



mongoDB

## MongoDB

آشنایی با پایگاه داده سندگرا

مهندس معصومه ابهامی

# فهرست

مقدمه

فصل اول

معرفی MongoDB

فصل دوم

مفاهیم موجود در MongoDB

فصل سوم

شروع کار با MongoDB

فصل چهارم

پرس و جوها

فصل پنجم

Sharding

فصل ششم

Aggregation ها

فصل هفتم

ایندکس گذاری

فصل هشتم

مدیریت در پایگاد داده ی MongoDB

فصل نهم

Replication ها

ضمیمه ی یک

نصب MongoDB

ضمیمه ی دو

آشنایی با shell پایگاه داده

ضمیمه ی سه

Bson-

Wire Protocol-

Data Files-

Namespaces and Extents-

با احترام تقدیم به خانواده ی بزرگ علم و ادب

این نسخه با وجود کاستی های زیاد در خدمت شما عزیزان قرار گرفته است که امیدوارم روزنه ای باشد برای ورود به دنیای بزرگ پایگاه داده های سند گرا و ادامه ی راه نگارش نسخه های بالاتر و بهتر.

همانطور که هیچ نوشته ای خالی از عیب و ایراد نیست بنابراین بنده را مفتخر می نمایید اگر در یافتن کاستی و عیب های این نوشته مرا کمک نمایید

ما با تلاش سعی در رفع نواقص داریم تا بتوانیم نوشته ای درخور لیاقت شما عزیزان را به شما تقدیم نماییم

با تشکر

معصومه ابهامی

زمستان 91-92

# فصل اول

## معرفی mongoDB

mongoDB یک پایگاه داده قوی و منعطف و مقیاس پذیر است. این پایگاه داده توانایی scale out کردن و خیلی از ویژگی های پایگاه داده های رابطه ای مثل ایندکس گذاری و دامنه ی پرس و جوها و ذخیره سازی را دارا می باشد.

mongoDB به طور باور نکردنی پر ویژگی است. خیلی از ویژگی ها مثل built-in بودن پشتیبانی کردن از MapReduce-style aggregation و MapReduce-style aggregation را دارا می باشد که بعدا در مورد هر کدام مطالبی را خواهیم آموخت.

mongoDB یک مدل داده ای Developer پسند و دارای پیکربندی administrator پسند است

و همچنین دارای API های زبان های عمومی بوسیله ی Driver ها و shell پایگاه داده است.

mongoDB تلاش می کند که راه شما را به سوی برنامه نویسی بدون نگرانی از مشکل ذخیره داده ها تغییر بدهد

mongoDB یک مدل داده ای غنی

mongoDB یک پایگاه داده سندگرا است نه یک پایگاه داده ی رابطه ای. دلیل اصلی برای کنار گذاشتن دیتابیس های رابطه ای توانایی scale out بودن زبان mongoDB است اما سایر مزایای دیگر را نیز دارد.

ایده ی پایه در طراحی mongoDB جایگزین کردن مفهوم row با یک مدل انعطاف پذیرتر به نام document است سندگرایی امکان ایجاد سند های داخلی و آرایه ها را به کاربر می دهد

و همچنین scheme-less یا همون بدون شما بودن دیتابیس را نیز دارا می باشد.

## Easy scaling

با توجه به سرعت رشد حجم داده ها و پیشروی در این تکنولوژی و حجم اطلاعاتی که در پایگاه داده نیاز به ذخیره سازی دارند برای اداره کردن این اطلاعات زیاد مدیریت جامعی احساس می شود.

چگونه پایگاه داده خود را مقیاس پذیر کنیم؟

برای مقیاس پذیر کردن پایگاه داده به دو گزینه می رسیم

1-scaling up (بزرگ کردن ماشین ها)

2-scaling out (پارتیشن بندی کردن در بین چندین ماشین)

از آنجایی که ایجاد ماشین بزرگ مقرون به صرفه نیست همچنین نیاز به فضای بزرگتری دارد scaling out کردن توسعه پذیرتر و اقتصادی تر است که برای اضافه کردن و بالا بردن کارایی می توان یک سرور مناسب خوب خریداری کرد و به مجموعه خود اضافه کرد. (بعدها در این کتاب به این مجموعه کلاستر نیز گفته می شود)

mongoDB بر پایه ی scale out بودن طراحی شده است سندگرا بودن مدل داده ها این اجازه را می دهد که داده ها به طور جداگانه در روی چندین سرور پخش شوند که داده ها و لود کردن یک گروه را به تعادل می رسونه. توزیع مجدد دسته ها اتوماتیک است که این امکان را می دهد که برنامه نویسان بدون نگرانی از ذخیره داده ها روی برنامه نویسی تمرکز کنند و برای افزایش ظرفیت داده ها تنها نیاز به تهیه ی یک سرور داشته باشند و به آسانی مشکل اینکه چگونه داده ها را ذخیره کنیم حل می شود.

ویژگی ها

مشکل است که ویژگی ها را در mongo چگونه بیان کنیم. در مقایسه با پایگاه داده های رابطه ای یا در مقایسه با دیگر پایگاه داده های سندگرای دیگر؟

خلاصه حرف این است که mongoDB واقعا خوب است و ابزارهای منحصر به فردی دارد که دیگر همتهای رقیبش ندارند.

Indexing: generic secondary indexes و پرس و جوهای سریع و یکتایی و compound و قابلیت geospatial indexing را به خوبی پشتیبانی می کند.

## انباشته ی JavaScript

بجای ذخیره سازی رویه ها تولید کننده ها می توانند علاوه بر ذخیره سازی از توابع و value ها در سمت سرور استفاده کنند.

## Aggregation

از MapReduce و دیگر ابزارهای Aggregation پشتیبانی می کند.

## Fixed-size collections

Capped collections دارای اندازه ثابتی هستند که این ویژگی مفید است برای نوع مشخصی از داده ها مثل logs.

## File storage

mongoDB از یک پروتکل ساده برای ذخیره سازی داده ها استفاده می کند.

بعضی از ویژگی ها با پایگاه داده های رابطه ای مشترک است اما بعضی در mongoDB وجود ندارد مثل join ها یا تراکنش های پیچیده چند سطری. زیرا پیاده سازی این ویژگی ها در سیستم های توزیع یافته مشکل است.

کارایی بالا بدون اتلاف سرعت

کارایی باورنکردنی هدف والای mongoDB است و از زمان طراحی به اجرا درآمده است که از پروتکل های سیمی باینری به عنوان مود اصلی برای فعل و انفعالات با سرور استفاده می کند.

اگرچه mongoDB خیلی قدرتمند است و سعی در نگه داشتن خیلی از ویژگی های پایگاه داده های رابطه ای را داشته ولی انتخاب خوبی برای انجام هرچیزی که توسط پایگاه داده های رابطه ای انجام می شود نیست.

در هر زمانی ممکن است پردازش به صورت offload و منطقی سمت کلاینت انجام گیرد که این طراحی سبب بالا بردن کارایی mongoDB شده است.

علاوه بر راه اندازی سرور پایگاه داده نیاز به خیلی از مدیریت های ریز داریم اگر یک سرور master خاموش شود به صورت اتوماتیک یک سرور slave جایگزین سرور master می شود.

فلسفه ی مدیریت در mongoDB این است که سرور باید اداره شود تا حد امکان و پیکربندی اتوماتیک این اجازه را می دهد که کاربران اتصالاتشان را در صورت نیاز بالا ببرند.



# فصل دوم

## مفاهیم موجود در

## mongoDB

در این فصل به بررسی مفاهیم موجود در MongoDB می پردازیم  
Document (سند) یک واحد پایه برای داده ها است و مفهومی شبیه ردیف در پایگاه داده های رابطه ای است

Collection (مجموعه) می توان برابر یک جدول بدون شما هست  
یک مثال از MongoDB می تواند یک گروه مستثل از پایگاه داده ها باشد که هر کدام مجموعه ها و دسترسی های خاص خود را دارد.

MongoDB از یک پوسته ی ساده ی JavaScript بوجود آمده است  
هر سند دارای یک کلید مخصوص "\_id" که در بین سند های یک مجموعه یکتا هست.

## Documents

در جاوا اسکریپت سند ها به عنوان یک object بیان می شود

```
{ "greeting" : "hello!" }
```

در این سند ساده که محتوی یک کلید به نام greeting با ارزش hello! است. اسناد می تواند پیچیده باشد مثل

```
{ "greeting" : "hello!" , "number" : 3 }
```

این مثال به خوبی چندین مفهوم را بیان می کند

1. ترکیب کلید و ارزشش در سند به ترتیب هستند بنابراین دو سند زیر باهم تفاوت دارند

```
{ "number" : 3, "greeting" : "hello!" }
```

{“greeting” : “hello!” , “number” : 3}

2. مقدارها درون یک سند ممکن است چندین نوع باشد مثلاً در مثال بالا هم نوع رشته داریم و هم نوع عدد.

3. کلید ها نباید دارای کاراکتر null باشند(0\)

4. کلید های که با \_ شروع می شوند به عنوان کلید های رزو شده مطرح می شوند اگرچه اجباری در این مورد وجود ندارد.

mongoDB type-sensitive و case-sensitive است بدین معنی که اسناد زیر باهم تفاوت دارند

{"foo" : 3}  
{"foo" : "3"}

{"foo" : 3}  
{"Foo" : 3}

نکته ی مهم و نهایی:

سندها در mongoDB نمی توانند شامل کلید های تکراری باشند. با این حساب سند زیر غیرقانونی است

{“greeting” : “hello!”, “greeting” : “hello word!”}

## نام گذاری مجموعه ها

برای نام گذاری می توان از فرمت utf-8 با یکسری از محدودیت ها استفاده کرد

- 1- رشته ی خالی ("" ) نباشد
- 2- شامل کاراکتر null نباشد(\0)
- 3- نباید دارای پیشوند system باشد زیرا این نام گذاری برای مجموعه های سیستمی رزو شده است مانند system.users شامل یوزرهای پایگاه داده است.
- 4- مجموعه هایی که توسط کاربر تعریف می شود نباید شامل کاراکتر \$ باشد

## پایگاه داده

مجموعه ای از مجموعه ها تشکیل یک پایگاه داده را می دهد. هر پایگاه داده در فایل جداگانه با دسترسی های خاص خود ذخیره می شود.

قوانین نام گذاری پایگاه داده

- 1- نباید شامل رشته تهی ("" ) باشد
- 2- نباید دارای کاراکترهای ( " ) و . و \$ و / و \ و یا کاراکتر null باشد
- 3- باید از حروف کوچک استفاده شود
- 4- ماکزیمم نام پایگاه داده باید 64 بایت باشد

دیتابیس های رزو شده در mongoDB که می توانیم به آنها دسترسی مستقیم داشته باشیم شامل:

Admin

دیتابیس ریشه است که شامل authentication ها می باشد اگر یک یوزر به admin اضافه شود به طور اتوماتیک دسترسی بر authentication ها را به ارث می برد.

## Local

این پایگاه داده دوباره ساخته نمی شود و برای ذخیره سازی هر مجموعه ای که باید روی یک سرور منفرد local باشد استفاده می گردد.

## Config

در سمت sharded و ذخیره سازی اطلاعات در مورد shard ها استفاده می شود.

زمانی که اسم پایگاه داده را همراه با اسم مجموعه می نویسیم یک فضای نام جامعی را تولید می کنیم که namespace نام دارد برای مثال مجموعه ی blog.post در پایگاه داده ی cms وجود دارد که cms.blog.post یک فضای نام می شود که این فضای نام دارای طول حداکثر 121 بایت و حداقل 100 بایت باشد.

فصل سوم

شروع کار با MongoDB

## شرح کار با MongoDB

برای کار با پوسته mongoDB ابتدا وارد مسیری که mongoDB را نصب کرده ایم می شویم سپس در cmd دستور mongod را اجرا می کنیم تا سرور دیتابیس شروع بکار کند cmd دیگری را باز کرده و mongo می نویسیم به این نکته توجه کنید که همواره برای بستن سرور از کلید ترکیبی ctrl و c استفاده می نمایم

The image shows a Windows Command Prompt window titled "Command Prompt - mongod". The output displays the MongoDB service starting on November 12, 2013, at 22:52:38. It indicates that MongoDB is running in 32-bit mode and provides a note about the 2 GB data limit for 32-bit versions. The logs also show the database version (v1.8.1) and the path to the data directory.

Below the main window, a smaller "Command Prompt - mongo" window is open, showing the MongoDB shell interface. It displays the shell version (1.8.1) and the connection to the 'test' database.

```

C:\> Command Prompt - mongod
Mon Nov 12 22:52:38 [initandlisten] MongoDB starting : pid=900 port=27017 dbpath
=/data/db/ 32-bit
*** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data
*** see http://blog.mongodb.org/post/137788967/32-bit-limitations
*** with --dur, the limit is lower
Mon Nov 12 22:52:39 [initandlisten] db version v1.8.1, pdfile version 4.5
Mon Nov 12 22:52:39 [initandlisten] git version: a429cd4f535b2499cc4130b06ff7c26
f41c00f04
Mon Nov 12 22:52:39 [initandlisten] build options: gcc version 4.5.2 32-bit
Mon Nov 12 22:52:39 [initandlisten] system info: {
  "processor": "x86_32",
  "os": "Windows",
  "version": "1.8.1",
  "modules": []
}
Mon Nov 12 22:52:39 [initandlisten] done
C:\> Command Prompt - mongo
> ^C
C:\Users\ada> mongo
MongoDB shell version: 1.8.1
connecting to: test
>
  
```

به صورت پیش فرض به پایگاه داده test متصل می شویم برای سوئیچ کردن به یک پایگاه داده ی دیگر از دستور

```
> use tutorial
```

switched to db tutorial

استفاده می کنیم برای مثال در دستر بالا به دیتابیس tutorial منتقل شده ایم

یک اتفاق جالب در MongoDB این است که ما به دیتابیس tutorial سوئیچ کرده ایم بدون اینکه آن را ایجاد کنیم. در حقیقت نیازی نیست که ما دیتابیس را ایجاد را کنیم زیرا دیتابیس و collection ها در موقع وارد کرده اولین document به صورت runTime ساخته می شوند

برای مثال فرض کنید که ما می خواهیم یک سند مثل سند زیر برای توصیف کاربر ایجاد کنیم

```
{username: "ashli"}
```

دستورات زیر را به ترتیب دنبال می کنیم

برای وارد کردن اولین سند خود به یک مجموعه نیاز داریم که با دستور زیر اولین مجموعه و اولین سند خود را ایجاد می کنیم

```
> db.users.insert({username: "ashli"})
```

اگر دقت کنیم پس از وارد کردن دستور و اجرای آن یک تاخیر ناچیزی را احساس می کنیم . این تاخیر ناشی از زمان لازم برای ایجاد دیتابیس و مجموعه و فضا دهی به این دو در روی دیسک می باشد.

