection. This information is primarily useful for diagnostic purposes.

`local.system.replset`

holds the replica set's configuration object as `local.system.replset` its single document. To view the object's configuration information, issue `rs.conf()` from the `mongo` shell. You can also query this collection directly.

`local.oplog.rs`

`local.oplog.rs` is the capped collection that holds the `oplog`. You set its size at creation using the `oplogSizeMB` setting. To resize the oplog after replica set initiation, use the [Change the Size of the Oplog](#) procedure. For additional information, see the [Oplog Size](#) section.

`local.replset.minvalid`

This contains an object used internally by replica sets to track replication status.

## admin

## Config

## journaling

## journal

یک فایل ترتیبی (sequential) از نوع باینری برای لاگ ترانس اکشن می باشد برای انتقال دیتابیس به حالت پایدار زمانی که دیتابیس hard shutdown می شود. journaling ابتدا دیتا را در journal می نویسد و بعد در دیتا فایل نوشته می شود. journaling به صورت پیش فرض از ورژن 2 نسخه 64 بیت فعال می باشد. فایل های journal در مسیر دیتا فایل قرار داند

مکانیسم WiredTiger از checkpoint برای ایجاد نقطه consistent برای ذخیره داده روی دیسک و recover توسط Mongoddb استفاده می کند. در صورتی که دیتابیس بین دو checkpoint به صورت ناخواسته خارج شود journaling برای Recover کردن از آخرین checkpoint مورد نیاز می باشد.

مراحل recovery با استفاده از journaling به صورت زیر می باشد

1. ابتدا از دیتا فایل identifier را برای آخرین checkpoint را جستجو می کند
2. در داخل journal رکوردی که با identifier آخرین checkpoint مطابقت دارد جستجو می کند.
3. تمام operation های داخل journal را از آخرین checkpoint اعمال می کند.

با journaling ، wiredtiger یک journal رکورد (داخل فایل journal یک رکورد ایجاد می کند) برای برای هر کلاینت که عملیات write را شروع می کند ایجاد می کند. journal رکورد شامل تمامی write های داخلی ناشی از نوشتن اولیه می شود. به عنوان مثال عملیات آپدیت یک داکيومنت در یک کالکشن که ناشی از تغییرات ایندکس می باشد، wiredtiger یک journal رکورد می سازد که رکورد شامل آپدیت و تغییرات ایندکس می باشد. هر رکورد آی دی مخصوص به خود را دارد. کمترین سایز journal رکوردها 128 بایت می باشد.



## پردازش journaling

Wiredtiger ، Mongoddb را به گونه ای کانفیگ می کند تا از in-memory بافر برای مرتب کردن رکوردهای journal استفاده نماید. تمامی journal رکورد ها تا حجم 128 کیلو بایت بافر می شوند.

wiretiger ذخیره شدن بافر journal رکورد ها روی دیسک در یکی از شرایط زیر انجام می دهد:

- برای ممبر های Replica set اگر operation ای منتظر oplog باشد (در شرایط زیر)
  - کوثری هایی که در حال اسکن هستند برخلاف oplog
  - عملیات read متاثر با عملیات دیگر باشد
- برای ممبرهای secondary بعد از عملیات batch برای oplog
- در صورتی که در عملیات write از j:true استفاده کرده باشیم wiredtiger سینک را انجام می دهد.
- سائز فایل journal حداکثر می تواند 100 مگا بایت باشد و به ازاء هر 100 مگا بایت wiredtiger یک فایل جدید ایجاد می کند زمانی که wiredtiger فایل جدید ایجاد می کند با فایل قبل سینک انجام می دهد.
- wiredtiger به صورت اتوماتیک فایل قدیمی journal را برای استفاده از تنها فایل لازم برای ریکاروری پاک می کند.
- بازهای زمانی 100 میلی ثانیه ای (پارامتر [storage.journal.commitIntervalMs](#))

## فایل های journal

برای فایل های journal، mongodb یک پوشه به اسم journal داخل دایرکتوری داخل دایرکتوری دیتا (dbpath) ایجاد می کند. فشرده سازی به صورت پیش فرض از طریق wiredtiger انجام می شود برای تغییر مکانیسم فشرده سازی می توان پارامتر journalCompressor را تغییر داد.

## journaling

Mongodb دارای دو ویو برای لایه استوریج می باشد ویو private برای نوشتن در فایل های journal (در رم) و ویو share برای ذخیره کردن نوشتن در دیتا فایل (بافر دیتا). در ابتدا mongodb عملیات نوشتن را در ویو private انجام می دهد.

1. mongodb ابتدا عملیات نوشتن را روی ویو private انجام می دهد.
2. mongodb سپس تغییرات ویو private را روی فایل journal روی دیسک هر 100 میلی ثانیه اعمال می کند. برای این کار از مکانیسمی به نام group commit استفاده می کند این مکانیسم باعث بهبود کارایی می شود زمانی که عملیات کامیت انجام می شود و تمامی write ها در زمان کامیت بلاک می شوند نوشتن در journal به صورت اتومیک می باشد.
3. به محض انجام journal commit، mongodb تمامی تغییرات را بین journal و ویو share انجام می دهد.
4. در آخر mongodb تغییرات ویو share را روی دیتا فایل اعمال می کند. این کار در بازهای زمانی 60 ثانیه ای انجام می شوند. mongodb از سیستم عامل می خواهد عملیات flush کردن بین ویو share و دیتا فایل را انجام دهد ممکن است سیستم عامل عملیات flush کردن در زمان هایی کمتر از 60 ثانیه انجام دهد مخصوصا زمانی که مموری سیستم کم باشد. برای تغییر بازه های زمانی نوشتن در دیتا فایل می توان از پارامتر `storage.syncPeriodSecs` استفاده نمود.

(برای ورژن 3)

فایل ~~journal~~ افزایشی است زمانی که حجم ~~journal~~ به یک گیگابایت برسد ~~mongodb~~ فایل جدید ایجاد می کند در صورتی که از پارامتر ~~storage.smallFiles~~ استفاده شود ~~mongod~~ زمان استارت ماکسیمم حجم فایل های ~~journal~~ را 128 مگابایت قرار می دهد.

فایل ~~lsn~~ در مسیر ~~journal~~ حاوی آخرین زمان ~~flush~~ کردن تغییرات به دیتا فایل می باشد. مسیر فایل ~~journal~~ می تواند شامل 2 یا 3 فایل ~~journal~~ باشد. در صورت ~~shutdown~~ شدن صحیح ~~mongodb~~ تمامی فایل های ~~journal~~ پاک می شوند و در صورت ~~dirty shutdown~~ فایل ها در مسیر باقی می ماند و برای ریکاوری استفاده می شود تا به نقطه مطمئن برسد. در صورتی ~~mongod~~ گرفتن فضای لازم برای ~~journal~~ برای بهبود کارایی مناسب بداند به جای نوشتن روی فایل قدیمی این فضا را می گیرد و فایل جدید می سازد. این کار با سیستم می باشد در صورتی که دقت کرده باشید زمان اولین استارت ~~mongod~~ مدت زمان کوتاهی برای تخصیص فضا به ~~journal~~ صرف می شود.

<https://www.mongodb.com/blog/post/how-mongodbs-journaling-works>

## **:Storage Engine**

دیتابیس ~~mongo~~ برای ذخیره سازی از ورژن 3.2 از ~~WiredTiger~~ به عنوان ~~storage engine~~ پیش فرض استفاده می کند در ورژن های قبلی از ~~MMAPv1~~ استفاده می شد

در ~~WiredTiger~~ از کنترل همزمانی در سطح ~~document-level~~ برای نوشتن استفاده می کند. برای ~~Read~~ و ~~Write~~ های همزمان ~~WiredTiger~~ از کنترل همزمانی خوشبینانه استفاده می کند. ~~WiredTiger~~ از لاک در سطح ~~global~~ ، ~~database~~ و ~~collection~~ استفاده می کند. در صورتی که ~~conflisct~~ بین دو عملیات رخ دهد یک عملیات انجام می شود و عملیات دوم مجدداً انجام می شود.

~~WiredTiger~~ از ~~MVCC~~ (MultiVersion Concurrency Control) استفاده می کند. در زمان شروع یک عملیات از دیتای عملیات ~~snapshot~~ می گیرد و داده های پایدار در مموری را شامل می شود. زمانی که عملیات نوشتن روی دیسک انجام می شود ~~WiredTiger~~ دیتای ~~snapshot~~ را با روشی که پایداری داده حفظ شود داخل دیتا فایل ها می نویسد. از ورژن ~~WiredTiger 3.6~~ از ~~checkpoint~~ استفاده می کند با

اینترنل های 60 ثانیه ای در ورژن های قبلی این اینتروال برای دیتای کاربر در هر 60 ثانیه یا به ازای هر 2 گیگ دیتا انجام می داد. در طول نوشتن checkpoint جدید ، checkpoint قدیمی valid می باشد(در صورت Terminate شدن یا رخ دادن خطایی در mongodb یا ریستارت شدن آن از checkpoint قبلی برای ریکاوری استفاده می کند). با استفاده از WiredTiger بدون journaling ، mongodb می تواند با استفاده از آخرین checkpoint ریکاوری را انجام دهد. و برای ریکاوری تغییرات بعد از checkpoint از journaling استفاده می کند.

### **:RAM**

رم مورد استفاده دیتابیس mongodb با استفاده از فرمول زیر محاسبه می شود

$$(0.5 * (4GB - 1 GB) = 1.5 GB)$$

و رد صورتی که رم کمتر از 256 باشد، 256 مگا بایت در نظر گرفته می شود.

### **RAID**

برای بهینه سازی عملکرد با توجه به لایه ذخیره سازی، از RAID-10 استفاده کنید. RAID-5 و RAID-6 به طور معمول عملکرد کافی برای پشتیبانی از استقرار MongoDB را ارائه نمی دهند.

## **MongoDB on Linux**

### **Kernel and File Systems**

در محیط لینوکس بهتر است از کرنل ورژن 2.6.36 به بعد استفاده شود. فایل سیستم مناسب XFS یا EXT4 می باشد که XFS پرفورمنس بهتری برای کار با mongodb دارد. با WiredTiger storage engine استفاده از XFS پرفورمنس بسیار بالاتری نسبت به EXT4 دارد.

### **Set vm.swappiness to 1**

**Swappiness** جزوه تنظیمات کرنل لینوکس برای استفاده از **Virtual memory** می باشد که رنج بین 0 تا 100 را می تواند داشته باشد

- تنظیم 0 به کرنل می گوید تنها از **swap** برای مشکلات **out-of-memory** استفاده کند.
- تنظیم 100 امکان مبادله دائم با دیسک را می دهد.

در صورتی که ورژن کرنل بالاتر از 3.5 باشد ممکن است مقدار صفر **swapping** را غیر فعال کند.

برای دیدن وضعیت **swappiness** از دستور زیر استفاده نمایید.

```
vi /etc/sysctl.conf
```

خط زیر را در فایل اضافه نمایید.

```
vm.swappiness = 1
```

در صورتی که امکان ریستارت سیستم وجود ندارد از دستور زیر استفاده برای اعمال استفاده نمایید.

```
sysctl vm.swappiness=1
```

## :Backup & Restore

متدهای مختلفی برای انجام عملیات پشتیبان گیری و بازیابی اطلاعات در MongoDB وجود دارد. از جمله استفاده از **Filesystem Snapshot** ها، استفاده از ابزارهای پیشنهادی MongoDB مانند **Cloud** **manager** یا **Ops manager**، و یا استفاده از **Mongodump**.