پس از درست وارد کردن سند با دستور زیر مطمئن می شویم که سند به درستی ذخیره شده است

```
> db.users.find()
```

پس از اجرا , نوشته ی زیر در خروجی ظاهر می گردد

```
{ _id : ObjectId("4bf9bec50e32f82523389314"),  
  username : "smith" }
```

اگر دقت کنیم, خواهیم دید که یک `_id` به سند ما اضافه می شود که این شناسه حکم کلید اصلی را برای ما دارد. در حقیقت در زمان ایجاد هر سند, یک شناسه به سند ما به صورت خودکار اضافه می گردد. این شناسه در بین سندهای یک مجموعه, یکتاست.

برای اضافه کردن یک سند جدید به مجموعه ی خود طبق مراحل زیر داریم

```
> db.users.save({username: "jones"})
```

حالا باید در مجموعه ی ما دو سند وجود داشته باشد

```
> db.users.count()  
2
```

برای مشاهده ی تمام اسناد موجود در مجموعه داریم:

```
> db.users.find()
```

```
{ _id : ObjectId("4bf9bec50e32f82523389314"), username : "ashli" }  
{ _id : ObjectId("4bf9bec90e32f82523389315"), username : "jones" }
```

همچنین با استفاده از دستور ساده زیر اقدام به انتخاب یک سند حاوی نام کاربری برابر `jones` در مجموعه ی مورد نظر می کنیم

```
> db.users.find({username: "jones"})  
{ _id : ObjectId("4bf9bec90e32f82523389315"), username : "jones" }
```

بروز رسانی

در همه ی عملیات `update` نیاز به دو آرگومان داریم.

آرگومان اول که سند مورد نظر برای آپدیت را مشخص می کند و آرگومان دوم مشخص می کند چگونه سند خود را بروز کنیم

برای مثال قصد داریم برای کاربر `ashli` مورد `country` را نیز اضافه کنیم

```
> db.users.update({username: "ashli"}, {$set: {country: "Canada"}})
```

حال اگر دستور زیر را اجرا کنیم مشاهده می شود که به سند حاوی نام کاربری `ashli` موردی بنام `country` نیز اضافه شده است

```
> db.users.find({username: "ashli"})
{ "_id" : ObjectId("4bf9ec440e32f82523389316"),
  "country" : "Canada", username : "ashli" }
```

حال اگر بخواهیم برای کاربر مورد نظر گزینه ی country را حذف کنیم کافی ست دستور زیر را اجرا نماییم

```
> db.users.update({username: " ashli "}, {$unset: {country: 1}})
```

برای ساختارهای پیچیده تر مانند ذخیره سازی پروفایل یک کاربر که دارای لیستی از favorites است داریم :

```
{ username: " ashli ",
  favorites: {
    cities: ["Chicago", "Cheyenne"],
    movies: ["Casablanca", "For a Few Dollars More", "The Sting"] }
}
```

همچنین می توان بجای از نو نوشتن سند از دستور زیر استفاده کرد

```
> db.users.update( {username: " ashli "},
{ $set: {favorites:
{
cities: ["Chicago", "Cheyenne"],
movies: ["Casablanca", "The Sting"]
}
}
})
```

برای مثال شما می خواهید کاربرانی را پیدا کنید که علاقه مندی هایش در movies برابر "Casablanca" است

```
> db.users.find({"favorites.movies": "Casablanca"})
```

نقطه (.) بین favorites و movies به موتور جستجو این دستور را می دهد که کلید favorites را پیدا کن که شامل کلید درونی movies با ارزش مساوی Casablanca باشد

برای مثال های بیشتر شما فرض کنید که با توجه به دانش خود می دانیم که هرکسی که Casablanca را دوست دارد به طور حتم the maltese falcon را نیز دوست دارد. برای بروزرسانی دیتابیس برای نمایش این واقعیت راه اول اینست که از عملگر \$set استفاده کنیم ولی برای استفاده ازین عملگر نیاز به دوباره نویسی آرایه ی movies داشته باشیم. راه حل دوم اینست که از دو عملگر \$push و \$addToSet استفاده کنیم. درحقیقت این دو عملگر عنصر جدیدی به آرایه اضافه می کند با این تفاوت که عملگر دوم از وارد کردن عنصر تکراری جلوگیری می کند. با اجرای دستور زیر داریم:

```
db.users.update( {"favorites.movies": "Casablanca"},
{$addToSet: {"favorites.movies": "The Maltese Falcon"} },
false,
true )
```

آرگومان اول شرط برای پیدا کردن سند مورد نظر است. آرگومان دوم دستور اضافه کردن عنصر جدید و در مورد آرگومان سوم بعدا توضیح خواهیم داد و آرگومان چهارم این را مشخص می کند که این دستور یک multi-update می باشد.

حذف کردن داده

برای حذف کردن یک مجموعه بنام foo دستور زیر را وارد می نمایم

```
> db.foo.remove()
```

اگر نیاز داشته باشیم که زیرمجموعه ی خاصی از یک مجموعه را حذف نمایم با استفاده از دستور زیر اقدام به آن حذف آن سند یا زیرمجموعه می نمایم

```
> db.users.remove({"favorites.cities": "Cheyenne"})
```

باید توجه داشته باشیم که دستور remove مجموعه را حذف نمی کند بلکه تمام اسناد درون آن پاک می شوند. برای حذف کامل مجموعه با تمام index هایش از دستور drop استفاده می نمایم

```
> db.users.drop()
```

## ایجاد مجموعه های بزرگ

یک مثال برای indexing این است که بطریق آن می توان به یک مجموعه تعداد زیادی اسناد به یک باره اضافه کرد مثلا در shell می توان با استفاده از دستور

```
for(i=0; i<200000; i++) {  
db.numbers.save({num: i});  
}
```

به تعداد 200000 سند به مجموعه ی numbers اضافه کردیم (به دلیل اینکه shell از مفسر JavaScript پشتیبانی می کند پس دستور بالا قابل قبول است)

نتیجه...

```
> db.numbers.count()  
200000
```

```
> db.numbers.find()  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac830a"), "num" : 0 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac830b"), "num" : 1 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac830c"), "num" : 2 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac830d"), "num" : 3 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac830e"), "num" : 4 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac830f"), "num" : 5 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8310"), "num" : 6 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8311"), "num" : 7 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8312"), "num" : 8 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8313"), "num" : 9 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8314"), "num" : 10 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8315"), "num" : 11 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8316"), "num" : 12 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8317"), "num" : 13 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8318"), "num" : 14 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8319"), "num" : 15 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831a"), "num" : 16 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831b"), "num" : 17 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831c"), "num" : 18 }  
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831d"), "num" : 19 }
```

برای دیدن بقیه مجموعه کافیسست بنویسیم

```
> it
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831e"), "num" : 20 }
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831f"), "num" : 21 }
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8320"), "num" : 22 }
...
```

همچنین با استفاده از دستور \$gt و \$lt که به ترتیب بیانگر (بزرگتر از) یا (کوچکتر از) است می توان پرس و جویهای زیر را داشته باشیم

```
> db.numbers.find( {num: {"$gt": 199995 }} )
{ "_id" : ObjectId("4bfbf1dedba1aa7c30afcade"), "num" : 199996 }
{ "_id" : ObjectId("4bfbf1dedba1aa7c30afcadf"), "num" : 199997 }
{ "_id" : ObjectId("4bfbf1dedba1aa7c30afcae0"), "num" : 199998 }
{ "_id" : ObjectId("4bfbf1dedba1aa7c30afcae1"), "num" : 199999 }
```

```
> db.numbers.find( {num: {"$gt": 20, "$lt": 25 }} )
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac831f"), "num" : 21 }
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8320"), "num" : 22 }
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8321"), "num" : 23 }
{ "_id" : ObjectId("4bfbf132dba1aa7c30ac8322"), "num" : 24 }
```

Explain()

برای آشنایی با explain() ابتدا با یک مثال شروع می کنیم

```
> db.numbers.find( {num: {"$gt": 199995 }} ).explain()
```

اگر بالا را اجرا نماییم مشاهده می کنیم که نتایج زیر به ما برگردانده می شود

```
{
  "cursor" : "BasicCursor",
  "nscanned" : 200000,
  "nscannedObjects" : 200000,
```

**اجرای دستور بدون ایندکس گذاری**

```
"n" : 4,
"millis" : 171,
"nYields" : 0,
"nChunkSkips" : 0,
"isMultiKey" : false,
"indexOnly" : false,
"indexBounds" : { }
}
```

ما می توانیم با استفاده از دستور زیر بروی کلید num خود ایندکس بگذاریم

```
> db.numbers.ensureIndex({num: 1})
```

دستور بالا ایندکس گذاری صعودی را روی کلید num در مجموعه ی number را ایجاد می کند. با استفاده از دستور `getindexes()` مشخص می شود که ایندکس گذاری انجام گرفته شده است

```
> db.numbers.getIndexes()
[
  {
    "name" : "_id_",
    "ns" : "tutorial.numbers",
    "key" : {
      "_id" : 1
    }
  },
  {
    "_id" : ObjectId("4bfc646b2f95a56b5581efd3"),
    "ns" : "tutorial.numbers",
    "key" : {
      "num" : 1,
      "name" : "num_1"
    }
  }
]
```

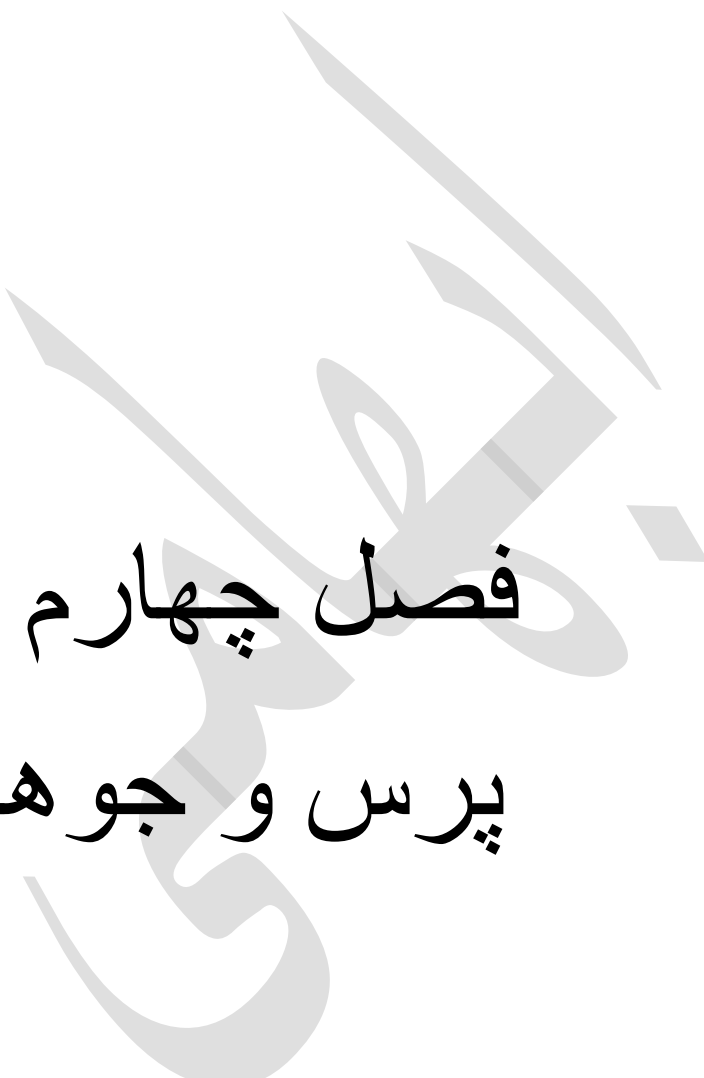
حال اگر دستور قبلی را اجرا نماییم مشاهده می کنیم که

```
> db.numbers.find({num: {"$gt": 199995 }}).explain()
{
  "cursor" : "BtreeCursor num_1",
  "indexBounds" : [
```

**اجرای دستور پس از  
ایندکس گذاری**

```
[  
  {  
    "num" : 199995},  
  {  
    "num" : 1.7976931348623157e+308}  
],  
"nscanned" : 5,  
"nscannedObjects" : 4,  
"n" : 4,  
"millis" : 0  
}
```

مشاهده می شود که با ایندکس گذاری زمان پرس و جو از 171 میلی ثانیه به کمتر از 1 میلی ثانیه کاهش پیدا کرده است.  
مدیریت پایگاه داده



# فصل چهارم

## پرس و جوها

همان طور که از قبل می دانیم برای نمایش هر چیزی که درون یک مجموعه است از دستور زیر استفاده می نمایم

```
> db.c.find()
```

دستور بالا تمام عناصر مجموعه ی C را برمی گرداند

حال اگر بخواهیم عناصر با `age=26` برگردانده شود کافی ست بنویسیم

```
> db.users.find({"age" : 27})
```

برای قرار دادن شروط چندتایی طبق زیر عمل می کنیم

Condition1 and condition2 and ...

```
> db.users.find({"username" : "joe", "age" : 27})
```

برای مشخص کردن کلیدهای برگشتی حاصل از جستجو کافی ست ارزش کلید مورد نظر را 1 کنیم

```
> db.users.find({}, {"username" : 1, "email" : 1})
{
  "_id" : ObjectId("4ba0f0dfd22aa494fd523620"),
  "username" : "joe",
  "email" : "joe@example.com"
}
```

توجه داشته باشید که همواره کلید `_id` در صورت انتخاب کردن یا انتخاب نکردن در نتیجه ما ظاهر می شود

اگر بخواهیم همه ی کلید ها بجز کلید موردنظر برگشت داده شوند ارزش کلید موردنظر را 0 می کنیم

```
> db.users.find({}, {"fatal_weakness" : 0})
```

دستور زیر می تواند از ظاهر شدن \_id در نتایج جستجو جلوگیری کند

```
> db.users.find({}, {"username" : 1, "_id" : 0})
{
  "username" : "joe",}
```