## :Mongodump با بک آپ

**mongodump** بخشی از استراتژی بک آپ به وسیله **mongorestore** می باشد. درواقع **mongodump** و **mongorestore** خدماتی هستند که با **BSON** کار میکنند و برای ایجاد بک آپ از دیتابیس های کوچک استفاده میشوند.

توجه داشته باشید که **mongodump** در خروجی خود، دسترسی به محتوای دیتابیس لوکال را محدود میکند.

**mongodump** با کانکت شدن به اینستنس های درحال اجرای **mongod** و **mongos** از داده ها بکاپ میگیرد. **mongodump** میتواند از سرور، دیتابیس، کالکشن یا حتی بخشی از کالکشن مورد نیاز با نوشتن کوئری بکاپ گیری کند. وقتی **mongodump** را بدون هیچ آرگومانی اجرا کنید، کامنت ما به اینستنس لوکال **mongodb** در پورت 27017 متصل شده و یک دیتابیس بکاپ به نام **dump/** در آن مسیر ایجاد میکند.

برای بکاپ گرفتن از اینستنس های **mongod** و **mongos** که بر روی پورت پیش فرض (27017) درحال اجرا هستند از کامند زیر استفاده کنید:

```
<your mongodb installation path>\bin\mongodump.exe
```

توجه: از **mongodump** برای بکاپ داده ها در ورژن های **mongodb 2.2** یا بعد از آن استفاده میشود.

برای اتصال **mongodump** به پورت یا هاست غیر پیش فرض از آپشن های **--host** و **--port** استفاده میشود:

```
<your mongodb installation path>\bin\mongodump.exe --host <your intended host> --port <port number>
```

برای تعیین مسیر متفاوت برای ذخیره خروجی، از آپشن های **--out** یا **--o** استفاده کنید:

```
<your mongodb installation path>\bin\mongodump.exe --out <your intended path>
```

برای محدود کردن مقدار داده های مشمول دیتابیس **dump** از آپشن های **--db** و **--collection** استفاده کنید:

```
<your mongodb installation path>\bin\mongodump.exe --collection  
CollectionName --db DatabaseName
```

این عملیات، یک **dump** از کالکشن و دیتابیس مورد نظر شما ایجاد میکند.

برای بازیابی بکاپ هایی که توسط **mongodump** گرفته شده از **mongorestore** استفاده میشود. **mongorestore** به صورت پیش فرض از بکاپ دیتابیس هایی که در مسیر **dump/** قرار دارند برای **restore** استفاده میکند.

برای استفاده از **mongorestore** و اتصال به یک **mongod** و **mongos** فعال در پورت پیش فرض، از کامند زیر استفاده میشود:

```
<your mongodb installation path>\bin\mongorestore.exe --db <database name>  
<path to the backup>
```

برای مثال:

```
c:\mongodb\server\3.0\bin\mongorestore.exe dump-2013-10-25/
```

این کامند، بکاپ دیتابیس در مسیر **dump-2013-10-25** را به اینستنس **mongod** که بر روی سرور محلی (27017) فعال است، وارد میکند.

درواقع **mongorestore** به طور پیش فرض به لوکال هاست و پورت محلی (27017) متصل میشود. اما برای بازیابی بر روی **host** یا **port** ای غیر از آن، باید از آپشن های **--host** و **--port** استفاده شود:

```
mongorestore --host mongodb1.example.net --port 3017 --username user --  
password pass /opt/backup/mongodump-2013-10-24
```

در کامند بالا **username** و **password** را برای احراز هویت وارد کرده ایم.

**: oplog**

oplog (operation log) یک کالکشن ویژه از نوع <sup>1</sup>capped می باشد. که تمامی operation های انجام شده در دیتابیس را ثبت می کند.

Mongodb همه operation ها را ابتدا روی Primary انجام می دهد سپس روی oplog سیستم primary فعال می شود. سپس سیستم های secondary این operation کپی و روی سیستم های خود به صورت asynchronous اعمال می کند. تمامی ممبرهای replicaset به کپی از oplog روی سیستم های خود دارند و در کالکشن local.oplog.rs وضعیت فعلی دیتابیس را مشخص می کند.

در جهت تسهیل کردن replication تمامی ممبر ها heartbeat (ping) برای سار اعضا می فرستند. هر کدام از ممبرهای secondary می توانند oplog را از سایر اعضا import کنند.

هر operation در oplog به صورت <sup>2</sup>idempotent می باشد. این به معنای داشتن نتیجه یکسان در صورت یک بار یا چندین بار اعمال می باشد.

مقدار سائز پیش فرض oplog به storage engine آن بستگی دارد.

**For Unix and Windows systems**

The default oplog size depends on the storage engine:

Storage Engine	Default Oplog Size	Lower Bound	Upper Bound
In-Memory Storage Engine	5% of physical memory	50 MB	50 GB
WiredTiger Storage Engine	5% of free disk space	990 MB	50 GB

در اغلب موارد مقدار پیش فرض مناسب می باشد. برای مثال اگر oplog 5% از فضای خالی دیسک باشد و که در 24 ساعت پر می شود. سیستم secondary می تواند برای 24 ساعت عملیات کپی کردن را متوقف کند

<sup>1</sup> capped collection یک کالکشن با اندازه ثابت می باشد برای پشتیبانی از عملیات های با insert و فراخوانی بالا و مانند یک بافر دایره ای عمل می کند در واقع در فضای لازم ایجاد می شود و دادهای جدید را با بازنوشتن روی قدیمی ترین داده انجام می دهد.

<sup>2</sup> idempotent به معنی کیفیت انجام یه تراکنش در صورت اعمال یک بار یا چندین باره با یک نوع ورودی می باشد.



بدون آنکه عملیات Replication دچار اختلال شود به هر حال اغلب replica set ها دارای حجم کاری کمتری هستند و تعداد بیشتر opreition ها را می توانند در oplog نگه دارند.

قبل از ایجاد oplog توسط mongod می توان با آپشن oplogsizeMB (پارامترهای mongod.conf) سایز آن را مشخص نمود. بعد از ایجاد replica set می توان با کامند admim replSetResizeOplog می توان سایز oplog را بدون نیاز به ریستارت شدن تغییر داد.

بعد از ورژن 4 oplog برای انجام 'majority commit point' می تواند بیش از اندازه پیکربندی رشد کند.

میزان workload سیستم برای تخمین سایز مناسب oplog لازم می باشد.

### سایز oplog

در صورتی که سیستم غالباً read داشته باشد و مقدار write آن پایین باشد، oplog با حجم پایین مناسب می باشد. هرچه سایز oplog بیشتر باشد تحمل پذیری نسبت به خطا بالاتر می رود و مجموعه را انعطاف پذیرتر می کند. در صورتی که یکی از موارد زیر باشد باید حجم oplog بیشتر در نظر گرفته شود.

- آپدیت چندین داکيومنت همزمان

oplog باید چندین آپدیت را به operation های مستقل و جدا تقسیم کند برای بهینه کردن operation ها به صورت idempotency که می تواند مزیت مناسبی برای oplog باشد بدون افزایش حجم داده یا سایز استفاده شده از دیسک می باشد.

- پاک کردن داده ها به اندازه داده های درج شده

در صورتی که در سیستم تقریباً داده ها درج شده پاک شوند دیتابیس روی دیسک رشد نمی کند ولی لاگ سیستم خیلی بزرگ می شود.

---

majority commit point 1 برای دیدن وضعیت آن می توان از دستور زیر استفاده نمود قسمت replSetGetStatus.optimes.lastCommittedOpTime مشخص می کند که majority commit یا وضعیت کامیت روی سایر ممبر ها به چه صورت می باشد..

```
db.adminCommand( { replSetGetStatus : 1} )
```

## • تعداد آپدیت زیاد همزمان

در صورتی که حجم آپدیت زیادی انجام شود، سائز داکيومنت افزایش نمی یابد و محتویات تعداد رکورد های زیادی تغییر می کند ولی از طحاظ تعداد رکورد تغییری حاصل نمی شود.

## وضعیت oplog

برای مشاهده وضعیت oplog شامل سائز ، زمان انجام تغییرات از کامند `rs.printReplicationInfo()` استفاده می شود در صورت وجود تاخیر (lag) در ممبر های secondary با استفاده از این دستور وضعیت اختلاف ممبرها مشخص می شود. میزان تاخیر از ورژن 4.2 به بعد توسط پارامتر `flowControlTargetLagSeconds` قابل پیکربندی می باشد مقدار پیش فرض آن 1 ثانیه می باشد این پارامتر در صورت فعال بودن <sup>1</sup> `flow control` (`enableFlowControl`) (به صورت پیش فرض فعال `true` می باشد) تاثیر گذار می باشد. `replication lag` تاخیر یک `operation` بین `primary` و سیستم های `Secondary` می باشد. `replication lag` مسئله قابل توجهی می باشد و می تواند تاثیر به سزایی روی `Replica set` بگذارد و باعث تغییر `role` سرور `primary` شود و سیستم را به حالت `inconsistent` ببرد. مقدار `delay` ممبر ها نمی تواند بیشتر از پارامتر `members[n].slaveDelay` باشد.

## Slow Oplog Application

از ورژن 4.2 ممبر های `Secondary` لاگ های oplog که زمان بیشتری نسبت به حد مشخص شده طول می کشد را نیز اعمال می کند. پارامتر `slows` جز پارامترهای `mongod` می باشد و مقدار پیش فرض آن 100 میلی ثانیه است که در `loglevel` ثبت می شود

برای مانیتورینگ از کامند زیر نیز می توان استفاده نمود.

```
db.adminCommand( { replSetGetStatus: 1 } )
```

<sup>1</sup> `flow control` برای کنترل میزان `majority commit` بین سرور `primary` و ممبرهای `Secondary` با استفاده از پارامترهای مشخص شده (`flowControlTargetLagSeconds`) می باشد

آیتم های که برای **oplog** نیاز به مانیتورینگ دارند به شرح زیر می باشند

Metric Description	Name	Metric Type	Availability
Size of the oplog (MB)	logSizeMB	Other	getReplicationInfo
Oplog window (seconds)	timeDiff	Other	getReplicationInfo
Replication Lag: delay between a write operation on the primary and its copy to a secondary (milliseconds)	members.optimeDate[primary] - members.optimeDate[secondary member] **	Work: Performance	replSetGetStatus
Replication headroom: difference between the primary's oplog window and the replication lag of the secondary (milliseconds)	getReplicationInfo.timeDiff x 1000 - (replSetGetStatus.members.optimeDate[primary] - replSetGetStatus.members.optimeDate[secondary member])	Work: Performance	getReplicationInfo and replSetGetStatus
Replica set member state	members.state	Resource: Availability	replSetGetStatus

## Monitoring

برای مانیتورینگ دیتابیس و اینستنس از روش های زیر می توانیم استفاده نماییم

### • mongotop

این یوتیلیتی زمان سپری شده برای خواندن و نوشتن توسط اینستنس **mongodb** مشخص می کند این اطلاعات در اساس داکيومنت ها مجزا می باشند. برای اجرا این یوتیلیتی دستور باید در محیط سیستم اجرا نمود نه در محیط شل **mongo**. به صورت پیش فرض از **Default: localhost:27017** استفاده می کند فرمت کلی دستور به صورت زیر می باشد.