شرط ها

عبارات مقایسه ای شامل موارد زیر است

"\$lt", "\$lte", "\$gt", "\$gte", "<", "<=", ">", ">="

```
> db.users.find({"age" : {"$gte" : 18, "$lte" : 30}})
```

دستور بالا کاربرانی را برمی گرداند که دارای رنج سنی بین 18 تا 30 سال هستند همچنین دستور زیر که کاربرانی را که قبل از 01/01/2007 عضو شده اند را برمی گرداند

```
> start = new Date("01/01/2007")
> db.users.find({"registered" : {"$lt" : start}})
```

برای انتخاب اسنادی که ارزشش (نام کاربری) برابر ارزش ورودی (جو) نیست از دستوری مشابه زیر استفاده می کنیم

```
> db.users.find({"username" : {"$ne" : "joe"}})
```

"\$ne" با هر نوع داده ای بکار برده می شود

Or queries

دو را برای انجام پرس و جوها با or وجود دارد اولین را استفاده از "\$in" که بروی یک کلید انجام می گیرد مانند مثال زیر که برندگان مسابقه ی بخت آزمایی را در صوت داشتن یکی از بلیط های با شماره 3 و 5 و 8 را مشخص می کند

```
> db.raffle.find({"ticket_no" : {"$in" : [3, 5, 8]}})
```

همچنین می توان ازین عملگر بروی کلیدی با ارزش های متنوع استفاده نمود برای مثال:

```
> db.users.find({"user_id" : {"$in" : [12345, "joe"]}})
```

نکته) دو عبارت زیر برابرند

```
{ticket_no : 725}=={ticket_no : {$in : [725]}}
```

نقطه ی مقابل عملگر \$in عملگر \$nin می باشد

```
> db.raffle.find({"ticket_no" : {"$nin" : [3, 5, 8]}})
```

راه حل دوم در بکارگیری or در شرط ها استفاده از "\$or" است

```
> db.raffle.find({"$or" : [{"ticket_no" : 3}, {"winner" : true}]})
```

و یا

```
> db.raffle.find({"$or" : [{"ticket_no" : {"$in" : [3, 5, 8]}}, {"winner" : true}]}))
```

\$not

برای مثال فرض کنید که ما نیاز به کاربرانی داریم که دارای id\_num برابر اعداد 2و3و4و5و7و8و9... هستند در ابتدا باید کاربرانی را که دارای id\_num برابر اعداد 1و6و11و16... هستند را جدا کنیم و بقیه کاربران را نمایش دهیم از آنجایی که کاربران جدا شده دارای شناسه ی [5,1]mod هستند بدین سان عمل می نمایم

```
> db.users.find({"id_num" : {"$not" : {"$mod" : [5, 1]}}})
```

(نکات)

1- شروط چندتایی قابلیت اجرا روی یک کلید را دارند اما بروزرسانی چندتایی قابلیت اجرا بروی یک کلید را ندارد برای مثال دستور زیر نامعتبر است زیرا همزمان دوبار کلید age را تغییر می دهد که غیرقابل قبول است

```
{"$inc" : {"age" : 1}, "$set" : {age : 40}}
```

نوع های خاص در پرس و جو ها

تا این جا با نوع های متنوعی آشنا شده ایم که در اسناد بکار گرفته شده اند اما بعضی ازین نوع ها در پرس و جوها رفتار متمایزی دارند که با آنها آشنا می شویم

Null

به عنوان یک بیت ناآشنا عمل می کند برای مثال اگر یک مجموعه با اسناد زیر داشته باشیم  
آنگاه

```
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }  
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }  
{ "_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }
```

جواب پرس و جوی زیر

```
> db.c.find({"y" : null})
```

```
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
```

مقدار سندی را برمی گرداند که دارای y با ارزش null است

null علاوه بر اینکه مقادیر null را برمی گرداند مقادیری که وجود ندارند را نیز برمی گرداند اگر پرس و جوی زیر را روی مجموعه ی بالا انجام دهیم داریم:

```
> db.c.find({"z" : null})
```

```
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
```

```
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }
```

```
{ "_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }
```

حال اگر بخواهیم کلیدر را جستجو کنیم که دارای مقدار null است و همچنین وجود دارد از عملگر "\$exists" استفاده می کنیم

```
> db.c.find({"z" : {"$in" : [null], "$exists" : true}})
```

عبارات منظم

mongoDB از کتابخانه ی Perl Compatible Regular Expression (PCRE)

برای بررسی عبارات منظم استفاده می کند

بنابراین هر سینتکسی که در PCRE مجاز است در mongoDB نیز مجاز است  
ما می توانیم عبارات منظم خود را در پوسته ی جاوا اسکریپت ابتدا امتحان کنیم سپس در پرس و جوهای خود از آنها استفاده کنیم که این کار به ما اطمینان خاطر را درمورد درستی عباراتمان می دهد

برای مثال ما می خواهیم همه ی کاربرانی را که دارای اسم joe یا Joe هستند را پیدا کنیم  
در این حالت می توانیم از یک عبارت منظم استفاده کنیم که به کوچکی یا بزرگی حروف حساس نیست مانند مثال زیر

```
> db.users.find({"name" : /joe/i})
```

استفاده از فلگ i در عبارات منظم مجاز است ولی نیاز نیست حال اگر بخواهیم فقط نام کاربرانی را که مشابه joey هستند را پیدا کنیم کافی ست عبارت منظم را به صورت شکل زیر تصحیح کنیم

```
> db.users.find({"name" : /joey?/i})
```

```
Command Prompt - mongo
> use myblog
switched to db myblog
> db.users.find()
{ "_id" : ObjectId("50a12ce4931755a9d256b2d7"), "name" : "ashli" }
> db.users.find(<"name" : /ashl?/i>)
{ "_id" : ObjectId("50a12ce4931755a9d256b2d7"), "name" : "ashli" }
> db.users.find(<"name" : /ashlio?/i>)
{ "_id" : ObjectId("50a12ce4931755a9d256b2d7"), "name" : "ashli" }
> db.users.find(<"name" : /ashli?/i>)
{ "_id" : ObjectId("50a12ce4931755a9d256b2d7"), "name" : "ashli" }
>
```

همچنین می توانیم عبارت منظم را با خودش مقایسه کنیم بدین صورت که ابتدا آن را وارد پایگاه داده می کنیم

```
> db.foo.insert({"bar" : /baz/})
> db.foo.find({"bar" : /baz/})
{
  "_id" : ObjectId("4b23c3ca7525f35f94b60a2d"),
  "bar" : /baz/
}
```

آرایه ها

پرس و جو بر روی آرایه ای از عناصر بسیار آسان می باشد بدین صورت که عناصر درون آرایه بطوری رفتار می کنند که هر کدام ارزشی از یک کلید کلی هستند

```
> db.food.insert({"fruit" : ["apple", "banana", "peach"]})
```

در این جا کلید fruit شامل آرایه ای از میوه های apple,banana,peach هست

```
> db.food.find({"fruit" : "banana"})
```

نتیجه ی پرس و جوی بالا برگشت سند بالا است  
دقت داشته باشیم که سند بالا برابر با سند غیرقانونی زیر نیست

```
{"fruit" : "apple", "fruit" : "banana", "fruit" : "peach"}
```

## \$all

اگر بخواهیم آرایه هایی با عناصر بیش از یک عدد را با هم مقایسه کنیم از عملگر \$all استفاده می نماییم

فرض کنید مجموعه ی زیر را داریم

```
> db.food.insert({"_id" : 1, "fruit" : ["apple", "banana", "peach"]})
> db.food.insert({"_id" : 2, "fruit" : ["apple", "kumquat", "orange"]})
> db.food.insert({"_id" : 3, "fruit" : ["cherry", "banana", "apple"]})
```

ما به دنبال اسنادی هستیم که دارای دو عنصر apple و banana هستند

```
> db.food.find({fruit : {$all : ["apple", "banana"]}})
{"_id" : 1, "fruit" : ["apple", "banana", "peach"]}
{"_id" : 3, "fruit" : ["cherry", "banana", "apple"]}
```

همانطور که مشاهده می شود ترتیب مهم نیست در سند اول apple قبل از banana آمده است ولی در سند دوم بعد از banana آمده است  
توجه!!!

```
{fruit : 'apple'} == {fruit : {$all : ['apple']}}
```

دو سند بالا با هم برابرند و مقدار برابری را برمی گردانند

باید دقت داشته باشیم که در پری وجوهایمان تعداد عناصر آرایه را بدرستی وارد کنیم و از کم و زیاد وارد کردن پرهیز کنیم

```
> db.food.find({"fruit" : ["apple", "banana", "peach"]})
```



```
> db.food.find({"fruit" : ["apple", "banana"]})
```



```
> db.food.find({"fruit" : ["banana", "apple", "peach"]})
```



اگر قرار باشد برای یک عنصر خاصی از آرایه پرس و جو بنویسیم از ایندکس استفاده می نمایم برای مثال اگر بدنبال عنصر سوم از آرایه هستیم می نویسیم **key.index**

```
> db.food.find({"fruit.2" : "peach"})
```

### \$size

یک شرط کاربردی در استفاده از آرایه ها شرط بروی اندازه ی یک آرایه است

```
> db.food.find({"fruit" : {"$size" : 3}})
```

نمی توانیم \$size را با دیگر \$condition ها مانند "\$gt" ترکیب کنیم اما می توانیم با اضافه کردن یک کلید size به آرایه این کمبود را جبران کنیم هرگاه که ما یک عنصر به آرایه اضافه می کنیم به کلید size یک واحد اضافه می شود

```
> db.food.update({"$push" : {"fruit" : "strawberry"}, "$inc" : {"size" : 1}})
```

حال می توانیم پرس و جوی زیر را اجرا کنیم

```
> db.food.find({"size" : {"$gt" : 3}})
```

### \$slice operator

فرض کنید که در مجموعه ی blog.post می خواهیم 10 تا comment اول را برگردانیم برای این کار از عملگر \$slice استفاده می نمایم

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : 10}})
```

به همین ترتیب اگر بخواهیم 10 عنصر آخر را برگردانیم داریم:

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : -10}})
```

برای برگرداندن عناصر میانی نیاز به یک آفست (نقطه ی شروع) و تعداد عناصر برگشتی داریم

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : [23, 10]}})
```

این دستور 23 عنصر اول را نادیده می گیرد و از عنصر 24 تا 34 را برمی گرداند نکته ی مهم در مورد این عملگر این است که بقیه ی کلیدها نیز برگردانده می شوند برای مثال ما یک سند با نام posts که حاوی کلیدهای زیر است را ایجاد کرده ایم

```
> db.blog.posts.insert({"title":"my blog post","content":"...", "comments":[{"name":"masume","content":"nice blog"}, {"name":"john","content":"best"}]})
> db.blog.posts.find()
{ "_id" : ObjectId<"50a20a26ca709095664bf756">, "title" : "my blog post", "content" : "...", "comments" : [
  {
    "name" : "masume",
    "content" : "nice blog"
  },
  {
    "name" : "john",
    "content" : "best"
  }
] }
1 >
```

با اجرای پرس و جوی نتیجه ی زیر برگشت داده شده است

```
> db.blog.posts.findOne({}, {"comments":{"$slice":-1}})
{
  "_id" : ObjectId<"50a20a26ca709095664bf756">,
  "title" : "my blog post",
  "content" : "...",
  "comments" : [
    {
      "name" : "john",
      "content" : "best"
    }
  ]
}
1
>
```

همانطور که دیدیم بقیه ی کلیدها نیز در خروجی نمایش داده شدند .

## پرس و جو ها در Embedded documents

برای طرح پرس و جو از embedded document ها دو روش کلی وجود دارد

پرس و جو بروی کل سند یا پرس و جو بروی یک کلید منفرد

برای مثال پرس و جو بروی یک سند embedded مانند مثال زیر

```
{
  "name" : {
    "first" : "masume",
    "last" : "ebhami"
  },
  "age" : 22
}
```

می تواند بصورت زیر باشد

```
> db.people.find({"name" : {"first" : "masume", "last" : "ebhami"}})
```

حال اگر masume بخواهد کلید middle name را نیز به سند خود اضافه کند دیگر

دستور بالا جواب نخواهد داد همچنین این نوع پرس و جو به ترتیب کلیدها نیز حساسیت

نشان می دهد بنابراین ترتیب زیر برای برگرداندن سند بالا ترتیب درستی نیست

```
{"last" : "Schmoe", "first" : "Joe"}
```

راه حل؟

با استفاده از کلید و ارزش مورد نظر می توانیم مشکلات پرس و جو قبلی را حل کنیم

به مثال زیر دقت کنید

```
> db.people.find({"name.first" : "masume", "name.last" : "ebhami"})
```

حال مجموعه ی زیر را در نظر بگیرید

```
{
  "content" : "...",
  "comments" : [
    { "author" : "joe",
      "score" : 3,
      "comment" : "nice post" },
    {
      "author" : "mary",
      "score" : 6,
      "comment" : "terrible post"
    }
  ]
}
```

نمی توان پرس و جوی زیر را داشته باشیم زیرا همه ی کلید ها مقایسه نشده اند و کلید  
comments در پر سو جوی زیر match نشده است

```
db.blog.find({"comments" : {"author" : "joe", "score" : {"$gte" : 5}}})
```

همچنین نمی توانیم پرس و جوی زیر را داشته باشیم زیرا این پرس و جو سندی را که زودتر  
نمایش داده شده را برمی گرداند و مقدار  
Joe را از سند اول و مقدار 6 را از سند دوم بر می گرداند

```
db.blog.find({"comments.author" : "joe", "comments.score" : {"$gte" : 5}})
```

پرس و جوی درست در زیر نمایش داده شده است

```
> db.blog.find({"comments" : {"$elemMatch" : {"author" : "joe", "score" : {"$gte" : 5}}}})
```

استفاده از عملگر \$elemMatch این امکان را می دهد که محتویات درون سند خود را  
گروه بندی کنیم و برای زمانی که چندین کلید درون embedded document در حال  
مقایسه است استفاده می شود

## \$where

بیشترین کاربرد از این عملگر برای مقایسه ی دو کلید در یک سند است. برای مثال اگر ما  
یک لیست از آیتم هارا داشته باشیم و بخواهیم اسنادی که دارای مقدار مساوی هستند را پیدا  
کنیم طبق مثال زیر عمل می نمایم