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]
```

برای مشاهده خروجی هر 15 ثانیه از دستور زیر استفاده می کنیم

خروجی دستور به صورت زیر می باشد

ns	total	read	write
admin.system.roles	0ms	0ms	0ms
admin.system.version	0ms	0ms	0ms
local.me	0ms	0ms	0ms
local.oplog.rs	0ms	0ms	0ms
local.replset.minvalid	0ms	0ms	0ms
local.startup_log	0ms	0ms	0ms
local.system.indexes	0ms	0ms	0ms

## • mongostat

این یوتیلیتی دید کلی از کوئری های در حال اجرای روی اسنتنس نمایش می دهد

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]
```

mongostat

خروجی دستور به صورت زیر می باشد

insert	query	update	delete	getmore	command	flushes	mapped	vsize	res	faults	locked	db	idx	miss %	qrio	arise	netin	netout	conn	time
*0	17	*0	*0	0	1 0	0	2.03g	4.44g	299m	0	statementPay:0.4%	0	0 0	1 0	2k	25k	30		15:30:46	
*2	25	*0	*0	0	3 0	0	2.03g	4.44g	299m	0	statementPay:0.0%	0	0 0	1 0	5k	40k	30		15:30:47	
*8	23	*0	*0	0	6 0	0	2.03g	4.44g	299m	0	statementPay:0.4%	0	0 0	4 0	2k	33k	30		15:30:48	
10	33	*0	*0	0	18 0	0	2.03g	4.44g	299m	0	statementPay:0.2%	0	0 0	3 0	12k	50k	30		15:30:49	
1	28	*0	*0	0	2 0	0	2.03g	4.44g	299m	0	statementPay:0.7%	0	0 0	3 0	5k	40k	30		15:30:50	

```
mongostat --rowcount 20 1
```

یا

```
mongostat --rowcount 20
```

```
mongostat -n 20 1
```

```
mongostat -n 20
```

هر یک ثانیه به مدت 20 ثانیه خروجی می دهد

insert	query	update	delete	getmore	command	flushes	mapped	vsize	res	faults	locked	db	idx	miss	%	qriw	ariw	netin	netout	conn	time
2	32	*0	*0	0	110	0	2.03g	4.44g	300m	0	statementPay:4.7%	0	0	0	0	0	0	6k	52k	30	15:43:19
2	28	1	*0	0	30	0	2.03g	4.44g	300m	0	statementPay:0.4%	0	0	0	0	0	10	7k	40k	30	15:43:20
*0	29	*0	*0	0	10	0	2.03g	4.44g	300m	0	statementPay:0.0%	0	0	0	0	0	20	4k	41k	30	15:43:21
9	42	1	*0	0	100	0	2.03g	4.44g	300m	0	statementPay:5.0%	0	0	0	0	0	30	13k	78k	30	15:43:22
2	16	*0	*0	0	70	0	2.03g	4.44g	300m	0	statementPay:0.1%	0	0	0	0	0	20	4k	38k	30	15:43:23
13	19	*0	*0	0	220	0	2.03g	4.44g	300m	0	statementPay:0.3%	0	0	0	0	0	10	13k	38k	30	15:43:24
4	21	*0	*0	0	50	0	2.03g	4.44g	300m	0	statementPay:0.7%	0	0	0	0	0	20	5k	40k	30	15:43:25
4	20	*0	*0	0	50	0	2.03g	4.44g	300m	0	statementPay:0.2%	0	0	0	0	0	10	6k	29k	30	15:43:26
1	20	*0	*0	0	20	0	2.03g	4.44g	300m	0	statementPay:0.4%	0	0	0	0	0	20	4k	35k	30	15:43:27
8	22	*0	*0	0	130	0	2.03g	4.44g	300m	0	statementPay:0.2%	0	0	0	0	0	0	9k	34k	30	15:43:28
insert	query	update	delete	getmore	command	flushes	mapped	vsize	res	faults	locked	db	idx	miss	%	qriw	ariw	netin	netout	conn	time
3	23	1	*0	0	120	0	2.03g	4.44g	234m	0	statementPay:0.4%	0	0	0	0	0	10	8k	73k	30	15:43:29
5	25	1	*0	0	60	0	2.03g	4.44g	300m	0	statementPay:0.2%	0	0	0	0	0	0	8k	41k	30	15:43:30
2	16	*0	*0	0	30	0	2.03g	4.44g	220m	0	statementPay:0.5%	0	0	0	0	0	0	3k	24k	30	15:43:31
10	27	*0	*0	0	110	0	2.03g	4.44g	300m	0	statementPay:0.2%	0	0	0	0	0	10	11k	42k	30	15:43:32
4	17	*0	*0	0	90	0	2.03g	4.44g	300m	0	statementPay:0.1%	0	0	0	0	0	20	5k	26k	30	15:43:33
2	18	*0	*0	0	110	0	2.03g	4.44g	300m	0	statementPay:0.5%	0	0	0	0	0	30	4k	25k	30	15:43:34
15	29	*0	*0	0	160	0	2.03g	4.44g	300m	0	statementPay:0.3%	0	0	0	0	0	20	15k	49k	30	15:43:35
5	24	1	*0	0	60	0	2.03g	4.44g	300m	0	statementPay:0.1%	0	0	0	0	0	20	7k	32k	30	15:43:36
*0	14	*0	*0	0	10	0	2.03g	4.44g	300m	0	statementPay:0.4%	0	0	0	0	0	10	2k	23k	30	15:43:37
*0	29	*0	*0	0	50	0	2.03g	4.44g	300m	0	statementPay:0.0%	0	0	0	0	0	0	3k	56k	30	15:43:38

برای خروجی گرفتن هر 5 دقیقه می توانیم از دستورات زیر استفاده کنیم دستورات معادل یکدیگر می باشند

```
mongostat --rowcount 0 300
```

```
mongostat -n 0 300
```

```
mongostat 300
```

خروجی گرفتن هر 5 دقیقه برای یک ساعت با دستورات زیر انجام می شود

```
mongostat --rowcount 12 300
```

```
mongostat -n 12 300
```

برای مشاهده کردن وضعیت تمامی ممبر ها در replaset از دستور زیر استفاده می کنیم

```
mongostat --discover
```

• <http://localhost:28017>

از این آدرس نیز می توان برای مانور کردن دیتابیس استفاده نمود.

• [مانیتورینگ زیبکس](#)

تمپلیت پیش فرض برای مانیتورینگ زیبکس در فایل زیر قرار دارند قرار داده شده است

آیتم های این تمپلیت خروجی دستور db.serverStatus می باشد.

توضیح هر یک از موارد تمپلیت به شرح زیر می باشد

## • backgroundFlushing

برای بررسی نحوه ذخیره سازی روی دیسک و **journaling** از این آیتم استفاده می شود. این آیتم موارد زیر را شامل می شود

### ✓ backgroundFlushing.flushes

تعداد دفعاتی که دیتابیس **flush** می کند و روی دیسک می نویسد را نشان می دهد. با آپ بودن دیتابیس رشد می کند (cumulative می باشد)

### ✓ backgroundFlushing.total\_ms

مجموع زمان بر حسب میلی ثانیه که دیتابیس **flush** می کند و روی دیسک می نویسد را نشان می دهد. با آپ بودن دیتابیس رشد می کند (cumulative می باشد)

### ✓ backgroundFlushing.average\_ms

✓ میانگین زمان بر حسب میلی ثانیه که دیتابیس **flush** می کند و روی دیسک می نویسد را نشان می دهد. با آپ بودن دیتابیس رشد می کند (cumulative می باشد). این مقدار می تواند معیاری برای حالت نرمال سیستم باشد در صورتی که سیستم حالت غیر منطقی داشته باشد این مقدار انحراف را نمایش می دهد.

### ✓ backgroundFlushing.last\_ms

زمان بر حسب میلی ثانیه که دیتابیس آخرین **flush** را انجام داده و کامل کرده است و روی دیسک می نویسد را نشان می دهد. برای بررسی پرفورمنس سیستم به صورت هیستوری استفاده می شود.

## • globalLock

گزارشی که لاگ های دیتابیس را نمایش می دهد. به طور کلی جزئیات بیشتری در مورد لاگ نمایش می دهد.

`globalLock.totalTime` ✓

مجموع زمان بر حسب میلی ثانیه از زمان آخرین شروع دیتابیس و ساخته شدن `globalLock` می باشد (cumulative می باشد)

`globalLock.currentQueue` ✓

گزارش که تعداد `operation` هایی که به دلیل لاگ در صف هستند را نمایش می دهد.

`globalLock.currentQueue.total` ✓

مجموع تمام `operation` هایی که در انتظار لاگ می باشند (مجموع

`globalLock.currentQueue.readers`

و `globalLock.currentQueue.writers` می باشد )

`globalLock.currentQueue.readers` ✓

تعداد `operation` هایی که در حال حاضر در صف هستند و در انتظار لاگ از نوع

`read` می باشد را نمایش می دهد.

The number of operations that are currently queued and waiting for the read lock. A consistently small read-queue, particularly of shorter operations, should cause no concern.

`globalLock.currentQueue.writers` ✓

تعداد `operation` هایی که در حال حاضر در صف هستند و در انتظار لاگ از نوع

`write` می باشد را نمایش می دهد.

`globalLock.activeClients` ✓

گزارشی که تعداد کلاینت ها و تعداد `Read` و `write` کلاینت ها را نشان می دهد

`globalLock.activeClients.total` ✓

مجموع تمام کانکشن های اینترنال به دیتابیس شامل **thread** های سیستمی و صف های **read** و **write** می باشد این مقدار از `globalLock.currentQueue.total` بیشتر می باشد.

`globalLock.activeClients.readers` ✓  
تعداد کانکشن های اکتیوی که عملیات **Read** را انجام می دهند نمایش می دهد.

`globalLock.activeClients.writers` ✓  
تعداد کانکشن های اکتیوی که عملیات **Write** را انجام می دهند نمایش می دهد.

• **dur**

A document that reports the `mongod` instance's **journaling-related operations** and performance. MongoDB reports on this data based on 3 second intervals, collected between 3 and 6 seconds in the past.

`dur.commits`

The number of transactions written to the **journal** during the last **journal group commit interval**.

`dur.journaledMB`

The amount of data in megabytes (MB) written to **journal** during the last **journal group commit interval**.

`dur.writeToDataFilesMB`

The amount of data in megabytes (MB) written from **journal** to the data files during the last **journal group commit interval**.

`dur.compression`

`dur.compression`

The compression ratio of the data written to the **journal**:



copy

copied

```
( journaled_size_of_data / uncompressed_size_of_data )  
dur.commitsInWriteLock
```

The count of the commits that occurred while a write lock was held. Commits in a write lock indicate a MongoDB node under a heavy write load and call for further diagnosis.

`dur.earlyCommits`

The number of times MongoDB requested a commit before the scheduled [journal group commit interval](#). Use this value to ensure that your [journal group commit interval](#) is not too long for your deployment.

`dur.timeMS`

A document that reports on the performance of the `mongod` instance during the various phases of journaling in the last [journal group commit interval](#).

`dur.timeMS.dt`

The amount of time, in milliseconds, over which MongoDB collected the `dur.timeMS` data. Use this field to provide context to the other `dur.timeMS` field values.

`dur.timeMS.prepLogBuffer`

The amount of time, in milliseconds, spent preparing to write to the journal. Smaller values indicate better journal performance.

`dur.timeMS.writeToJournal`

The amount of time, in milliseconds, spent actually writing to the journal. File system speeds and device interfaces can affect performance.

`dur.timeMS.writeToDataFiles`

The amount of time, in milliseconds, spent writing to data files after journaling. File system speeds and device interfaces can affect performance.

```
dur.timeMS.remapPrivateView1
```

The amount of time, in milliseconds, spent remapping copy-on-write memory mapped views. Smaller values indicate better journal performance.

```
dur.timeMS.commits
```

The amount of time, in milliseconds, spent for commits.

```
dur.timeMS.commitsInWriteLock
```

The amount of time, in milliseconds, spent for commits that occurred while a write lock was held.

### connections

A document that reports on the status of the connections. Use these values to assess the current load and capacity requirements of the server.

```
connections.current
```

The number of incoming connections from clients to the database server . This number includes the current shell session. Consider the value of [connections.available](#) to add more context to this datum.

The value will include all incoming connections including any shell connections or connections from other servers, such as [replica set](#) members or [mongos](#) instances.

```
connections.available
```

The number of unused incoming connections available. Consider this value in combination with the value of `connections.current` to understand the connection load on the database, and the [UNIX ulimit Settings](#) document for more information about system thresholds on available connections.

`connections.totalCreated`

Count of **all** incoming connections created to the server. This number includes connections that have since closed.

`connections.active`

The number of active client connections to the server. Active client connections refers to client connections that currently have operations in progress.

ve journaling enabled.

## network

copy

copied

```
"network" : {  
  "bytesIn" : <num>,  
  "bytesOut" : <num>,  
  "numRequests" : <num>  
},
```

network

A document that reports data on MongoDB's network use.

`network.bytesIn`

The number of bytes that reflects the amount of network traffic received *by* this database. Use this value to ensure that network traffic sent to the `mongod` process is consistent with expectations and overall inter-application traffic.

`network.bytesOut`

The number of bytes that reflects the amount of network traffic sent *from* this database. Use this value to ensure that network traffic sent by the `mongod` process is consistent with expectations and overall inter-application traffic.

`network.numRequests`

The total number of distinct requests that the server has received. Use this value to provide context for the `network.bytesIn` and `network.bytesOut` values to ensure that MongoDB's network utilization is consistent with expectations and application use.

## opcounters

copy

copied

```
"opcounters" : {
  "insert" : <num>,
  "query" : <num>,
  "update" : <num>,
  "delete" : <num>,
  "getmore" : <num>,
  "command" : <num>
},
```

opcounters

A document that reports on database operations by type since the `mongod` instance last started.

These numbers will grow over time until next restart. Analyze these values over time to track database utilization.

### NOTE

The data in `opcounters` treats operations that affect multiple documents, such as bulk insert or multi-update operations, as a single operation.

See `metrics.document` for more granular document-level operation tracking.

Additionally, these values reflect received operations, and increment even when operations are not successful.

`opcounters.insert`

The total number of insert operations received since the `mongod` instance last started.

`opcounters.query`

The total number of queries received since the `mongod` instance last started.

`opcounters.update`

The total number of update operations received since the `mongod` instance last started.

`opcounters.delete`

The total number of delete operations since the `mongod` instance last started.

`opcounters.getmore`<sup>1</sup>

The total number of “getmore” operations since the `mongod` instance last started. This counter can be high even if the query count is low. Secondary nodes send `getMore` operations as part of the replication process.

`opcounters.command`

The total number of commands issued to the database since the `mongod` instance last started.

`opcounters.command` counts all **commands** **except** the write commands: `insert`, `update`, and `delete`.

## repl

copy

copied

```
"repl" : {
  "hosts" : [
    <string>,
    <string>,
    <string>
  ],
  "setName" : <string>,
  "setVersion" : <num>,
  "ismaster" : <boolean>,
  "secondary" : <boolean>,
  "primary" : <hostname>,
  "me" : <hostname>,
  "electionId" : ObjectId(""),
  "rbid" : <num>,
  "replicationProgress" : [
    {
      "rid" : <ObjectId>,
      "optime" : { ts: <timestamp>, term: <num> },
      "host" : <hostname>,
      "memberId" : <num>
    },
    ...
  ]
}
```

repl

A document that reports on the replica set configuration. `repl` only appear when the current host is a replica set. See [Replication](#) for more information on replication.

`repl.hosts`

An array of the current replica set members' hostname and port information ("host:port").

`repl.setName`

A string with the name of the current replica set. This value reflects the `--replSet` command line argument, or `replSetName` value in the configuration file.

```
repl.ismaster
```

A boolean that indicates whether the current node is the **primary** of the replica set.

```
repl.secondary
```

A boolean that indicates whether the current node is a **secondary** member of the replica set.

```
repl.primary
```

*New in version 3.0.*

The hostname and port information ("`host:port`") of the current **primary** member of the replica set.

```
repl.me
```

*New in version 3.0:* The hostname and port information ("`host:port`") for the current member of the replica set.

```
repl.rbid
```

*New in version 3.0.*

**Rollback** identifier. Used to determine if a rollback has happened for this **mongod** instance.

```
repl.replicationProgress
```

*Changed in version 3.2:* Previously named `serverStatus.repl.slaves`.

*New in version 3.0.*

An array with one document for each member of the replica set that reports replication process to this member. Typically this is the primary, or secondaries if using chained replication.

To include this output, you must pass the `repl` option to the `serverStatus`, as in the following:

copy

copied

```
db.serverStatus({ "repl": 1 })
db.runCommand({ "serverStatus": 1, "repl": 1 })
```

The content of the `repl.replicationProgress` section depends on the source of each member's replication. This section supports internal operation and is for internal and diagnostic use only.

`repl.replicationProgress[n].rid`

An ObjectId used as an ID for the members of the replica set. For internal use only.

`repl.replicationProgress[n].optim  
e`

Information regarding the last operation from the `oplog` that the member applied, as reported from this member.

`repl.replicationProgress[n].ho  
st`

The name of the host in `[hostname]:[port]` format for the member of the replica set.

`repl.replicationProgress[n]  
.memberID`

The integer identifier for this member of the replica set.

metrics



A document that returns various statistics that reflect the current use and state of a running `mongod` instance.

`metrics.document`

A document that reflects document access and modification patterns. Compare these values to the data in the `opcounters` document, which track total number of operations.

`metrics.document.deleted`

The total number of documents deleted.

`metrics.document.inserted`

The total number of documents inserted.

`metrics.document.returned`

The total number of documents returned by queries.

`metrics.document.updated`

The total number of documents updated.

`mem`

A document that reports on the system architecture of the `mongod` and current memory use.

`mem.bits`

A number, either 64 or 32, that indicates whether the MongoDB instance is compiled for 64-bit or 32-bit architecture.

`mem.resident`

The value of `mem.resident` is roughly equivalent to the amount of RAM, in megabytes (MB), currently used by the database process. During normal use, this value tends to grow. In dedicated database servers, this number tends to approach the total amount of system memory.

`mem.virtual`

`mem.virtual` displays the quantity, in megabytes (MB), of virtual memory used by the `mongod` process.

With `journaling` enabled and if using MMAPv1 storage engine, the value of `mem.virtual` is at least twice the value of `mem.mapped`. If `mem.virtual` value is significantly larger than `mem.mapped` (e.g. 3 or more times), this may indicate a memory leak.

`mem.supported`

A boolean that indicates whether the underlying system supports extended memory information. If this value is false and the system does not support extended memory information, then other `mem` values may not be accessible to the database server.

`mem.mapped`

*Only for the MMAPv1 storage engine.*

The amount of mapped memory, in megabytes (MB), by the database. Because MongoDB uses memory-mapped files, this value is likely to be roughly equivalent to the total size of your database or databases.

`mem.mappedWithJournal`

*Only for the MMAPv1 storage engine.*

The amount of mapped memory, in megabytes (MB), including the memory used for journaling. This value will always be twice the value of `mem.mapped`. This field is only included if journaling is enabled.

`mem.note`

The field `mem.note` appears if `mem.supported` is false.

The `mem.note` field contains the text: "not all mem info support on this platform".

## globalLock

copy

copied

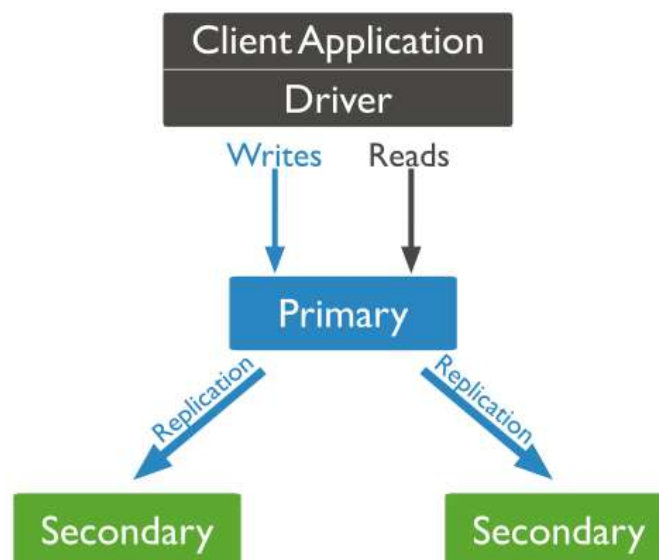
```
"globalLock" : {  
  "totalTime" : <num>,  
  "currentQueue" : {  
    "total" : <num>,  
    "readers" : <num>,  
    "writers" : <num>  
  },  
  "activeClients" : {  
    "total" : <num>,  
    "readers" : <num>,  
    "writers" : <num>  
  }  
},
```

**Replication**

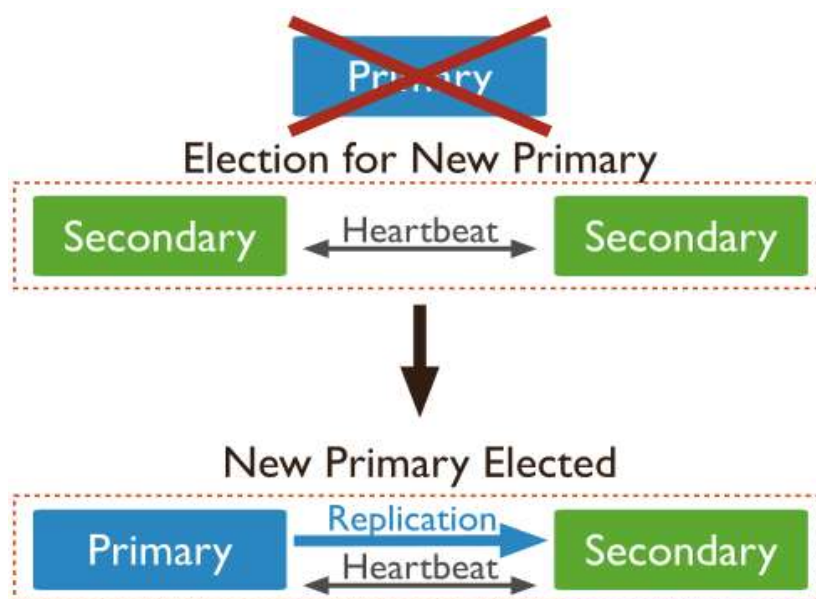
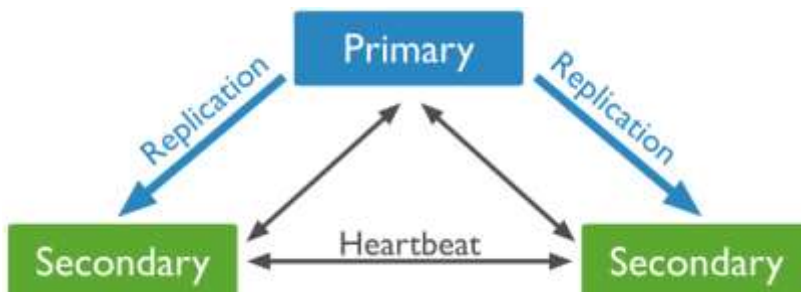
یک replica set در MongoDB در واقع گروهی از پردازش های mongod میباشد که از دیتاسِت های مشابه نگهداری میکند. replica set ها امکانات زیادی از جمله قابلیت دسترس پذیری بالا را برای ما فراهم میکنند. در واقع replication با ایجاد چندین کپی از دیتا بر روی سرورهای مختلف، خیال ما را از ایجاد مشکل برای یکی از سرورها آسوده میکند.