```
> db.foo.insert({"apple" : 1, "banana" : 6, "peach" : 3})
```

```
> db.foo.insert({"apple" : 8, "spinach" : 4, "watermelon" : 4})
```

حال در سند بالا می خواهیم سندی را که در آن spinach و watermelon دارای  
مقدار مساوی هستند را برگردانیم. برای این کار از دستور \$where استفاده می کنیم که  
شامل متدهای java script است

```
> db.foo.find({"$where" : function () {  
... for (var current in this) {  
... for (var other in this) {  
... if (current != other && this[current] == this[other]) {  
... return true;  
... }  
... }  
... }  
... return false;  
})
```

... }));

اگر مقدار **true** برگردانده شود قسمتی از مجموعه جواب در سند وجود دارد و اگر مقدار **false** برگردانده شود سند دارای کلیدهای مساوی نبوده است همانطور که دیدیم ما از یک تابه استفاده کردیم که می توانیم بجای آن از یک رشته پرس وجو استفاده نماییم بدین ترتیب دو پرس وجوی زیر یکسان خواهند بود

```
> db.foo.find({"$where" : "this.x + this.y == 10"})
```

```
> db.foo.find({"$where" : "function() { return this.x + this.y == 10; }"})
```

فصل پنجم

Sharding

## Sharding

**Sharding** اشاره دارد به پردازش از طریق چندبخش کردن اطلاعات و ذخیره کردن هر بخش در یک ماشین جداگانه. در بعضی مواقع از پارتیشن بندی برای توصیف این مفهوم نیز استفاده می شود.

با بخش بخش کردن اطلاعات مدیریت و ذخیره سازی اطلاعات بزرگ بدون نیاز به ماشین های بزرگ و قدرتمند امکان پذیر است.

**Shard** کردن بصورت دستی تقریباً در بیشتر دیتابیس ها انجام می گیرد. زمانی که یک **application** حاوی چندین اتصال به سرورهای پایگاه داده ای جداگانه است که هر کدام به طور کامل مستقل از هم هستند در این حالت به مدیریت پیچیده برای ذخیره سازی داده های گوناگون بروی سرورهای متفاوت و همچنین طرح پرس و جوها بروی سرور مناسب برای برگرداندن جواب نیاز داریم البته این شیوه اجرایی است اما دردسر زیاد دارد.

**mongoDB** از مزیت **shard** کردن بصورت اتوماتیک بهره دار است که بسیاری از دردسرهای **shard** کردن را از بین برده است

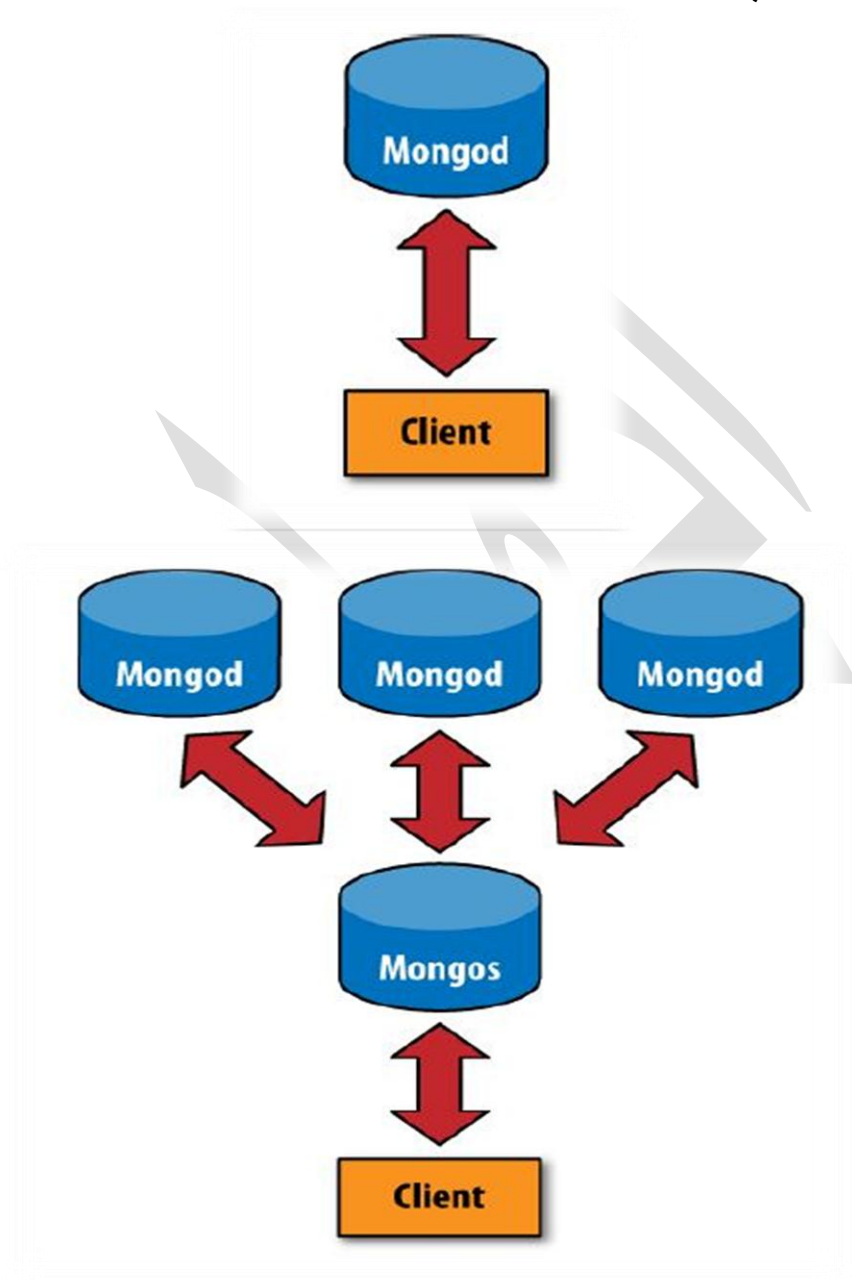
## AutoSharding

مفهوم **sharding** در **mongo** به این معنی است که مجموعه ها به تکه های کوچکتر بنام **chunks** تقسیم می شوند

این **chunk** ها در سراسر **shard** ها پخش می شوند که هر **shard** مسؤول یک زیر بخشی از کل دیتاها است بنابراین با استفاده از یک پردازش مسیریابی که **mongos** نام دارد عملیات **sharding** انجام می گیرد

مسیریاب مکان همه ی داده ها را می داند بنابراین زمانی یک **application** به آن وصل می شود می تواند درخواست هایش را مانند درخواست های معمولی به یک دیتابیس **mongoDB** بدهد

با توجه به دو شکل زیر در میابیم که در شکل اول از sharding استفاده نشده است و کاربر به طور مستقیم به دیتابیس وصل است اما در شکل دوم با استفاده از mongos عمل sharding انجام گرفته است که سیستم در کل طوری رفتار می کند که شبیه بدون sharding کاربر با پایگاه داده تعامل دارد



کی از shard کردن استفاده می کنیم؟

- 1- زمانی که فضای دیسک بروی ماشین فعلی کم بیاد
  - 2- زمانی که نیاز به نوشتن سریه داده ها داریم
  - 3- زمانی که می خواهیم یک قسمت بزرگی از داده هارا در حافظه نگهداری کنیم
- در حقیقت ما می توانیم ابتدا shading را نکنیم و بعدا در صورت نیاز از sharding استفاده کنیم

## Shard کردن

برای shard کردن ابتدا یک کلید از مجموعه انتخاب می کنیم که برحسب آن کلید داده ها را جدا می کنیم این کلید shad key نام دارد  
با یک مثال بیشتر به این موضوع می پردازیم:  
فرض کنید که یک مجموعه حاوی اسناد داریم که اطلاعات اشخاص را نگهداری می کند  
اگر shard key ما بر اساس نام اشخاص باشد در این صورت یک shard می تونه اسامی رو که با A-F شروع میشه رو نگهداری کنه یک shard اسامی رو که با G-P و shard آخری شامل اسامی که با Q-Z شروع میشه رو نگهداری بکنه  
فرض کنید که یک مجموعه داریم که log هارو نگهداری می کند و ما می خواهیم که آن را shard کنیم اگر ما عمل shard را اجرا کنیم و به MongoDB بگوییم که از timestamp به عنوان shard key استفاده کند در آن حال ما فقط یک shard روی همه ی داده هایمان داریم و با وارد کردن داده ها همه ی آنها به یک shard واحد می روند  
چنانچه یک shard دیگر اضافه کنیم در این حالت مجموعه به دو قسمت تقسیم خواهد شد که chunks نام دارد که یک chunk حاوی اسنادی است که در رنج ارزش shard key هستند مثلا در یک chunk حاوی اسناد در رنج timestamp بین  $-\infty$  تا june 26 2010 و یک chunk حاوی اسناد june 27 2010 تا  $+\infty$  است

Sharding چه تاثیری روی عملیات دارد؟

به عنوان یک end user نمی توانیم تفاوتی بین یک مجموعه ی shard شده و یک مجموعه ی shard نشده ببینیم

فرض کنید که همون مجموعه اشخاص را داریم که برحسب اسم shard شده است که سه تا shard در رنج A تا Z داریم که در اینصورت پرس و جوهای متفاوت به روش های متفاوت اجرا خواهند شد

```
db.people.find({"name" : "Susan"})
```

mongos این پرس و جو را به طور مستقیم به shard حاوی اسامی شروع شونده با Q-Z خواهد فرستاد و جواب را به کاربر بر می گرداند

```
db.people.find({"name" : {"$lt" : "L"}})
```

در اینصورت به shard های A-F و G-P مراجعه می شود

```
db.people.find().sort({"email" : 1})
```

mongos پرس و جو را به همه ی shard ها می فرستد و سپس نتایج را با هم مرتب سازی ادغامی می کند و به عنوان یک نتیجه ی واحد بر می گرداند

```
db.people.find({"email" : "joe@example.com"})
```

در اینصورت این پرس و جو به صورت پشت سر هم به همه ی chunk ها فرستاده می شود تا نتیجه برگردانده شود

### عملیات sharding

دو مرحله برای عملیات shard وجود دارد

**Sharding** در اساس شامل سه جزء است در ارتباط با هم عمل می کنند

**Shard**

یک محفظه ای که زیرمجموعه هایی از داده های یک مجموعه را نگهداری می کند همچنین یک سرور mongod یا یک مجموعه ی المثنی است اگر هم در یک shard چندین سرور وجود داشته باشد باز یک سرور اصلی است و سرورهای دیگر نیز شامل همان داده ها هستند

### Mongos

یک پردازش مسیریابی است که در همه ی توزیع های mongo وجود دارد که در اصل یک درخواست مسیریابی و جواب های جمعی است

### Config server

پیکربندی کلاسترها را ذخیره می کند. کدام داده در کدام shrd است. چون mongos هیچ چیزی را بطور همیشگی ذخیره نمی کند به مکانی نیاز داریم که پیکربندی shard ها را نگهداری کند

در ابتدا باید پیکربندی سرور و mongos را انجام بدیم در ابتدا سرور را Config میکنیم زیرا برای mongos به آن نیاز داریم

برای Config سرور قبل از همه ی عملیات از قبیل mongod شروع می کنیم

```
$ mkdir -p ~/dbs/config
```

```
$ ./mongod --dbpath ~/dbs/config --port 20000
```

که ان پیکربندی به فضا یا منابع زیادی نیاز ندارد (فضایی حدود 1 کیلو بایت در هر 200 مگا بایت از اطلاعات واقعی)

حال به mongos نیاز داریم که عملیات مسیریابی در اتصالات را انجام دهد که این عملیات مسیریابی نیاز دراد که بداند Config server کجاست

```
$ ./mongos --port 30000 --configdb localhost:20000
```

اضافه کردن یک shard  
در حقیقت shard یک نمونه از mongod است

```
$ mkdir -p ~/dbs/shard1
```

```
$ ./mongod --dbpath ~/dbs/shard1 --port 10000
```

حال ما به mongos متصل شده ایم و می توانیم shard خود را به کلاسترمان اضافه کنیم  
برای اطمینان از اینکه به mongos متصل شده ایم:

```
$ ./mongo localhost:30000/admin
```

```
MongoDB shell version: 1.6.0
```

```
url: localhost:30000/admin
```

```
connecting to localhost:30000/admin
```

```
type "help" for help
```

برای اضافه کردن shard داریم :

```
> db.runCommand({addshard : "localhost:10000", allowLocal : true})
```

```
{
```

```
"added" : "localhost:10000",
```

```
"ok" : true
```

```
}
```

کلید allowLocal فقط این ضرورت را دارد که بیان می کند shard ما روی localhost است

### Sharding Data

فرض کنید که می خواهیم مجموعه ی bar را در دیتابیس foo بر حسب کلید \_id shard کنیم  
در ابتدا sharding را برای دیتابیس foo فعال می کنیم

```
> db.runCommand({"enablesharding" : "foo"})
```

Shard کردن دیتابیس مجموعه ها را در shard های مختلف ذخیره می کند که برای  
shard کردن یک مجموعه این عمل پیشنهاد است.

به یکباره می توانیم از ابتدا یک مجموعه را بوسیله ی دستور shardcollection  
مجموعه را shard کنیم

```
> db.runCommand({"shardcollection" : "foo.bar", "key" : {"_id" : 1}})
```

حال ما مجموعه را بر حسب \_id shard کرده ایم. حال اگر داده اضافه کنیم بصورت  
خودکار داده ها با توجه به \_id در shard های مربوطه قرار می گیرد

# فصل ششم

# Aggregation

mongoDB ابزارهای زیادی را برای aggregation فراهم ساخته است از شمارش ساده اسناد درون یک مجموعه گرفته تا آنالیز داده های پیچیده با استفاده از mapreduce

## Count

ساده ترین ابزار aggregation که تعداد اسناد درون یک مجموعه را برمی گرداند

```
> db.foo.count()
```

```
0
```

```
> db.foo.insert({"x" : 1})
```

```
> db.foo.count()
```

```
1
```

علی رغم سبب یک مجموعه عملیات شمارش با سرعت زیادی انجام می گیرد همچنین می توانیم یک پرس و جو به count بدهیم تا نتیجه را برگرداند

```
> db.foo.insert({"x" : 2})
```

```
> db.foo.count()
```

```
1
```

```
> db.foo.count({"x" : 1})
```

```
1
```

## Distinct

این دستور همه ی مقدار های متفاوت برای کلید داده شده را برمی گرداند که باید یک مجموعه و یک کلید به عنوان ورودی به این دستور داده شود مانند مثال زیر:

```
> db.runCommand({"distinct" : "people", "key" : "age"})
```

فرض کنید مجموعه ی زیر را با اسناد داده شده داریم

```
{"name" : "Ada", "age" : 23}
```

```
{"name" : "ashli", "age" : 30}
```

```
{"name" : "john", "age" : 25}
```

```
{"name" : "Andy", "age" : 30}
```

حال نتیجه ی دستور بالا بروی این مجموعه به قرار زیر است

```
> db.runCommand({"distinct" : "people", "key" : "age"})
```

```
{"values" : [20, 35, 60], "ok" : 1}
```

## Group

اگر با زبان های sql آشنایی داشته باشید باید بگوییم که group در mongo شبیه group by در sql است.

Group این اجازه را می دهد که مجموعه ی خود را به زیرگروه های جداگانه بر حسب مقدار کلید انتخابی تقسیم بندی کنیم

برای مثال فرض کنید که ما سائیتی داریم که موجودی را هر چند دقیقه یکبار از ساعت 10 صبح تا 4 عصر ثبت می کند و همواره آخرین موجودی را نگهداری می کند  
حال اگر بخواهیم مقدار آخرین موجودی در 30 روز قبل را مشاهده کنیم با گروه بندی کار ما آسان تر می شود

```
{"day": "2010/10/03", "time": "10/3/2010 03:57:01 GMT-400", "price": 4.23}
```

```
{"day": "2010/10/04", "time": "10/4/2010 11:28:39 GMT-400", "price": 4.27}
```

```
{"day": "2010/10/03", "time": "10/3/2010 05:00:23 GMT-400", "price": 4.10}
```

```
{"day": "2010/10/06", "time": "10/6/2010 05:27:58 GMT-400", "price": 4.30}
```

```
{"day": "2010/10/04", "time": "10/4/2010 08:34:50 GMT-400", "price": 4.01}
```

ما می خواهیم لیست آخرین موجودی هر روز را مشاهده کنیم مانند زیر

```
[{"time": "10/3/2010 05:00:23 GMT-400", "price": 4.10}, {"time": "10/4/2010 11:28:39 GMT-400", "price": 4.27}, {"time": "10/6/2010 05:27:58 GMT-400", "price": 4.30}]
```

برای این کار کافی ست که مجموعه ی خود را برحسب روز گروه بندی کنیم و در هر گروه آخرین سند را پیدا کنیم

```
> db.runCommand({"group" : {
... "ns" : "stocks",
... "key" : "day",
... "initial" : {"time" : 0},
... "$reduce" : function(doc, prev) {
... if (doc.time > prev.time) {
... prev.price = doc.price;
... prev.time = doc.time;
... }
... }}}})
```

## حال به بررسی بخش های این دستور می پردازیم

"ns" : "stocks",  
 "key": "day"  
 کند. قطعا در این مثال اسنادی که دارای روز یکسان هستند در یک گروه قرار خواهند گرفت  
 "initial" : { "time" : 0 }

"\$reduce" : function(doc, prev) { ... }

این تابع برای هر سند درون مجموعه یکبار فراخوانی می شود که سند جاری را می گیرد و یک انباشه برای اسناد است که در این مثال ما تاریخ سند جاری را با تاریخ انباشه مقایسه می کنیم و در صورت آخرین سند بودن زمان و موجودی را در انباشه ذخیره می کنیم. توجه داشته باشید که برای هر گروه یک انباشه جداگانه خواهیم داشت در ابتدا گفتیم که می خواهیم نتایج مربوط به 30 روز گذشته باشد پس نیاز به شرطی داریم که این مشکل را حل کند

```
> db.runCommand({"group" : {
... "ns" : "stocks",
... "key" : "day",
... "initial" : {"time" : 0},
... "$reduce" : function(doc, prev) {
... if (doc.time > prev.time) {
... prev.price = doc.price;
... prev.time = doc.time;
... }},
... "condition" : {"day" : {"$gt" : "2010/09/30"}}
... })
```

دستور بالا آرایه ای از 30 روز را می آورد که هر کدام یک گروه هستند هر گروه شامل کلیدی است که برحسب آن گروه بندی شده است مثلا در این مثال ("day" : string) و مقدار prev نهایی. اگر بعضی اسناد شامل کلید day نباشند در یک گروه با عنصر day:null گروه بندی می شوند که برای حذف این گروه ها کافی ست که {"\$exists" : true} را در قسمت شرط بنویسیم که دستور group تعداد اسناد استفاده شده و تعداد مقدارهای متفاوت برای کلید را بر می گرداند

```
> db.runCommand({"group" : {...}})
{
"retval" :
```

چنانچه بخواهیم نتیجه بازگشتی بصورتی که خودمان می خواهیم باشد از Finalizer استفاده می نماییم

**Finalizer**

Finalizer برای کم کردن حجم اطلاعات خروجی از سمت دیتابیس به سمت کاربر استفاده می شوند که این یک ویژگی خیلی مهمی است زیرا برای خروجی گروه بندی نیاز به یک قالبی مناسبی از جواب دیتابیس داریم

برای نشان دادن این امر با یک مثال شروع می کنیم

در این مثال ما وبلاگی داریم که دارای پست هایی با تگ های متفاوت است حال می خواهیم محبوب ترین تگ در هر روز را مشاهده نماییم در اینصورت نیاز داریم که گروه بندی بر حسب روز داشته باشیم که مقدار تگ ها را نگهداری کند

که نتیجه ای شبیه زیر را بر می گرداند

**www.ParsBook.org**

```
{ "day" : "2010/01/13", "tags" : { "soda" : 5, "php" : 2 } },
{ "day" : "2010/01/14", "tags" : { "python" : 6, "winter" : 4, "nosql" : 15 } }
]
```

سپس ما مقدار بزرگترین تگ را در سمت کاربر پیدا می کنیم اما فرستادن همه ی تگ ها به سمت کاربر دارای سربار زیادی است چاره چیست؟  
یک finalizer برای تابعی است که روی هر گروه قبل از فرستاده شدن به سمت کاربر اجرا می شود که در این حالت خروجی ما مرتب و بدون هیچ اضافاتی است

```
> db.runCommand({"group" : {
... "ns" : "posts",
... "key" : {"tags" : true},
... "initial" : {"tags" : {}},
... "$reduce" : function(doc, prev) {
... for (i in doc.tags) {
... if (doc.tags[i] in prev.tags) {
... prev.tags[doc.tags[i]]++;
... } else {
... prev.tags[doc.tags[i]] = 1;
... }
... },
... "finalize" : function(prev) {
... var mostPopular = 0;
... for (i in prev.tags) {
... if (prev.tags[i] > mostPopular) {
... prev.tag = i;
... mostPopular = prev.tags[i];
... }
... }
... delete prev.tags
... }}}}

```

حال فقط نتایجی برگشت داده می شود که مورد نیاز ما است

```
[
{ "day" : "2010/01/12", "tag" : "winter"},
{ "day" : "2010/01/13", "tag" : "soda"},
{ "day" : "2010/01/14", "tag" : "nosql"}
]
```

استفاده از تابع به عنوان کلید

در گروه بندی می توانیم بجای اینکه از یک کلید به عنوان معیاری برای گروه بندی استفاده کنیم از تابع های دلخواهی برای گروه بندی استفاده کنیم در این حالت فقط کافی ست بجای key کلمه ی keyf را بنویسیم مانند دستور زیر

```
> db.posts.group({"ns" : "posts",  
... "$keyf" : function(x) { return x.category.toLowerCase(); },  
... "initializer" : ... })
```

## MapReduce

یکی از ابزارهای aggregation است که بیشتر از هر کاری را که می توانیم با count group distinct انجام بدهیم با mapreduce نیز انجام بدهیم همچنین با این روش aggregation می شود عملیات موازی سازی بروی چندین ماشین نیز انجام بگیرد که در این صورت مساله به چند قسمت تقسیم می شود و هر قسمت در یک ماشین جداگانه مساله حل می شود و پس از اتمام کار در هر ماشین جواب ها با هم ادغام می شوند و به عنوان یک جواب نهایی بازگست داده می شود

# فصل هفتم Indexing

اینکه تعیین کنیم چگونه یک ایندکس گذاری بهینه روی پرس و جویمان داشته باشیم نیاز به مهارت بالا دارد اما باعث افزایش کارایی می شود  
بر فرض ما یک پرس و جوی ساده بروی یک کلید داریم

```
> db.people.find({"username" : "mark"})
```

زمانی که یک کلید در پرس و جو استفاده می شود با ایندکس گذاری سرعت پرس و جو را بالا می بریم. در این مثال می خواهیم یک ایندکس بروی username ایجاد کنیم برای این کار از متد ensureIndex استفاده می نماییم

```
> db.people.ensureIndex({"username" : 1})
```

یک ایندکس فقط برای یکبار روی هر مجموعه ایجاد می شود. اگر بخواهیم همان ایندکس را برای بار دیگر ایجاد کنیم هیچ اتفاقی نمی افتد

ایندکس روی یک کلید باعث می شود که پرس و جوها بروی آن کلید سریعتر اجرا شود اما پرس و جویهای دیگر را سریع نمی کند حتی اگر شامل کلید ایندکس شده باشد  
برای مثال با ایندکس گذاری جاری بروی پرس و جوی زیر تاثیری رخ نمی دهد

```
> db.people.find({"date" : date1}).sort({"date" : 1, "username" : 1})
```

یک قانون کلی این است که همه ی کلید های استفاده شده در پرس و جو باید ایندکس گذاری شوند برای مثال باید در پرس و جوی بالا هر دو کلید date و username ایندکس گذاری شوند

```
> db.ensureIndex({"date" : 1, "username" : 1})
```

اگر بیشتر از یک کلید داشته باشیم در آن صورت باید انتخاب درستی برای ایندکس گذاری کنیم  
برای مثال مجموعه ی زیر را داریم

```
{ "_id" : ..., "username" : "smith", "age" : 48, "user_id" : 0 }  
{ "_id" : ..., "username" : "smith", "age" : 30, "user_id" : 1 }  
{ "_id" : ..., "username" : "john", "age" : 36, "user_id" : 2 }  
{ "_id" : ..., "username" : "john", "age" : 18, "user_id" : 3 }  
{ "_id" : ..., "username" : "joe", "age" : 36, "user_id" : 4 }  
{ "_id" : ..., "username" : "john", "age" : 7, "user_id" : 5 }  
{ "_id" : ..., "username" : "simon", "age" : 3, "user_id" : 6 }  
{ "_id" : ..., "username" : "joe", "age" : 27, "user_id" : 7 }  
{ "_id" : ..., "username" : "jacob", "age" : 17, "user_id" : 8 }  
{ "_id" : ..., "username" : "sally", "age" : 52, "user_id" : 9 }  
{ "_id" : ..., "username" : "simon", "age" : 59, "user_id" : 10 }
```

حال می خواهیم بر حسب { "username" : 1, "age" : -1 } ایندکس گذاری کنیم که نتایج زیر برگشت داده می شود

```
{ "_id" : ..., "username" : "jacob", "age" : 17, "user_id" : 8 }  
{ "_id" : ..., "username" : "joe", "age" : 36, "user_id" : 4 }  
66 | Chapter 5: Indexing  
{ "_id" : ..., "username" : "joe", "age" : 27, "user_id" : 7 }  
{ "_id" : ..., "username" : "john", "age" : 36, "user_id" : 2 }  
{ "_id" : ..., "username" : "john", "age" : 18, "user_id" : 3 }  
{ "_id" : ..., "username" : "john", "age" : 7, "user_id" : 5 }  
{ "_id" : ..., "username" : "sally", "age" : 52, "user_id" : 9 }  
{ "_id" : ..., "username" : "simon", "age" : 59, "user_id" : 10 }  
{ "_id" : ..., "username" : "simon", "age" : 3, "user_id" : 6 }  
{ "_id" : ..., "username" : "smith", "age" : 48, "user_id" : 0 }  
{ "_id" : ..., "username" : "smith", "age" : 30, "user_id" : 1 }
```

همانطور که دیده می شود نام کاربری برحسب افزایش حروف الفبا به ترتیب قرار می گیرد و در اسم های برابر سن برحسب کاهش سن مرتب شده اند

ایندکس گذاری بروی **age** و **username** همچنین باعث تسریع پرس و جوها روی **username** دارد به طور کلی اگر ایندکس ما روی **n** کلید باشد در اینصورت پرس و جوها بروی هر یک از پیشوندهای این کلیدها سریع است مثلاً اگر ایندکسی با کلیدهای { "a" : 1, "b" : 1, "c" : 1, ..., "z" : 1 } را داشته باشیم هر کلیدی مانند کلید های زیر ایندکس می شوند { "a" : 1, "b" : 1, "c" : 1 }, { "a" : 1, "b" : 1 }, { "a" : 1 } ولی بروی { "b" : 1 }, { "a" : 1, "c" : 1 }, کاربردی ندارد بنابراین کلید باید پیشوندهایی از ایندکس ما باشد

# فصل هشتم

## مدیریت در MongoDB

## شروع کار با MongoDB

برای روشن کردن سرور MongoDB از دستور `mongod` استفاده می نماییم. همچنین دارای گزینه های مدیریتی زیادی است که می توان آنها را با استفاده از دستور `mongod -help` مشاهده کنیم. موارد مهم در زیر آمده است

### --dbpath

یک مسیر فرعی برای دایرکتوری داده ها اختصاص می دهد. مسیر پیش فرض برای داده ها `data/db` است هر `mongod` نیاز به دایرکتوری داده ی خودش دارد حال اگر ما سه پردازش `mongod` داشته باشیم نیاز به سه دایرکتوری جداگانه داریم. زمانی که `mongod` اجرا می شود یک فایل بنام `mongod.lock` در دایرکتوری داده ایجاد می شود که از پردازش `mongod` های دیگر در داخل دایرکتوری جاری جلوگیری می کند. حال اگر سعی داشته باشیم `mongod` دیگر را در همان دایرکتوری داده اجرا کنیم با خطای زیر مواجه می شویم

"Unable to acquire lock for lockfilepath: /data/db/mongod.lock."

### --port

یک شماره پورت به سرور `mongod` ما اختصاص می دهد به طور پیش فرض پورت 27017 برای `mongod` می باشد که استفاده از آن توسط پردازش های دیگر بعید است. اگر بخواهیم چندین `mongod` را اجرا کنیم نیاز به پورت های متفاوت داریم اگر بخواهیم `mongod` را روی یک پورت که در حال استفاده است اجرا کنیم با خطای زیر مواجه می شویم

"Address already in use for socket: 0.0.0.0:27017"

### --fork

Fork کردن سرور اجرای MongoDB را به صورت حیرت انگیزی قدرتمند می نماید

### --logpath

همه ی خروجی ها را به یک فایل مخصوصی که سریع تر از خروجی خط دستور است می فرستد. که در صورت نبود فایلی را ایجاد می کند که `log` ها را نگهداری می کند و `log` های قدیمی تر را پاک می نماید. در صورتی که بخواهیم `log` های قدیمی تر نیز حفظ شوند از `logappend --` در کنار `logpath` استفاده می کنیم

### --config

از یک فایل تنظیمات که شامل گزینه های اضافی است که در خط فرمان نیامده است استفاده می کند

پیکربندی بر پایه ی فایل

mongoDB از پیکربندی اطلاعات از روی یک فایل پشتیبانی می کند  
برای گرفتن گزینه های پیکربندی از یک فایل از دستور -f یا -config استفاده می کنیم  
گزینه هایی که در یک فایل پیکربندی پشتیبانی می شوند دقیقاً گزینه های مورد پذیرش در خط فرمان است  
این یک مثال از یک فایل پشتیبانی است