یک replica set شامل چندین نود داده و یک نود اختیاری به نام arbiter است. در مورد نودهای داده تنها یک نود به عنوان نود اصلی (primary) در نظر گرفته میشود و باقی نودها، نود ثانویه (secondary) فرض میشوند.

نود اصلی تمام عملیات نوشتاری را دریافت میکند و یک replica set میتواند تنها یک نود اصلی داشته باشد. اگرچه در برخی شرایط، ممکن است یک نود دیگر به صورت موقتی به عنوان نود اصلی در نظر گرفته شود اما اساس کار، وجود یک نود اصلی میباشد. به شکل زیر توجه کنید:



تمامی تغییرات بر روی داده های اصلی در oplog نود اصلی ذخیره میشود و نودهای ثانویه از oplog نود اصلی کپی برداری کرده و عملیات لازم را بر روی دیتاسِت های خود اعمال میکنند. اگر نود اصلی به هر دلیل از دسترسی خارج شود، یکی از نودهای ثانویه واجد شرایط به عنوان نود ثانویه انتخاب میشود. به مکانیزم زیر توجه کنید:



### Replica set secondary Members

- Priority 0 Replica set Members
- Hidden Replica set Members
- Delayed Replica set Members

از این مدل برای انتقال دیتا با تاخیر استفاده می شود برای کانفیگ این مدل از دستور زیر استفاده می کنیم.

```
{
  "_id" : <num>,
  "host" : <hostname:port>,
  "priority" : 0,
```

```
"slaveDelay" : <seconds>,  
"hidden" : true  
}
```

اکنون به توضیح چگونگی اجرای replication در MongoDB میپردازیم:

### 1. Centos در Replica set

تنظیم Hostname سیستم نام و آی پی (حداقل دو سرور) سیستم به صورت زیر:

```
hostname NewServerName  
vi /etc/hosts  
192.168.X.X NewServerName1 NewServerName1.localdomain localhost4 localhost4.localdomain4  
192.168.X.Y NewServerName2 NewServerName2.localdomain localhost4 localhost4.localdomain4  
192.168.X.Z NewServerName3 NewServerName3.localdomain localhost4 localhost4.localdomain4  
  
vi /etc/hostname  
hostname
```

و بعد از انجام تنظیمات با استفاده از دستور زیر سرویس شبکه را ریست می کنیم.

```
service network restart
```

برای فعال کردن Authentication and Authorization روی Replica set مراحل زیر را انجام می

دهیم

روی سیستم Primary به mongod حالت Standalone وصل شوید

سپس کاربر ادمین را با دستور زیر بسازید

```
db.createUser(
{
  user: "Admin",
  pwd: "abc123",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDatabase" ]
}
)

db.grantRolesToUser("Admin ", [ { role: "userAdminAnyDatabase", db: "admin" } ])

db.grantRolesToUser("Admin ", [ { role: "clusterAdmin", db: "admin" } ])
```

کلید در ماشین Primary ساخته می شود و باید روی تمامی ماشین ها کپی شود

```
mkdir /etc/mongokey /
chown mongod. /etc/mongokey/ -fR
```

روی ماشین Primary دستور زیر را اجرا و فایل را در سایر سرور ها کپی نمایید

```
cd /etc/mongokey
openssl rand -base64 741 > mongoKey
chmod 600 mongod-keyfile
```

```
scp mongoKey root@host2:/etc/mongokey/
scp mongoKey root@host3:/etc/mongokey/
```

برای ایجاد replicate تغییرات زیر را در فایل کانفیک mongod ایجاد می کنیم

### vi /etc/mongod.conf

```
net:
  port: 27017
  bindIp: 0.0.0.0 # Listen to local interface only, comment to listen on all interfaces.
security:
  authorization: enabled
  keyFile: /etc/mongokey/mongoKey
```

```
#operationProfiling:
replication:
  replSetName: "replicate-name"
#sharding:
##Enterprise-Only Options
#auditLog:
#snmp:
```

بعد از اعمال تغییرات **mongodb** را ریستارت نمایید

```
systemctl restart mongod.service
```

برای ایجاد replicaset دستور زیر را در سیستمی که به عنوان Primary در نظر گرفته شده است اجرا نمایید

```
rs.initiate{ )
_ id : "opnbank-rs,"
members] :
_ } id: 0, host: "opn-bank1:27018,{ "
_ } id: 1, host: "opn-bank2:27018,{ "
_ } id: 2, host: "opn-bank3:27018{ "
[
{
```

برای مشاهده و اعمال دستورات دستور زیر را روی همه ماشین ها اجرا نمایید

```
rs.initiate()
```

در صورت بروز اشتباه و نیاز به پاک کردن Replica set مراحل زیر باید انجام شود.

در صورت نیاز به پاک کردن Replica set باید از دستورات زیر زمانی که **mongodb** در حالت عادی استارت است استفاده نماییم (تغییرات replication در فایل کانفیگ کامنت باشد)

با کاربری که دسترسی لازم را داشته باشد لاگین نمایید

در صورتی که دسترسی نداشته باشد با دستور زیر دسترسی لازم را می دهیم



```
db.grantRolesToUser("admin", [{role: "readWrite", db: "local"}]);
```

به دیتابیس local سوئیچ نمایید

```
use local
```

دستور زیر را اجرا نمایید

```
db.system.replset.remove({});
```

برای اطمینان از پاک شدن دستور زیر را اجرا نمایید

```
db.system.replset.find();
```

برای ساخت کاربر اپلیکیشن از دستورات زیر استفاده نمایید.

```
db.createUser(
{
  user: "opnusr",
  pwd: "7ex4B1Gl",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDatabase" ]
}
)
```

```
db.grantRolesToUser("opnusr", [ { role: "userAdminAnyDatabase", db: "admin" } ])
db.grantRolesToUser("opnusr", [ { role: "clusterAdmin", db: "admin" } ])
```

## 2. Replica set در ویندوز

برای ایجاد replica set در دیتابیس به دو صورت میتوان عمل کرد. یکی به صورت اجرای دستی رپلیکیشن، و دیگری به صورت ایجاد رپلیکا ست به فرم سرویس در ویندوز. با توجه به اینکه اجرای دستی رپلیکیشن فقط تا زمان بالا بودن سیستم و اجرای بدون قطع اتصال اینستنس های mongod و mongos پایدار است، به توضیح مدل دوم (ایجاد سرویس) میپردازیم.

برای ایجاد replica set با سه نود در سرویس های ویندوز به ترتیب زیر عمل میکنیم:

1) ابتدا برای داده ها و لاگ های هر یک از نودها، یک مسیر مجزا ایجاد میکنیم:

```
mkdir -p E:/srv/mongodb/rs0-0 E:/srv/mongodb/rs0-1 E:/srv/mongodb/rs0-2
```

2) سپس برای هر نود یک فایل کانفیگ در مسیر نصب MongoDB به صورت دستی ایجاد میکنیم

-- در پوشه bin در مسیر نصب mongodb سه فایل با پسوند .cfg ایجاد کرده و اطلاعات زیر را با استفاده از notepad در هر یک وارد میکنیم:

```
logpath=<your          intended          log          path>\mongod.log
dbpath=<your          intended          data          path>
port=<your          intended          port          number>
replSet=<your intended replica set name>
```

#for example:

#the following information is for a Primary node:

```
logpath=E:\srv\mongodb\rs0-0\log\mongod.log
dbpath=E:\srv\mongodb\rs0-0\db
port=27017
replSet=rs0
```

#the following information is for a secondary node

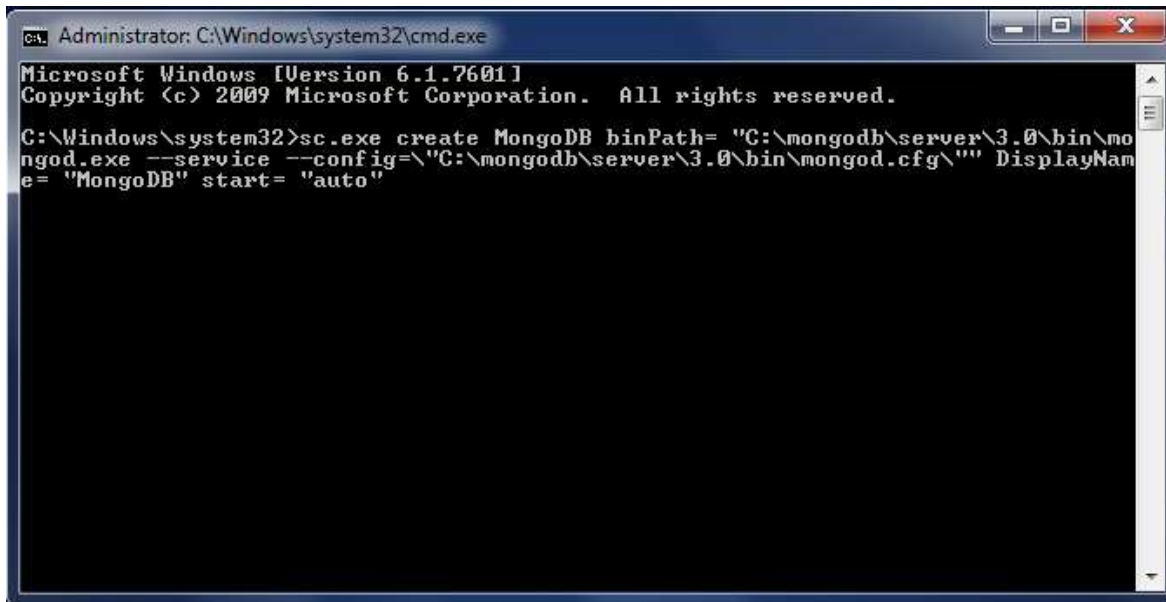
```
logpath=E:\srv\mongodb\rs0-1\log\mongod.log
dbpath=E:\srv\mongodb\rs0-1\db
port=27018
replSet=rs0
```

- توجه داشته باشید که قسمت replSet برای همه نودهای یک Replica Set باید یکسان باشد.

3) بعد از ایجاد فایل های کانفیگ، باید سرویس مناسب هر یک از این نودها را در محیط cmd با دسترسی administration با استفاده از کامندی مشابه زیر ایجاد کنیم:

```
sc.exe create MongoDB binPath= "<your Mongodb installation path>\bin\mongod.exe --service --config=\"<your Mongodb installation path>\bin\mongod.cfg\" DisplayName= "MongoDB" start= "auto"
```

برای مثال:



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>sc.exe create MongoDB binPath= "C:\mongodb\server\3.0\bin\mongod.exe --service --config=\"C:\mongodb\server\3.0\bin\mongod.cfg\" DisplayName= "MongoDB" start= "auto"
```

اگر سرویس با موفقیت ایجاد شده باشد شما پیام CreateService SUCCESS را دریافت میکنید.

4) سپس هر یک از سرویس های ایجاد شده را با دستور زیر استارت میکنیم:

```
net start <your service name>
```

برای مثال:

```
net start MongoDB
```

5) اکنون از پنل cmd به محیط shell در نود اصلی میرویم و replica set را با تعریف این نود به عنوان عضو اول و primary، ایجاد میکنیم:

```
<your Mongodb installation path>\bin\mongo --port <primary port number>
```

برای مثال:

```
C:\mongodb\server\3.0\bin\mongo --port 27017
```

سپس:

```
rsconf = {
  _id: "rs0",
  members:[
    {
      _id: 0,
      host: "<hostname>:27017"
    }
  ]
}
```

- بجای <hostname> باید host name سیستم خود را وارد کنیم.  
اکنون rsconf را به عنوان ورودی rs.initiate() قرار میدهیم:

```
rs.initiate( rsconf )
```

در این مرحله میتوانیم با کامند زیر، کانفیگوریشن رپلیکاستی که ایجاد کرده ایم را مشاهده کنیم:

```
rs.conf()
```

با زدن این کامند، اطلاعاتی شبیه به تصویر زیر به نمایش در خواهد آمد:

```
C:\Windows\system32\cmd.exe - mongo
2017-01-21T09:51:43.297+0330 I CONTROL  Hotfix KB2731284 or later update is not
installed, will zero-out data files
MongoDB shell version: 3.0.7
connecting to: test
RS0:STARTUP2> rs.conf(<
{
  "_id" : "RS0",
  "version" : 1,
  "members" : [
    {
      "_id" : 0,
      "host" : "ROKHIDEH:27017",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {
      },
      "slaveDelay" : 0,
      "votes" : 1
    }
  ],
  "settings" : {
    "chainingAllowed" : true,
    "heartbeatTimeoutSecs" : 10,
    "getLastErrorModes" : {
    },
    "getLastErrorDefaults" : {
      "w" : 1,
      "wtimeout" : 0
    }
  }
}
```

اکنون باید دو نود secondary خود را به replica set اضافه کنیم:

```
rs.add("<hostname>:<port number>")
rs.add("<hostname>:<port number>")
```

برای مثال:

```
rs.add("KHORRAMI:27018")
rs.add("KHORRAMI:27019")
```

در این مرحله کار به پایان رسیده و میتوان اطلاعات replica set را با کامند زیر مشاهده کرد:

```
rs.status()
```

```

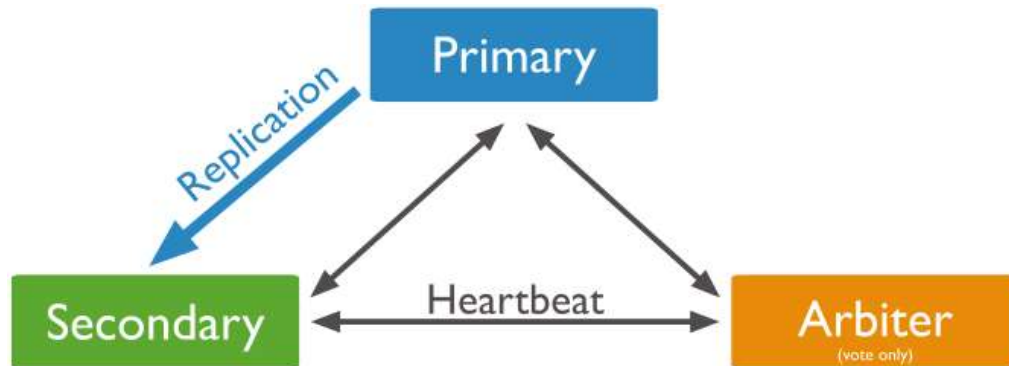
C:\Windows\system32\cmd.exe - mongo
RS0:PRIMARY> rs.status(<)
{
  "set" : "RS0",
  "date" : ISODate("2017-01-21T06:24:44.887Z"),
  "myState" : 1,
  "members" : [
    {
      "_id" : 0,
      "name" : "ROKHIDEH:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 555,
      "optime" : Timestamp(1484979870, 1),
      "optimeDate" : ISODate("2017-01-21T06:24:30Z"),
      "electionTime" : Timestamp(1484979748, 1),
      "electionDate" : ISODate("2017-01-21T06:22:28Z"),
      "configVersion" : 3,
      "self" : true
    },
    {
      "_id" : 1,
      "name" : "ROKHIDEH:27018",
      "health" : 1,
      "state" : 5,
      "stateStr" : "STARTUP2",
      "uptime" : 26,
      "optime" : Timestamp(0, 0),
      "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
      "lastHeartbeat" : ISODate("2017-01-21T06:24:44.050Z"),
      "lastHeartbeatRecv" : ISODate("2017-01-21T06:24:28.682Z"),
      "pingMs" : 0,
      "configVersion" : 2
    },
    {
      "_id" : 2,
      "name" : "ROKHIDEH:27019",
      "health" : 1,
      "state" : 5,
      "stateStr" : "STARTUP2",
      "uptime" : 14,
      "optime" : Timestamp(0, 0),
      "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
      "lastHeartbeat" : ISODate("2017-01-21T06:24:44.103Z"),
      "lastHeartbeatRecv" : ISODate("2017-01-21T06:24:44.461Z"),
      "pingMs" : 9,
      "configVersion" : 3
    }
  ],
  "ok" : 1
}

```

## Replica Set Arbiter

همچنین میتوان یک دیستنس دیگر تحت عنوان arbiter به replica set اضافه کرد. arbiter ها از هیچ دیتاستی نگهداری نمیکنند. هدف اصلی arbiter ها نگهداری حد نصاب در replica set به وسیله واکنش به heartbeat ها (پینگ هایی که اعضا هر دو ثانیه برای هم ارسال میکنند و از روی بازگشت آن متوجه در دسترس بودن نودهای دیگر میشوند) و election ها (الگوریتمی که تلاش میکند مقدم ترین نود ثانویه را در هنگام نیاز مشخص کند) میباشد که توسط اعضای replica set درخواست میشود. مزیت arbiter ها این است که چون از هیچ دیتاستی نگهداری نمیکنند، روش بهتر و کم هزینه تری نسبت به یک عضو ثانویه با تعداد زیادی دیتاست برای فراهم کردن نوبت و حدنصاب ها در replica set هستند. اگر replica set ما دارای

تعداد زوجی عضو باشد بهتر است با اضافه کردن یک arbiter، به انتخاب نود اصلی در هنگام دریافت election سرعت بیشتری ببخشیم. لازم به ذکر است که arbiter نیاز به سخت افزار اضافی ندارد.



### تمظیمات سایز Oplog size

برای محیط ویندوز و لینوکس حداقل فضای مورد نیاز برای inmemory 50 مگابایت و برای سایر مدل ها 990 مگابایت می باشد و حداکثر سایز 50 گیگا بایت می باشد.

برای سایز Oplog size

اضافه کردن arbiter به replica set :

1. ساختن دایرکتوری داده برای arbiter :

```
mkdir /data/arb
```

2. استارت کردن **arbiter**، تعیین دایرکتوری داده و نام **replica set** :

```
mongod --port 27020 --dbpath /data/arb --replSet rs0
```

3. کانکت شدن به نود اصلی و اضافه کردن **arbiter** به **replica set** :

```
rs.addArb("KHORRAMI:27020")
```

### رپلیکیشن Master Slave

در این مدل رپلیکیشن، مانند مدل **replica set**، میتوان تعداد زیادی نود غیر اصلی (**slave**) و تنها یک نود اصلی (**master**) وجود داشته باشد.

برای ایجاد رپلیکیشن **master-slave** ابتدا باید دو اینستنس **mongod** را استارت کنیم. یکی برای مود **master** و دیگری برای مود **slave**:

- برای استارت اینستنس **master** مطابق زیر عمل کنید:

ابتدا یک دایرکتوری برای نود **master** در مسیر دلخواه ایجاد میکنیم:

```
mkdir <your intended path>
```

**for example:** `mkdir e:\data\masterdb`

سپس با دستور زیر نود **master** را ایجاد میکنیم:

```
./mongod --dbpath <your master path> --port <your intended port number> --master
```

**for example:** `~\mongod.exe --dbpath e:\data\masterdb --port 10000 --master`



با آپشن `--master`، `mongod` یک کالکشن با نام `local.oplog.$main` میسازد که در آن، “oprations” log عملیاتی را که slave ها برای کپی کردن اعمال نود اصلی (master) انجام میدهند، لیست میکند.

- برای استارت اینستنس slave مطابق زیر عمل کنید:

ابتدا یک دایرکتوری برای نود slave ایجاد میکنیم:

```
mkdir <your intended path>
```

**for example:** `mkdir e:\dataSlave\db`

سپس با تعیین یک پورت جداگانه برای نود slave، این نود را با دستور زیر ایجاد میکنیم:

```
./mongod --dbpath <your slave path> --port <your intended port number> --slave  
--source localhost:<your master port number>
```

**for example:** `~\mongod.exe --dbpath e:\dataSlave\db --port 10001 --slave --source localhost:10000`

## Sharding

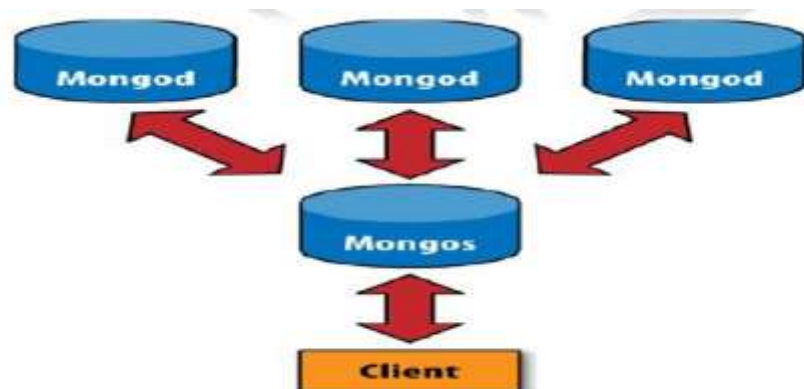
سیستم دیتابیس ها با حجم زیاد دیتا یا در مواجهه با حداکثر ظرفیت برنامه های اجرایی امکان دارد با مشکل مواجه شوند. به عنوان مثال با کوئری های با حجم زیاد (مصرف CPU بالا) یا استفاده از بیش از RAM یا مشکل داشتن زمان I/O همه از مواردی هستند که نشان دهنده ارتقا سیستم می باشند

برای توسعه سیستم دو روش وجود دارد

- توسعه عمودی در این روش منابع سیستم افزایش داده می شود مانند استفاده از CPU قدرتمندتر اضافه کردن RAM یا افزایش Storage.
- توسعه افقی در این روش منابع در سرور های مختلف تقسیم می شوند. برای افزایش منابع در صورت نیاز سرورهایی اضافه می گردند. زمانی که منابع سیستم زیاد نباشد و حجم کاری بالا باشد می توان از منابع سایر سرور ها استفاده نمود.

Sharding در اصطلاح همان پارتیشن بندی می باشد. Sharding این امکان را فراهم می کند تا اطلاعات را به بخش مختلف شکسته و در سیستم های جداگانه ذخیره کنیم. در واقع با chunks (تکه های کوچکتر از اطلاعات) کردن و قرار گرفتن در سیستم های جداگانه یا محل های متفاوت و استفاده از mongos می توانیم sharding را انجام دهیم.

در sharding ، mongod برای از mongos استفاده می کند مانند شکل زیر



از Sharding زمانی استفاده می شود که فضای دیسک در سرور کم باشد یا نیاز به نوشتن سری از داده ها باشد و یا بخواهیم مقدار زیادی از داده ها را در حافظه نگهداریم.

## Configuration File

در **mongodb** امکان پیکربندی موارد **mongod** و **mongos** برای استفاده در هنگام اجرای آن ها وجود دارد. فایل های کانفیگ شامل تنظیماتی هستند که معادل آپشن های کامند لاین **mongod** و **mongos** میباشد. در واقع استفاده از فایل های کانفیگ، مدیریت آپشن های **mongod** و **mongos** را آسان تر میکند. به خصوص در مواردی با مقیاس بزرگتر. همچنین میتوان در این فایلها یادداشت هایی برای توضیح بیشتر موارد استفاده شده قرار داد.

فایلهای کانفیگ در **mongodb** از فورمت **YAML** استفاده میکنند (**YAML** یک سوپرست از **JSON** است).

این فایل **tab** را پشتیبانی نمی کند و باید از **space** به جای آن استفاده نمود.

فایل نمونه زیر شامل تنظیمات زیادی است که ممکن است با فایل کانفیگ محلی شما منطبق یا مشابه باشد:

```
systemLog:
  destination: file
  path: "/var/log/mongodb/mongod.log"
  logAppend: true
storage:
  journal:
    enabled: true
processManagement:
  fork: true
net:
  bindIp: 127.0.0.1
  port: 27017
setParameter:
  enableLocalhostAuthBypass: false
...
```

اسکرپت های موجود در پکیج های اصلی **mongodb** در سیستم عامل لینوکس، شامل مقادیر مشخصی برای **systemLog.path**، **storage.dbpath** و **processManagement.fork** میباشد که اگر ما این تنظیمات را در فایل کانفیگ پیش فرض با تغییراتی قرار دهیم، ممکن است **mongod** استارت نشود.

استفاده از فایل کانفیگ:

برای استارت mongod و mongos با استفاده از فایل کانفیگ، باید این فایل را مانند نمونه زیر با آپشن --config یا -f مشخص کنیم:

```
mongod --config /etc/mongod.conf
```

```
mongos --config /etc/mongos.conf
```

```
mongod -f /etc/mongod.conf
```

```
mongos -f /etc/mongos.conf
```

### :Core Options

#### shard processManagement Options •

```
processManagement:  
  fork: <boolean>  
  pidFilePath: <string>
```

نام آپشن	نوع	پیش فرض	توضیحات
Fork	Boolean	False-0	با این گزینه میتوان حالت daemon را برای اجرای mongod و mongos در پس زمینه انتخاب کرد. بطور پیش فرض این دو مورد به عنوان daemon اجرا نمیشوند. همچنین در صورت استفاده از پکیج های لینوکس، اگر در این گزینه تغییری ایجاد نمایید ممکن است اسکریپت های اصلی و پیش فرض پکیج به درستی کار نکنند. به همین دلیل در این صورت باید از init script خود استفاده کرده و اسکریپت های built-in را غیر فعال کنید.
pidFilePath	String	-	مسیری برای نگهداری PID های mongod و mongos مشخص میکند. این آپشن به همراه آپشن fork برای ردیابی بهتر پردازش های mongod و mongos استفاده میشود. بدون تعیین این آپشن، پردازش ها بدون فایل PID ایجاد میشوند.

برای mongo روی cloud آپشن هایی وجود دارد به دلیل عدم استفاده از cloud اسکیپ می شود

### • security Options

```
security:
  keyFile: <string>
  clusterAuthMode: <string>
  authorization: <string>
  transitionToAuth: <boolean>
  javascriptEnabled: <boolean>
  redactClientLogData: <boolean>
  sasl:
    hostname: <string>
    serviceName: <string>
    saslauthdSocketPath: <string>
  enableEncryption: <boolean>
  encryptionCipherMode: <string>
  encryptionKeyFile: <string>
  kmip:
    keyIdentifier: <string>
    rotateMasterKey: <boolean>
    serverName: <string>
    port: <string>
    clientCertificateFile: <string>
    clientCertificatePassword: <string>
    serverCAFile: <string>
  ldap:
    servers: <string>
    bind:
      method: <string>
      saslMechanism: <string>
      queryUser: <string>
      queryPassword: <string>
      useOSDefaults: <boolean>
      transportSecurity: <string>
      timeoutMS: <int>
      userToDNMapping: <string>
```

```
authz:
  queryTemplate: <string>
```

نام آپشن	نوع	پیش فرض	توضیحات
keyFile	String	-	مسیری برای key file ها تعیین میکند که رمزهای اشتراک گذاری شده mongodb را که برای اعتبار دادن به sharded cluster و replica set استفاده میشود، ذخیره میکنند.
clusterAuthMode	String	keyFile	این حالت برای cluster authentication استفاده میشود. این آپشن میتواند موارد keyFile ، sendKeyFile ، sendX509 و x509 را به عنوان ورودی بپذیرد.
Authorization	String	Disabled	برای کنترل دسترسی هر یک از کاربران برای database و resource و operation میباشد. با این مقدار پیش فرض، یک کاربر میتواند به هر دیتابیس دسترسی داشته و هر کاری را انجام بدهد.
javascriptEnabled	Boolean	True	فعال یا غیر فعال سازی اجرای JavaScript در سمت سرور به وسیله این گزینه امکان پذیر است. وقتی غیر فعال باشد، نمیتوان عملی که اجرای جاوا اسکریپت در سمت سرور را انجام میدهند را استفاده کرد. عملی نظیر \$where یا کامند mapReduce یا group

## • storage Options

```
storage:
  dbPath: <string>
  indexBuildRetry: <boolean>
  repairPath: <string>
  journal:
    enabled: <boolean>
    commitIntervalMs: <num>
  directoryPerDB: <boolean>
  syncPeriodSecs: <int>
  engine: <string>
```

```

mmapv1:
  preallocDataFiles: <boolean>
  nsSize: <int>
  quota:
    enforced: <boolean>
    maxFilesPerDB: <int>
  smallFiles: <boolean>
  journal:
    debugFlags: <int>
    commitIntervalMs: <num>
wiredTiger:
  engineConfig:
    cacheSizeGB: <number>
    journalCompressor: <string>
    directoryForIndexes: <boolean>
  collectionConfig:
    blockCompressor: <string>
  indexConfig:
    prefixCompression: <boolean>
inMemory:
  engineConfig:
    inMemorySizeGB: <number>

```

نام آپشن	نوع	پیش فرض	توضیحات
dbpath	String	-	مشخص کننده مسیری که mongod داده های خود را در آن ذخیره میکند. این آپشن فقط برای mongod در دسترس میباشد.
indexBuildRetry	Boolean	True-1	مشخص میکند که mongod ایندکس های ناقص را در استارت آپ بعدی در کجا بازسازی (rebuild) کند؟ این آپشن در مواردی که mongod حین ساخت یک ایندکس خاموش شده یا استاپ میشود، کاربرد دارد. برای جلوگیری از rebuild ایندکس ها توسط mongod باید این گزینه را در حالت false-0 قرار دهید.
repairPath	String	A _tmp_repairDatabase_<num> directory under the dbPath	مشخص کننده مسیر کاری ای که MongoDB در حین عملیات --repair از آن استفاده میکند. وقتی --repair کامل میشود، این مسیر خالی بوده

و dbpath شامل repaired file ها میشود.			
فعال یا غیر فعال کردن ماندگاری journal برای اطمینان از معتبر بودن و برگشت پذیر بودن فایل های داده از طریق این گزینه امکان پذیر است. این آپشن تنها زمانی که --dbpath مشخص شده باشد اعمال میشود. همچنین این آپشن تنها برای mongod در دسترس است. توجه شود اگر هر یک از اعضای voting یک رپلیکاست بدون journaling اجرا شود، باید writeConcernMajorityJournalDefault را برابر false قرار دهید.	True در سیستمهای 64 بیتی و false در سیستمهای 32 بیتی	Boolean	journal.enabled
اگر این آپشن true باشد MongoDB از یک مسیر مجزا برای ذخیره داده های هر دیتابیس استفاده میکند. هر دایرکتوری یک نام مرتبط با نام دیتابیس داشته و در زیر مجموعه مسیر dbpath قرار میگیرد.	False-0	Boolean	directoryPerDB
ماشین ذخیره سازی دیتابیس های mongod که شامل موارد mmapv1 ، wiredTiger ، inMemory میباشد.	wiredTiger	string	engine

## • storage.mmapv1 Options

```
storage:
  mmapv1:
    preallocDataFiles: <boolean>
    nsSize: <int>
    quota:
      enforced: <boolean>
      maxFilesPerDB: <int>
    smallFiles: <boolean>
    journal:
      debugFlags: <int>
      commitIntervalMs: <num>
```

نام آپشن	نوع	پیش فرض	توضیحات
preallocDataFiles	Boolean	True-1	فعال یا غیر فعال کردن توضیحات یا



preallocation فایل های داده			
سایز فایل های namespace را معین میکند. هر کالکشن و ایندکس به عنوان یک namespace برشمرده میشود. ماکزیمم سایز هر namespace 2024 مگابایت است.	16	Int	nsSize
فعال یا غیر فعال سازی اجرای محدودیت حداکثری برای تعداد فایل های داده ای که یک دیتابیس میتواند داشته باشد. در صورت فعال بودن این آپشن، MongoDB حداکثر دارای 8 فایل داده به ازای هر دیتابیس میشود.	False-0	Boolean	quota.enforced
محدودیت تعداد فایل های داده به ازای هر دیتابیس. برای استفاده از این آپشن باید quota.enforced هم فعال شود	8	Int	quota.maxFilesPerDB
در صورت فعال بودن MongoDB از سایز پیش فرض کوچکتری استفاده میکند. این گزینه سایز اولیه فایل های داده را کاهش داده و ماکزیمم آن را برابر 512 مگابایت قرار میدهد. همچنین سایز هر journal را از یک گیگابایت به 128 مگابایت کاهش میدهد.	False-0	Boolean	smallFiles
این گزینه بیشتر برای تست استفاده میشود نه برای استفاده عمومی. این گزینه بر روی درستی data file در هنگام خاموشی غیر طبیعی سیستم تاثیر میگذارد.	-	Int	journal.debugFlags

## storage.WiredTiger Options •

```
storage:
  wiredTiger:
    engineConfig:
      cacheSizeGB: <number>
      journalCompressor: <string>
```