```
# Start MongoDB as a daemon on port 5586
```

```
port = 5586
```

```
fork = true # daemonize it!
```

```
logpath = mongod.log
```

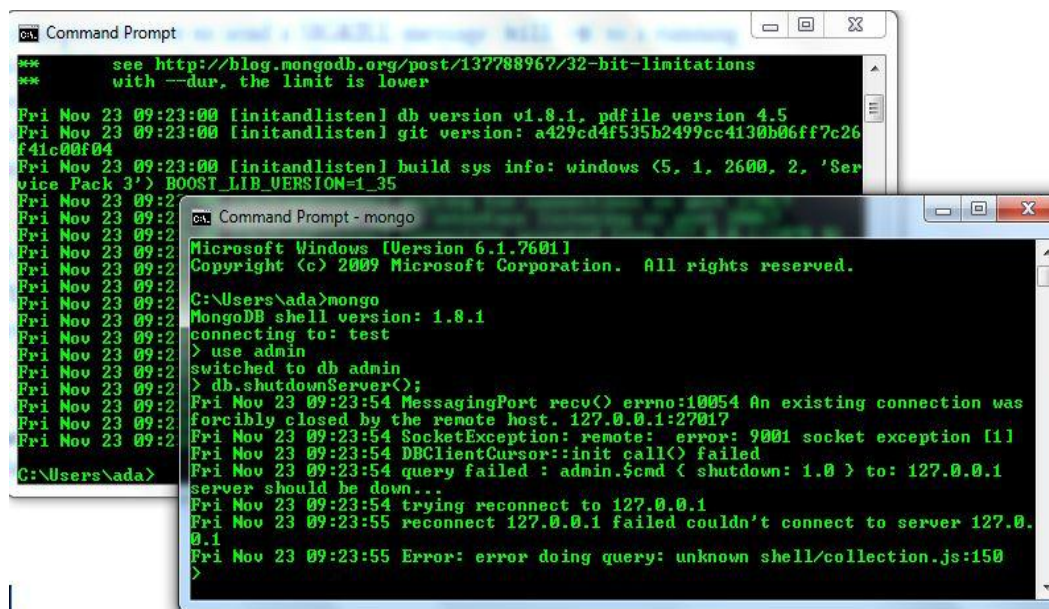
که برابر با همان تنظیماتی است که در خط فرمان انجام می دهیم  
همین جنبه های جالبی در مورد این فایل ها وجود دارد :

- 1- هر متنی که با # شروع شود یک توضیح است نه یک دستور
- 2- سینتکس برای مشخص کردن گزینه ها شبیه option=value است
- 3- برای گزینه های شبیه fork- برابر این است که مقدار را true قرار دهیم

متوقف کردن mongoDB

گزینه های زیادی برای متوقف کردن mongoDB وجود دارد که عام ترین راه حل است که زمانی که سرور در ترمینال در حال اجراست هم زمان کلید ترکیبی ctrl,c را بفشاریم تا سرور توقف کند. راه حل دیگر استفاده از دستور kill است  
دستور دیگر برای خاموش کردن سرور این است که به عنوان ادمین وارد دیتابیس admin بشویم و دستور زیر را اجرا نماییم

```
{“shutdownServer()”:1}
```



```
see http://blog.mongodb.org/post/137788967/32-bit-limitations
with --dur, the limit is lower

Fri Nov 23 09:23:00 [initandlisten] db version v1.8.1, pfile version 4.5
Fri Nov 23 09:23:00 [initandlisten] git version: a429cd4f535b2499cc4130b06ff7c26
f41c00f04
Fri Nov 23 09:23:00 [initandlisten] build sys info: windows (5, 1, 2600, 2, 'Ser
vice Pack 3') BOOST_LIB_VERSION=1_35

C:\Users\ada>mongo
MongoDB shell version: 1.8.1
connecting to: test
> use admin
switched to db admin
> db.shutdownServer();
Fri Nov 23 09:23:54 MessagingPort recv() errno:10054 An existing connection was
forcibly closed by the remote host. 127.0.0.1:27017
Fri Nov 23 09:23:54 SocketException: remote: error: 9001 socket exception [1]
Fri Nov 23 09:23:54 DBClientCursor::init call() failed
Fri Nov 23 09:23:54 query failed : admin.$cmd { shutdown: 1.0 } to: 127.0.0.1
server should be down...
Fri Nov 23 09:23:54 trying reconnect to 127.0.0.1
Fri Nov 23 09:23:55 reconnect 127.0.0.1 failed couldn't connect to server 127.0.
0.1
Fri Nov 23 09:23:55 Error: error doing query: unknown shell/collection.js:150
>
```

## نظارت یا monitoring

نظارت بروی کارایی و سالم بودن سرور برای هر مدیر سیستمی اهمیت دارد .  
خوشبختانه mongoDB دارای عامل هایی است که این کار را آسان کرده است  
استفاده از صفحه ی مدیریت

این واسطه به طور پیش فرض بروی سرور http روی پورت 1000 به بالای پورت  
محلی درایور mongo ما قرار دارد که این صفحه دارای اطلاعات اولیه از سرور ما  
است

برای دیدن این صفحه کافی ست سرور را روشن کنیم و در مرورگر خود بنویسیم  
<http://localhost:28017>

چنانچه از گزینه ی port- استفاده کنیم در شماره پورت عدد پورت درایور محلی  
1000+ را می نویسیم

همانطور که در شکل زیر مشاهده می شود امکاناتی از قبیل تایید و جستجو و ایندکس  
گذاری و replication و اطلاعاتی در مورد سرور وجود دارد  
برای اینکه بتوانیم استفاده کاملی از این صفحه داشته باشیم نیاز به روشن کردن پشتیبان  
REST است که با دستور rest- این کار انجام می گیرد  
چنانچه بخواهیم صفحه ی ادمین را خاموش باشد هم زمان که mongod را روشن می  
کنیم می نویسیم nohttpinterface-

نکته)

هیچ گاه تلاش نکنید که از طریق http به درایور های mongoDB و پورتهای  
درایور ها متصل بشوید. زیرا پورتهای درایور تنها با پروتکل سیمی محلی mongoDB  
کنترل می شود نه با درخواست های http

اگر دستور <http://localhost:27017> را در مرورگر خود بنویسیم با نوشته ی زیر مواجه می شویم

You are trying to access MongoDB on the native driver port.  
For http diagnostic access, add 1000 to the port number





وضعیت های سرور  
ابزار های آمارگیری زیادی برای وضعیت گیری در مورد سرور وجود دارد با نوشتن دستور `serverStatus` در خط فرمان نتایج زیر را نمایش می دهد  
(با توجه به platform و نسخه ی mongoDB ممکن است کلید ها با همدیگر متفاوت باشند)

```
Command Prompt - mongo
connecting to: test
> db.runCommand(<{"serverStatus": 1}>)
{
  "host" : "ada-PC",
  "version" : "1.8.1",
  "process" : "mongod",
  "uptime" : 805,
  "uptimeEstimate" : 797,
  "localTime" : ISODate("2012-11-23T17:49:15.067Z"),
  "globalLock" : {
    "totalTime" : 804448012,
    "lockTime" : 5000,
    "ratio" : 0.000006215442048976062,
    "currentQueue" : {
      "total" : 0,
      "readers" : 0,
      "writers" : 0
    },
    "activeClients" : {
      "total" : 0,
      "readers" : 0,
      "writers" : 0
    }
  },
  "mem" : {
    "bits" : 32,
    "resident" : 5,
    "virtual" : 69,
    "supported" : true,
    "mapped" : 0
  },
  "connections" : {
    "current" : 1,
    "available" : 19999
  },
  "extra_info" : {
    "note" : "fields vary by platform"
  },
  "indexCounters" : {
    "note" : "not supported on this platform"
  },
  "backgroundFlushing" : {
    "flushes" : 13,
    "total_ms" : 1,
    "average_ms" : 0.07692307692307693,
    "last_ms" : 0,
    "last_finished" : ISODate("2012-11-23T17:48:50.720Z")
  },
  "cursors" : {
    "totalOpen" : 0,
    "clientCursors_size" : 0,
    "timedOut" : 0
  },
  "network" : {
    "bytesIn" : 140,
    "bytesOut" : 169,
    "numRequests" : 2
  }
}
```

همچنین می توانیم اطلاعات مربوط به وضعیت را در یک فایل json مشاهده کنیم که برای مشاهده این فایل در مرورگر خود می نویسیم

([http://localhost:28017/\\_status](http://localhost:28017/_status))

به بررسی چند کلید مهم می پردازیم

"globalLock"

"mem"

"indexCounters"

"misses"

"hits"

"background Flushing"

"opcounters"

"asserts"

Mongostat

اگرچه **serverStatus** خیلی قدرتمند است اما دارای مکانیسم کاربرپسندی برای کارهای نظارتی ندارد

خوشبختانه ی توزیع های **mongoDB** یک **mongostat** با **mongoDB** همراه کرده اند که یک صفحه ی کاربر پسند برای خروجی وضعیت سرور دارد

**Mongostat** اطلاعات مهمی از وضعیت سرور را در خروجی چاپ می کند که در ثانیه وضعیت سرور را در یک خط جدید به ما می دهد که باعث می شود ما یک دید **real-time** از وضعیت سرور داشته باشیم

هر ستونی که در **mongostat** چاپ می شود دارای عناوینی مشابه **inserts/s**, **commands/s**, **vsize**, **% locked** است که دقیقاً برابر با داده های موجود در **serverStatus** است

Third-Party Plug-Ins

## Security

اولویت اول برای هر مدیر سیستم این است که از امنیت سیستم خود اطمینان کافی را داشته باشد. بهترین راه حل برای کنترل امنیت با MongoDB این است که MongoDB در محیطی امن و بروی ماشین های امنی که توانایی اتصال با سرور را دارند اجرا بشود

### اصول authentication

هر دیتابیس در MongoDB این امکان را دارد که شامل کاربران متعددی باشد زمانی که امنیت سیستم را فعال می نماییم تنها کاربران authenticate شده اجازه ی اعمال خواندن و نوشتن بروی دیتابیس ها را دارند لازم به ذکر است که MongoDB در بحث authentication از یک پایگاه داده ی اختصاصی بنام admin استفاده می کند که کاربر ادمین به عنوان یک سوپر کاربر در نظر گرفته می شود بعد از authentication کاربر ادمین می تواند هر کاری بروی دیتابیس های موجود انجام دهد و همچنین قادر به انجام دستوراتی شبیه shutdown و show dbs است قبل از شروع بحث و فعال کردن امنیت ابتدا لازم است که یک کاربر ادمین به سیستم خود اضافه کنیم

```
> use admin
switched to db admin
> db.addUser("root", "abcd");
{
  "user" : "root",
  "readOnly" : false,
  "pwd" : "1a0f1c3c3aa1d592f490a2addc559383"
}
> use test
switched to db test
> db.addUser("test_user", "efgh");
{
  "user" : "test_user",
  "readOnly" : false,
  "pwd" : "6076b96fc3fe6002c810268702646eec"
}
> db.addUser("read_only", "ijkl", true);
{
  "user" : "read_only",
  "readOnly" : true,
  "pwd" : "f497e180c9dc0655292fee5893c162f1"
}
```

همانگور که مشاهده می کنید یک کاربر ادمین و دو کاربر در دیتابیس test داریم

که یک کاربر `read_only` است و فقط اجازه ی خواندن از دیتابیس را دارد که این کار با `true` کردن مقدار آرگومان سوم متد انجام پذیرفته است و کاربر دیگر هم اجازه ی نوشتن و هم اجازه ی خواندن را دارد

همچنین می توان با فراخوانی متد `addUser` سطح دسترسی کاربر را هم تعیین کرد که برای این کار ابتدا باید امنیت فعال باشد  
(نکته) تابه `addUser` تنها برای اضافه کردن کاربر جدید نیست بلکه می توان با آن رمز و وضعیت `read_only` بودن را نیز تغییر داد. این کار بدین صورت است که کافی ست متد را با نام کاربری و رمز جدید یا وضعیت جدید فراخوانی کنیم  
فعال کردن امنیت

ابتدا سرور را `restart` می کنیم برای فعال کردن امنیت دستور `auth` - را در خط فرمان می نویسیم

```
> use test
switched to db test
> db.test.find();
error: { "$err" : "unauthorized for db [test] lock type: -1 " }
> db.auth("read_only", "ijkl");
1
> db.test.find();
{ "_id" : ObjectId("4bb007f53e8424663ea6848a"), "x" : 1 }
> db.test.insert({"x" : 2});
unauthorized
> db.auth("test_user", "efgh");
1
> db.test.insert({"x": 2});
> db.test.find();
{ "_id" : ObjectId("4bb007f53e8424663ea6848a"), "x" : 1 }
{ "_id" : ObjectId("4bb0088cbe17157d7b9cac07"), "x" : 2 }
> show dbs
assert: assert failed : listDatabases failed:{
"assertion" : "unauthorized for db [admin] lock type: 1
",
"errmsg" : "db assertion failure",
"ok" : 0
}
> use admin
switched to db admin
> db.auth("root", "abcd");
1
```