```

    directoryForIndexes: <boolean>
collectionConfig:
    blockCompressor: <string>
indexConfig:
    prefixCompression: <boolean>

```

نام آپشن	نوع	پیش فرض	توضیحات
engineConfig.cacheSizeGB	Float	-	بیشترین مقدار cache داخلی که WiredTiger برای همه داده ها استفاده میکند.
engineConfig.journalCompressor	String	Snappy	نوعی از فشرده سازی که میتوان از آن برای فشرده کردن داده journal در WiredTiger استفاده کرد. مقادیر قابل استفاده شامل zlib ، snappy ، none میشود.
engineConfig.directoryForIndexes	Boolean	False-0	وقتی این گزینه فعال باشد mongod ایندکس ها و کالکشن ها را در مسیرهای جداگانه ای با نام index و collection در زیر مسیر dbpath قرار میدهد.
collectionConfig.blockCompressor	String	Snappy	نوع پیش فرض فشرده سازی داده کالکشن ها. مقادیر قابل استفاده این آپشن شامل none ، snappy ، zlib میباشد. توجه داشته باشید که این گزینه فقط بر روی کالکشن های تازه ساخته شده اثر میگذارد و در صورت تغییر، کالکشن های قبلی به شیوه ای که زمان ساختشان تعیین شده است فشرده میشوند.
indexConfig.prefixCompression	Boolean	True-1	فعال یا غیر فعال کردن prefix compression برای داده ایندکس ها. این گزینه فقط بر روی ایندکس های تازه ساخته شده اثر

میگذارد. در مورد ایندکس هایی که قبل از تغییر این گزینه ساخته شده اند، این گزینه اثرگذار نخواهد بود.			
---	--	--	--

### • storage.inmemory Options

```
storage:
  inMemory:
    engineConfig:
      inMemorySizeGB: <number>
```

نام آپشن	نوع	پیش فرض	توضیحات
engineConfig.inMemorySizeGB	Float	50% از رم فیزیکی کمتر از 1 گیگابایت	بیشترین مقدار مموری اختصاص داده شده برای داده های in-memory storage engine شامل ایندکس ها، oplog اگر mongod جزو رپلیکاست باشد و رپلیکاست یا sharded cluster metadata و ... این گزینه فقط در نسخه Enterprise مונگو دی بی در دسترس میباشد.

### • operationProfiling Options

```
operationProfiling:
  slowOpThresholdMs: <int>
  mode: <string>
```

نام آپشن	نوع	پیش فرض	توضیحات
slowOpThresholdMs	Int	100	
mode	String	Off	مشخص کننده سطح database profiling که اطلاعاتی را درباره کارایی operation درون کالکشن system.profile وارد میکند. مقادیر قابل استفاده برای این گزینه off (برای هیچ یک از عملگرها) ، slowOp (فقط شامل عملگرهای

slow) و all (برای همه عملگرها) میباشد.

## • replication Options

```
replication:
  oplogSizeMB: <int>
  replSetName: <string>
  secondaryIndexPrefetch: <string>
  enableMajorityReadConcern: <boolean>
```

نام آپشن	نوع	پیش فرض	توضیحات
oplogSizeMB	Int	-	بیشترین اندازه log operation رپلیکیشن را مشخص میکند. mongod یک oplog بر اساس بیشترین مقدار فضای در دسترس ایجاد میکند. برای سیستمهای 64 بیتی اندازه oplog معمولاً 5٪ از disk space در دسترس میباشد. اگر بعد از ایجاد oplog توسط mongod این گزینه را تغییر دهید، تغییری در اندازه آن ایجاد نخواهد شد.
replSetName	String	-	مشخص کننده نام رپلیکاست است که همه نودهای آن باید از این نام استفاده کنند.
secondaryIndexPrefetch	String	All	این گزینه فقط در ماشین ذخیره سازی mmapv1 قابل استفاده است. ایندکس هایی که اعضای secondary یک رپلیکاست قبل از اعمال operation ها از oplog درون مموری لود میکنند. مقادیر قابل استفاده در این آپشن شامل none ، all ، _id_only میشود.

diagnosticDataCollectionDirectoryPath

<https://docs.mongodb.com/manual/reference/parameters/>

## Performance

زمانی که برنامه در حال توسعه می باشد آنالیز کارایی برنامه و دیتابیس اهمیت بالایی دارد که می تواند شامل استراتژی دسترسی به دیتابیس ، سخت افزار موجود و تعداد کانکشن های به دیتابیس باشد.

برای بررسی کارایی در MongoDB میتوان موارد زیر را مورد توجه قرار داد

1- Locking performance

2- Memory and MMAPv1 storage engine

3- Number of Connections

4- Database Profiling

### Locking Performance

Mongodb از سیستم لاکینگ برای سازگاری داده استفاده می کند. زمانی که عملیاتی زمان زیادی در حال اجرا باشد و یا در صف باشد اولویت اجرای درخواست ها و عملیات های در انتظار لاک کاهش می یابد.

با استفاده از کامند (`db.serverStatus()`) می توان وضعیت لاک های سیستم را در `locks` و `globalLock` مشاهده نمود. برای محاسبه میانگین زمان انتظار باید `locks.timeAcquiringMicros` را بر `locks.acquireWaitCount` تقسیم نمود.

`locks.deadlockCount` مشخص می کند چه تعداد `deadlock` داشته ایم.

در صورتی که `globalLock.currentQueue.total` مقدار بالایی داشته باشند نشان دهنده این می باشد که تعداد زیادی از عملیات ها در انتظار `lock` می باشند و کارایی مناسبی می باشد.

در صورتی که مقدار `globalLock.totalTime` بالا باشد نشان دهنده این می باشد که دیتابیس به صورت عمده در وضعیت `lock` می باشد.

## Memory and the MMAPv1 Storage Engine

### Memory Use

Storage engine اصلی برای Mongoddb قبل از ورژن 3.2 می باشد مناسب برای خواندن و نوشتن زیاد با update های زیاد می باشد.

Storage engine In-memory در ورژن Mongoddb Enterprise می باشد. که داکيومنت ها را علاوه بر دیسک در memory برای پاسخگویی و پیش بینی بهتر نگهداری می کند.

### Number of Connections

....

### Database Prtofiling

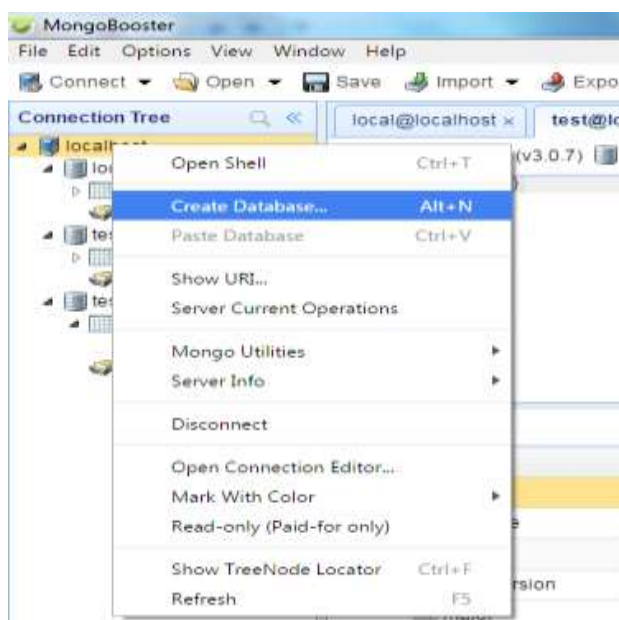
....

## Mongobooster

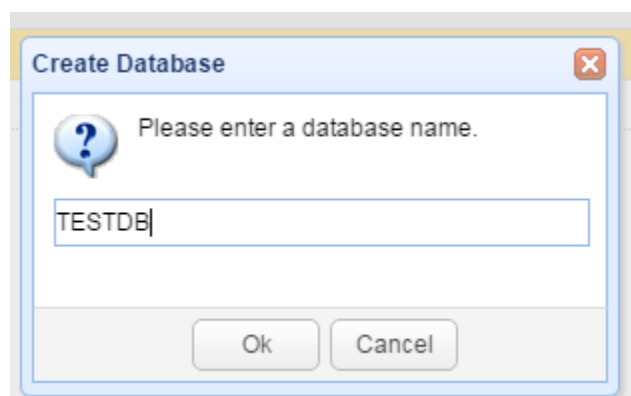
ابزارهای مختلفی برای کار با MongoDB وجود دارد یکی از این ابزار ها mongobooster می باشد در ادامه به صورت گرافیکی ساخت آبجکت ها با این ابزار را کار می کنیم.

### ساخت دیتابیس با Mongobooster :

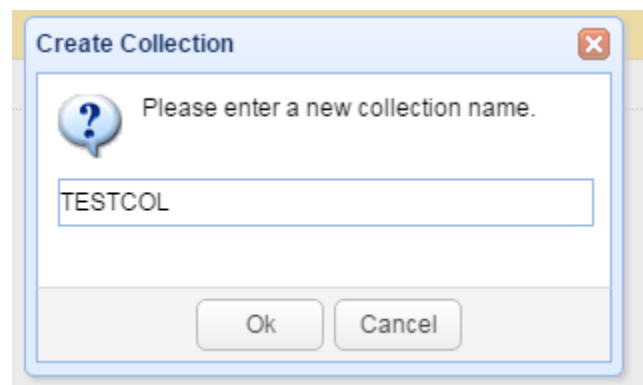
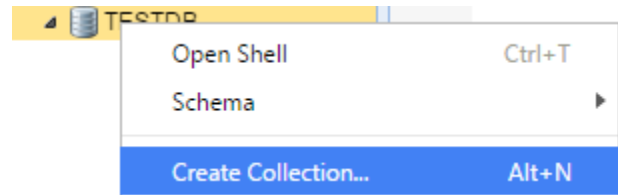
برای ساخت دیتابیس به صورت زیر عمل می کنیم .



در کامند باز شده نام دیتابیس را وارد می کنیم ( MongoDB ، به کوچک یا بزرگ بودن حروف حساس می باشد)



برای ساخت کالکشن به روش زیر عمل می کنیم

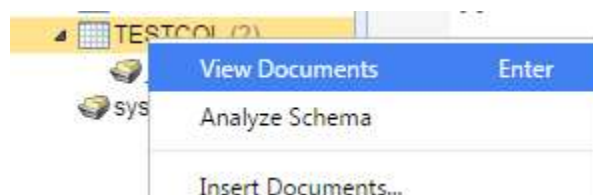


برای درج داکيومنت (رکورد) از دستور زیر استفاده می کنیم

```
db.TESTCOL.insert([{"user_id": "abc456,"  
  age: 5,  
  status: "B"  
}])
```

برای نمایش داکيومنت های درج شده از روش زیر استفاده می کنیم





خروجی به شکل زیر خواهد بود

Key	Value	Type
(1) ObjectId("5858f0aee80160b5c78e47b") ( 4 fields )		Document
_id	ObjectId("5858f0aee80160b5c78e47b")	ObjectId
user_id	abc123	String
age	55	Double
status	A	String
(2) ObjectId("5858f0aee80160b5c78e47d") ( 4 fields )		Document

## دیتابیس های موجود در MongoDB

### دیتابیس Admin

دیتابیس ریشه است و شامل authentication می باشد

### دیتابیس Local

برای نگهداری آبجکت های سرور لوکال می باشد و یک بار ساخته می شود.

### دیتابیس Config

برای نگهداری namespace ها می باشد (مجموعه **blog.spot** در دیتابیس **cms** قرار دارد پس **namespace** ، **cms.blog.spot** را تشکیل می دهد)

<https://docs.mongodb.com/>

<https://docs.mongodb.com/manual/>

<https://admin-docs.com/databases/mongodb/mongodb-installation/install-mongodb-on-centos-7-64-bit/>

<https://docs.mongodb.com/manual/core/journaling/#journaling-record-write-operation>