```
> show dbs
admin
local
test
```

زمانی که برای بار اول به پایگاه داده متصل می شویم اجازه ی انجام هیچ گونه عملیات خواندن و نوشتن را نداریم پس از **authenticate** شدن به عنوان یک کاربر **read\_only** اجازه ی انجام کارهای ساده مانند جستجو را داریم زمانی که قصد ورود داده را داریم مجدداً با شکست مواجه می شویم زیرا یک کاربر **read\_only** هستیم کاربر **test\_user** که کاربری دارای مجوز نوشتن است می تواند داده های جدید را به دیتابیس وارد کند اما نمی تواند دستور **show dbs** را اجرا کند که این دستور تنها توسط کاربران ادمین اجرا می شوند

#### نحوه ی کارکرد authentication

زمانی که کاربران به دیتابیس ها اضافه می شوند این کاربران در قالب اسنادی در مجموعه ی **system.users** ذخیره می شوند که ساختار این اسناد به شکل زیر است

```
{"user" : username, "readOnly" : true, "pwd" : password hash}
```

که **password hash** یک **hash** برای نام کاربری و رمز است بدیهی است که برای حذف یک کاربر کافی ست که سند مورد نظر را در مجموعه ی **system.users** پاک کنیم مانند زیر:

```
> db.auth("test_user", "efgh");
1
> db.system.users.remove({"user" : "test_user"});
> db.auth("test_user", "efgh");
0
```

سایر ملاحظات برای برقراری امنیت سیستم گزینه های زیادی برای تعیین سطح دسترسی در **mongoDB** وجود دارد که باید به آنها توجه داشته باشیم در ابتدا زمانی که از **authentication** استفاده می کنیم باید دقت کنیم که پروتکل سیمی **mongoDB** به صورت رمز شده نیست برای این کار باید از کابل های **SSH** یا دیگر مکانیسم های مشابه برای رمز کردن انتقالات بین کاربر و سرور **mongoDB** استفاده کنیم. همیشه سرور **mongoDB** پشت یک فایر وال یا یک شبکه ی قابل دسترسی تنها با سرور برنامه ی ما باشد.

اگر **mongoDB** شما بروی یک ماشینی است که از بیرون اجازه دسترسی دارد در این حالت با استفاده از گزینه ی **--bindip** یک **ip** محلی به شما داده می شود که **mongoDB** شما بروی آن است. برای مثال برای متصل شدن از سرور برنامه تون به **mongod** کافیست دستور **mongod --bindip localhost** استفاده کنیم  
برای غیرفعال کردن اجرای جاوا اسکریپت در سمت سرور در زمان شروع شدن دیتابیس از دستور **noscripting** - استفاده می کنیم



پشتیبان گیری و تعمیر

mongoDB تمام داده ها را در دیتا دایرکتوری ذخیره می کند که عمل پشتیبان گیری به آسانی با کپی کردن این فایل ها انجام می گیرد  
چون ایمنی نیست که از این فایل ها در زمان اجرای mongoDB کپی بگیریم. پس ابتدا سرور mongoDB را خاموش می کنیم و سپس عمل کپی را انجام می دهیم

mongodump and mongorestore

یک روش برای عمل پشتیبان گیری در حین کار mongoDB استفاده از mongodump است



# فصل نهم

# Replicaton

در کپی‌برداری یا Replication، داده‌ها و جداول یک پایگاه‌داده روی چندین سرور قرار می‌گیرد و از طریق فرایندهایی، داده‌های مربوط به پایگاه‌های داده فرعی با داده‌های پایگاه داده اصلی هماهنگ می‌شود. به این ترتیب سیستم نرم‌افزاری استفاده‌کننده از این پایگاه داده، برای دسترسی به داده مورد نظر خود به جای سرور اصلی، به نزدیک‌ترین سرور محلی معرفی شده به آن مراجعه می‌کند. در نتیجه از ترافیک شبکه کاسته می‌شود و سرعت تهیه اطلاعات نیز افزایش می‌یابد. کپی‌برداری به روش پایه / پیرو (Master/Slave) در پایگاه‌داده MySQL برای اولین بار در سال 2000 و در نسخه بتای این پایگاه‌داده عرضه شد.

Replication هسته‌ی مرکزی مدیریت سیستم‌های پایگاه داده در هر دیتابسی می‌باشد در صورتی که بخواهیم داده‌هایمان بعد از عیب یا نقص پایگاه داده دوباره قابل دسترسی باشند به این عامل نیازمندیم

برای اطمینان از اینکه داده‌های تولیدی ما روی بیشتر از یک ماشین وجود دارد نیاز به replication داریم

در حقیقت replication یک توزیع و نگهداری سرور پایگاه داده بروی چندین ماشین است دو نوع replication در MongoDB وجود دارد master-slave-1 و

## replica set-2

در هر دو نوع یک نود اصلی اطلاعات نوشته شده را می‌گیرد سپس بقیه‌ی نودها اطلاعات را می‌خوانند و تلاش می‌کنند این اطلاعات را به صورت غیر همزمان در خود بنویسند

هر دو نوع دارای مکانیسم های مشابهی دارند با این تفاوت که در replica set زمانی که نود اصلی به هر دلیلی از کار بیفتد یکی از نودهای دیگر ترفیع رتبه یافته و جایگزین نود اصلی می شود همچنین دارای قابلیت های دیگری از جمله recovery آسان و توپولوژی های گسترش پیچیده و... بوده است و به همین دلیل اجبار زیادی برای استفاده از master-slave replication نداریم

چرا replication اهمیت دارد؟

هر دیتابیزی در برابر صدمات محیطی که در آن اجرا می شود آسیب پذیر است و در حقیقت replication یک نوع بیمه کردن داده هایمان در برابر این صدمات است برای مثال داریم:

1- اتصال شبکه بین سرور و برنامه مان قطع شود

2- زمان نقص تجهیزات به حدی زیاد باشد که از برگشت سرور جلوگیری کند

و...

## Master-Slave Replication

یکی از عمومی ترین replication ها است که یک برای انجام عملیات backup, failover, read scaling مورد استفاده قرار می گیرد و خیلی انعطاف پذیر است



*A master with one slave*



*A master with three slaves*

که در ابتدا یک نود *master* با یک یا چند نود *slave* ایجاد می شود که هر نود *slave* آدرس نود *master* را می داند .

برای شروع کردن یک *master* دستور `mongod --master` را اجرا می کنیم و برای ایجاد نود *salve* دستور `mongod --slave --source master_address` که آدرس *master* را که ابتدا اجرا کرده ایم را باید بنویسیم

یک مثال ساده که بروی یک ماشین اجرا می شود را مثال می زنیم البته می توان روی چندین ماشین نیز اجرا کنیم

ابتدا یک دایرکتوری برای ذخیره داده های *master* و یک پورت را انتخاب می کنیم

```
$ mkdir -p ~/dbs/master
$ ./mongod --dbpath ~/dbs/master --port 10000 --master
```

حالا شروع به پیکربندی *slave* می کنیم که یک دیتادایرکتوری و پورت جداگانه را برایش انتخاب می کنیم که برای هر *slave* نیاز داریم تا *master* را مشخص کنیم که این کار با گزینه ی *source* تعیین می شود

```
$ mkdir -p ~/dbs/slave
$ ./mongod --dbpath ~/dbs/slave --port 10001 --slave --source
localhost:10000
```

همه ی نودهای *salve* باید به یک نود *master* برگردانده یا *replicate* شوند

هیچ مکانیسمی برای *replicate* کردن نود های *slave* به یک نود *slave* وجود ندارد زیرا نود *slave* *oplog* های خودش را نگه نمی دارد (توضیح داده می شود)

## Options

*Option* های مفیدی در رابطه با *master-slave* وجود دارد که به چندی از آنها اشاره می کنیم

### --only

به روی نود *salve* برای مشخص کردن فقط یک دیتابیس برای *replicate* کردن استفاده می شود (که بصورت پیش فرض برای *replicate* کردن همه ی دیتابیس ها است)

### --slavedelay

به روی نود *salve* برای اضافه کردن یک تاخیر در حد ثانیه است که برای استفاده کردن در زمانی است که عملیات از سوی *master* اجرا شده اند که از وارد کردن داده های نادرست یا پاک کردن ناگهانی داده های مهم جلوگیری می کند

## Adding and Removing Sources

ما می‌توانیم master را زمانی که slave را ایجاد می‌کنیم با استفاده از `--source` مشخص کنیم اما می‌توانیم این کار را با استفاده از پورتهای `mongoDB` انجام می‌دهیم فرض کنید که یک master بروی `localhost:27017` است و ما slave را بدون هیچ سورسی شروع می‌کنیم. سپس master را به مجموعه‌ی `sources` ها اضافه می‌کنیم

```
$ ./mongod --slave --dbpath ~/dbs/slave --port 27018
```

اکنون شروع می‌کنیم به اضافه کردن `localhost:27017` به عنوان سورسی برای slave هایمان که ابتدا shell را باز می‌کنیم و دستور زیر را اجرا می‌کنیم

```
> use local
```

```
> db.sources.insert({"host" : "localhost:27017"})
```

اگر به slave's log نگاه کنیم خواهیم دید که با `localhost:27017` همگام شده است اگر یک جستجویی روی مجموعه‌ی سورس خود بلافاصله بعد از وارد کردن داده داشته باشیم سندی را که وارد کرده ایم را مشاهده می‌کنیم

```
> db.sources.find()
```

```
{
  "_id" : ObjectId("4c1650c2d26b84cc1a31781f"),
  "host" : "localhost:27017"
}
```

یکی از slave log ها خاتمه‌ی همگام سازی را نشان می‌دهد که سند برای نشان داده این همگام سازی به شکل زیر آپدیت می‌شود

```
> db.sources.find()
```

```
{
  "_id" : ObjectId("4c1650c2d26b84cc1a31781f"),
  "host" : "localhost:27017",
  "source" : "main",
  "syncedTo" : {
    "t" : 1276530906000,
    "i" : 1
  }
}
```

```
},  
"localLogTs" : {  
  "t" : 0,  
  "i" : 0  
},  
"dbsNextPass" : {  
  "test_db" : true  
}  
}
```

## Replica set

در حقیقت یک ویرایش روی نوع **master-slave** است. اما تفاوت عمده ای که درارد این است که دیگر در این مدل **master** یکی نیست. به یکیش به عنوان **cluster** انتخاب می شود و می توند به دیگری در اینصورت از کار افتادن تغییر کند.

با این وجود این مدل اینطور به نظر می رسد که دارای یک نود **master** است که **primary** نام دارد و یک یا چند نود **slave** است که **secondaries** نام دارد

ما ابتدا با پیکربندی ساده **replica** آشنا می شویم و کارکرد **replication** بروی **mongoDB** را توضیح می دهیم

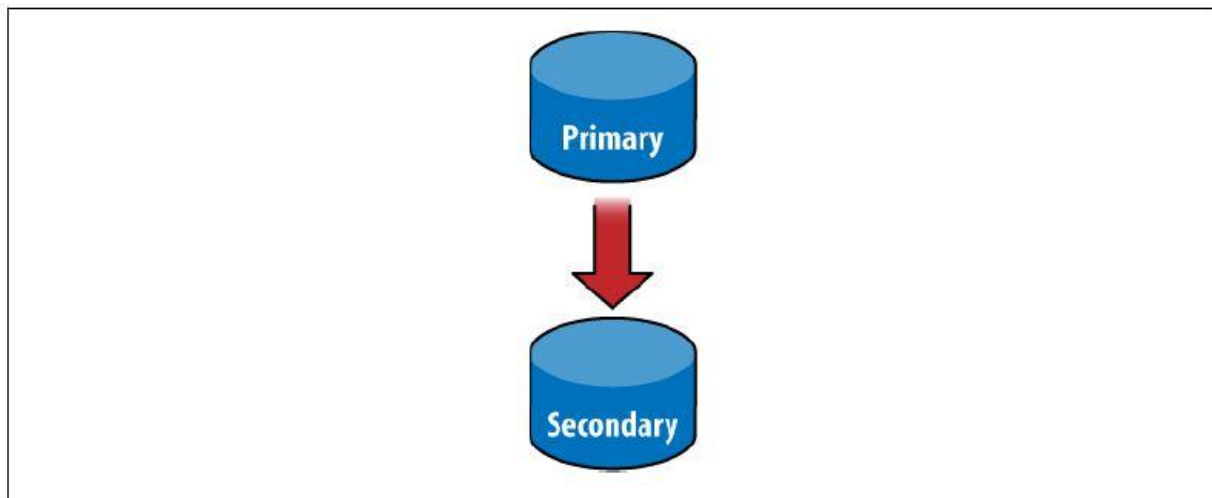


Figure 9-3. A replica set with two members

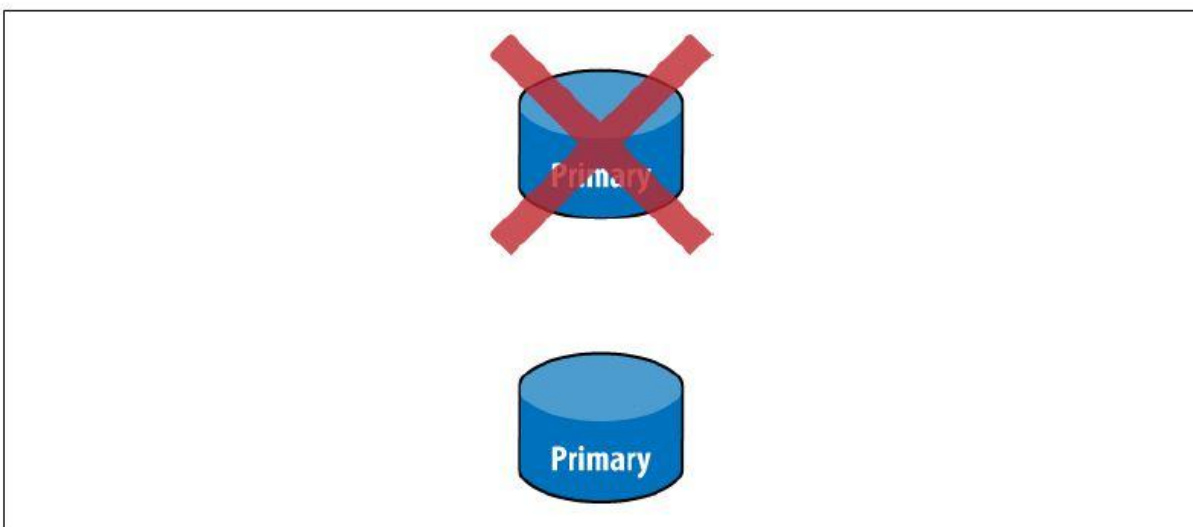


Figure 9-4. When the primary server goes down, the secondary server will become master

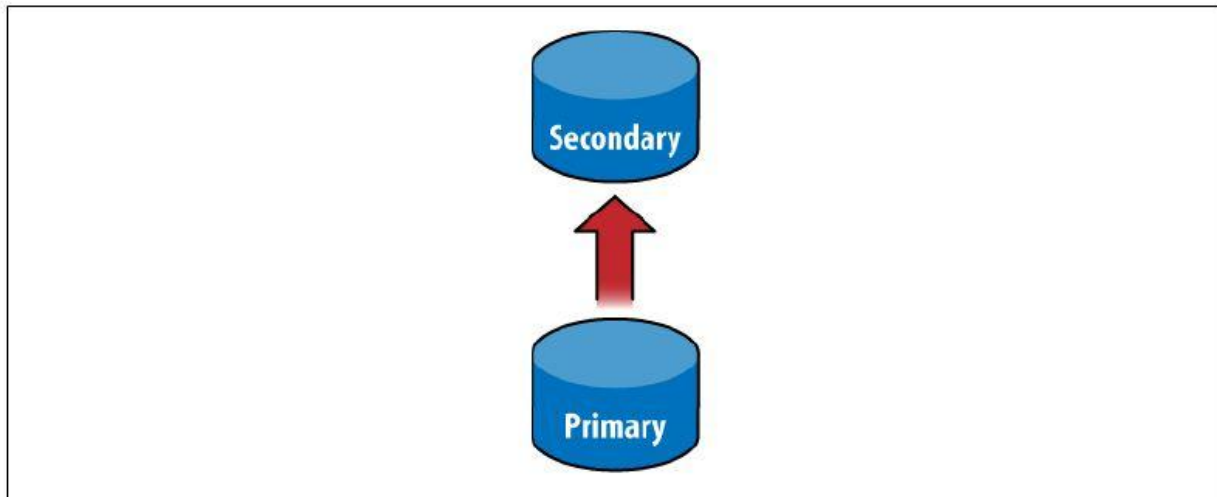


Figure 9-5. If the original primary comes back up, it will begin slaving off of the new primary

## Initializing a Set

تنظیمات replica set کمی پیچیده تر از master-slave است  
ابتدا با تنظیمات یک مجموعه ی ساده و کوچک که شامل دو سرور است بحث خود را  
شروع می کنیم

در ابتدا دیتادایرکتوری را ایجاد می کنیم و پورت هر سرور را مشخص می کنیم

```
$ mkdir -p ~/dbs/node1 ~/dbs/node2
```

قبل از شروع بکار کردن سرور ابتدا یک نام برای replica set تعیین می کنیم.  
Replica set خود را "blort" می نامیم. اکنون سرور خود را شروع بکار می کنیم  
تنها گزینه ی تازه ای که وارد می کنیم --replSet است که اجازه می دهد سرور بداند که  
عضوی از blort است که شامل اعضای دیگر در Morton:10002 نیز است.

```
$ ./mongod --dbpath ~/dbs/node1 --port 10001 --replSet blort/morton:10002
```

سرور های دیگر را در همان مسیر روشن می کنیم

```
$ ./mongod --dbpath ~/dbs/node2 --port 10002 --replSet blort/morton:10001
```

اگر بخواهیم سومین سرور را اضافه کنیم که با یکی از دو دستور زیر این کار را انجام می  
دهیم

```
$ ./mongod --dbpath ~/dbs/node3 --port 10003 --replSet blort/morton:10001
$ ./mongod --dbpath ~/dbs/node3 --port 10003 --replSet
blort/morton:10001,morton:10002
```

یکی از خوبی های replica set توانایی خود شناسایی است که می توانیم یک سرور را به مجموعه اختصاص بدهیم و MongoDB به طور اتوماتیک به همه ی نود ها به طور اتوماتیک متصل شود



ضمیمہ ی

اول

نصب mongoDB بروی اکثر پلتفرم ها کار خیلی آسونی است که نسخه mongo برای پلتفرم های windows و mac os x و linux و حتی solaris وجود دارد که نسخه های هر پلتفرم را می توان از سایت [www.mongodb.org](http://www.mongodb.org) دانلود کرد

زمانی که ما از نصب mongoDB صحبت می کنیم در اصل هدف ما تنظیم سرور یا همان mongod است که این سرور بروی یک سرور که می تواند master یا slave باشد نصب می شود

انتخاب نسخه ی مورد نظر

نسخه های خیلی متنوعی از mongoDB در سایت مربوطه گذاشته شده است که این نسخه ها از نسخه ی 1.6.0 شروع شده و تا الان ادامه دارد به طور مثال

1.6.1 1.6.15 1.7.0 1.7.2 1.7.10 و 2.2.3...

## Windows Install

ابتدا باید توجه کنیم که بتوانیم نسخه ی مورد نظر را با دقت انتخاب کنیم سپس از سایت مورد نظر فایل زیپ شده windows را دانلود می کنیم باید به 32 یا 64 بیت بودن نیز توجه کنیم!

حال نیاز به یک دایرکتوری نیاز داریم که فایل های دیتابیس را در آن ذخیره کنیم

به صورت پیش فرض این فایل را در قسمت c:\data\db ایجاد می کنیم. البته می توانیم این دایرکتوری را در هر جای فایل سیستم خود ایجاد کنیم با این شرط که زمان اجرا باید دایرکتوری را به دایرکتوری خودمون تغییر بدهیم

حال cmd را باز میکنیم و به دایرکتوری که فایل زیپ شده را در آن extract کرده ایم می رویم و وارد bin شده و فایل mongod.exe را اجرا می کنیم

```
$ bin\mongod.exe
```

اگر دایرکتوری ما دایرکتوری پیش فرض نبود در این جا تغییرات را وارد می کنیم

```
$ bin\mongod.exe --dbpath C:\Documents and Settings\Username\My Documents\db
```

## POSIX (Linux, Mac OS X, and Solaris) Install

برای نصب در این پلتفرم ها باید فایل زیپ شده را از سایت مورد نظر دانلود کنیم سپس به طریق زیر مراحل را دنبال می کنیم

ابتدا باید دایرکتوری لازم برای فایل های دیتا را بسازیم به صورت پیش فرض این فایل را در قسمت `\data\db` ایجاد می کنیم. البته می توانیم این دایرکتوری را در هر جای فایل سیستم خود ایجاد کنیم با این شرط که زمان اجرا باید دایرکتوری را به دایرکتوری خودمون تغییر بدهیم

```
$ mkdir -p /data/db  
$ chown -R $USER:$USER /data/db
```

برای باز کردن فایل های `tar.gz` مثل زیر عمل می کنیم

```
$ tar xzf mongodb-linux-i686-1.6.0.tar.gz  
$ cd mongodb-linux-i686-1.6.0
```

دستور `mkdir` دایرکتوری را در صورت نبود با تمام پدرهایش می سارد برای مثال اگر دایرکتوری `\data` و سپس دایرکتوری `\data\db` را ایجاد می کند

`Chown` این امکان را می دهد که بتوانیم اجازه نوشتن در فایل را به یک کاربر خاص بدهیم یا می توانیم یک فولدر را در `home` خود ایجاد کنیم و به `mongoDB` اختصاص دهیم

حال برای شروع به دایرکتوری که فایل زیپ شده را در آن باز کرده ایم می رویم و دستتر زیر را ایجاد می کنیم

```
$ bin/mongod
```

که زمانی که از دایرکتوری متفاوت برای دایرکتوری داده استفاده می کنیم دایرکتوری را در اینجا تغییر می دهیم

```
$ bin/mongod --dbpath ~/db
```





# ضمیمه ی دو Shell

در حالت عادی فرض ما بر این است که ما `mongod` را بر دوی سرور خود و در پورت پیش فرض نصب کرده ایم

اما در صورتی که مانند حالت بالا نباشد ابتدا به سرور و پورت مربوطه وصل می شویم

```
$ bin/mongo staging.example.com:20000
```

به صورت پیش فرض ابتدا به دیتابیس `test` وصل می شویم در صورت اتصال به دیتابیس دیگر از دستور زیر استفاده می کنیم

```
$ bin/mongo localhost:27017/admin
```

دستور بالا به سرور `localhost` بروی پورت `27017` و دیتابیس `admin` متصل می شود. همچنین می توانیم به `shell` بدون اتصال به هیچ پایگاه داده ای متصل شویم از دستور زیر استفاده می کنیم

```
$ bin/mongo --nodb
MongoDB shell version: 1.5.3
type "help" for help
>
```

به یاد داشته باشید که `db` تنها `databaseConnection` نیست و ما می توانیم `connection` های متفاوتی به محیط های متفاوتی که سرور در آنها قرار دارد داشته باشیم.

برای مثال با استفاده از `sharding` می توانیم چندین `connection` داشته باشیم

```
> mongos = connect("localhost:27017")
connecting to: localhost:27017
localhost:27017
> shard0 = connect("localhost:30000")
connecting to: localhost:30000
localhost:30000
> shard1 = connect("localhost:30001")
connecting to: localhost:30001
localhost:30001
```

بنابراین ما می توانیم از `mongos` و `shard0` و `shard1` به عنوان متغیرهای `db` استفاده کنیم

## Shell Utilities

با استفاده از shell می توانیم به چندین سرور متصل شویم

```
> shard_db = connect("shard.example.com:27017/mydb")  
connecting to shard.example.com:27017/mydb  
mydb  
>
```

همچنین می توانیم دستورات مربوط به shell را اجرا کنیم

```
> runProgram("echo", "Hello", "world")  
shell: started mongo program echo Hello world  
0  
> sh6487| Hello world
```

و بسیاری از کاربرد های جالب shell وجود دارد که باید به manual آن مراجعه کنید

### *buildInfo*

```
{"buildInfo" : 1}
```

دستور که توسط ادمین اجرا می شود و اطلاعاتی در مورد mongoDB و شماره ی نسخه ی سرور و سیستم عامل میزبان می دهد

### *collStats*

```
{"collStats" : collection}
```

اطلاعاتی در مورد مجموعه ها سایز داده ها و فضای ذخیره سازی و سایز ایندکس ها می دهد

### *Distinct*

```
{"distinct" : collection, "key": key, "query": query}
```

ارز شهای متفاوت کلید های یافت شده در پرس و جوی جاری را می دهد

### *Drop*

```
{"drop" : collection}
```

همه ی اطلاعات درون یک مجموعه را حذف می کند

### *dropDatabase*

```
{"dropDatabase" : 1}
```

همه ی داده ها را از دیتابیس جاری حذف می کند

### *dropIndexes*

`{"dropIndexes" : collection, "index" : name}`

همه ی ایندکس ها با نام `name` را از مجموعه حذف می کند در صورتی که نام "\*" باشد  
همه ی ایندکس ها را حذف می کند

### *getLastError*

`{"getLastError" : 1}`

خطاها و سایر وضعیت های ایجاد شده تحت آخرین عملیات را نمایش می دهد

```
> db.count.update({x : 1}, {$inc : {x : 1}}, false, true)
```

```
> db.runCommand({getLastError : 1})
```

```
{  
  "err" : null,  
  "updatedExisting" : true,  
  "n" : 5,  
  "ok" : true  
}
```

دستور بالا تعداد اسنادی را که تحت تاثیر آپدیت قرار گرفته اند را نمایش می دهد

### *isMaster*

`{"isMaster" : 1}`

سرور را چک می کند تا ببیند که `master` است یا `slave`

### *listCommands*

`{"listCommands" : 1}`

لیست دستورات موجود بروی سرور جاری را همراه با توضیحاتی در مورد آنان را نمایش می دهد

## *listDatabases*

**{"listDatabases" : 1}**

دستوری که توسط ادمین اجرا می شود و لسیت دیتابیس ها را به ما می دهد

## *Ping*

**{"ping" : 1}**

وجود داشتن سروری را چک می کند این دستور به سرعت نتیجه را برمی گرداند حتی اگر سرور قفل شده باشد

## *renameCollection*

**{"renameCollection" : a, "to" : b}**

مجموعه ی a را به b تغییر نام می دهد و باید هردو دارای نام کامل باشند برای مثال foo.bar به مجموعه ی bar در دیتابیس foo اشاره دارد

## *repairDatabase*

**{"repairDatabase" : 1}**

پایگاه داده ی جاری را تعمیر و فشرده می کند که ممکن است یک دستور با مدت زمان اجرای طولانی باشد

## *serverStatus*

**{"serverStatus" : 1}**

آمار های اجرایی برای سرور جاری را می دهد

# ضمیمه ی 3

## Wire Protocol

درایورهای که به سرور **mongoDB** دستیابی دارند از یک پروتکل سیمی **TCP/IP** استفاده می نمایند که به صورت اساسی شامل یک پوشش ضخیم دور داده های **BSON** است. برای مثال برای یک دستور **insert** نیاز به 20 بایت هدر دیتا و نام مجموعه و لیست اسناد **BSON** برای ورود به مجموعه ی مورد نظر داریم

## Data Files

علاوه بر وجود دیتا دایرکتوری که بصورت پیش فرض در **data/db** موجود است. فایل های جداگانه دیگری برای هر پایگاه داده وجود دارد هر دیتابیس شامل یک فایل **.ns** و چند فایل دیتا است که به طور یکنواخت تعداد آن ها در حال گسترش است  
بنابراین دیتابیس **foo** حاوی یک غایل بنام **foo.ns** و فایل های **foo.0** و **foo.1** و **foo.2** و غیره می باشد.

اندازه ی فایل های داده برای یک پایگاه داده بدین صورت است که هر فایل دو برابر می شود تا به اندازه ی ماکزیمم 2 گیگ بایت خود برسد

این رفتار باعث می شود که در ذخیره سازی دیتابیس های کوچک مشکل به هدر رفتن فضاهای روی دیسک و فضاهای زاید را نداشته باشیم و دیتابیس های بزرگ در مکان های نزدیک هم ذخیره شوند

mongoDB هم چنین دیتا فایل های را از قبل به دیتابیس ها اختصاص می دهد که این کار باعث کارایی بهتر سیستم می شود که برای غیرفعال کردن این قابلیت کافی ست دستور --noproalloc را اجرا کنیم

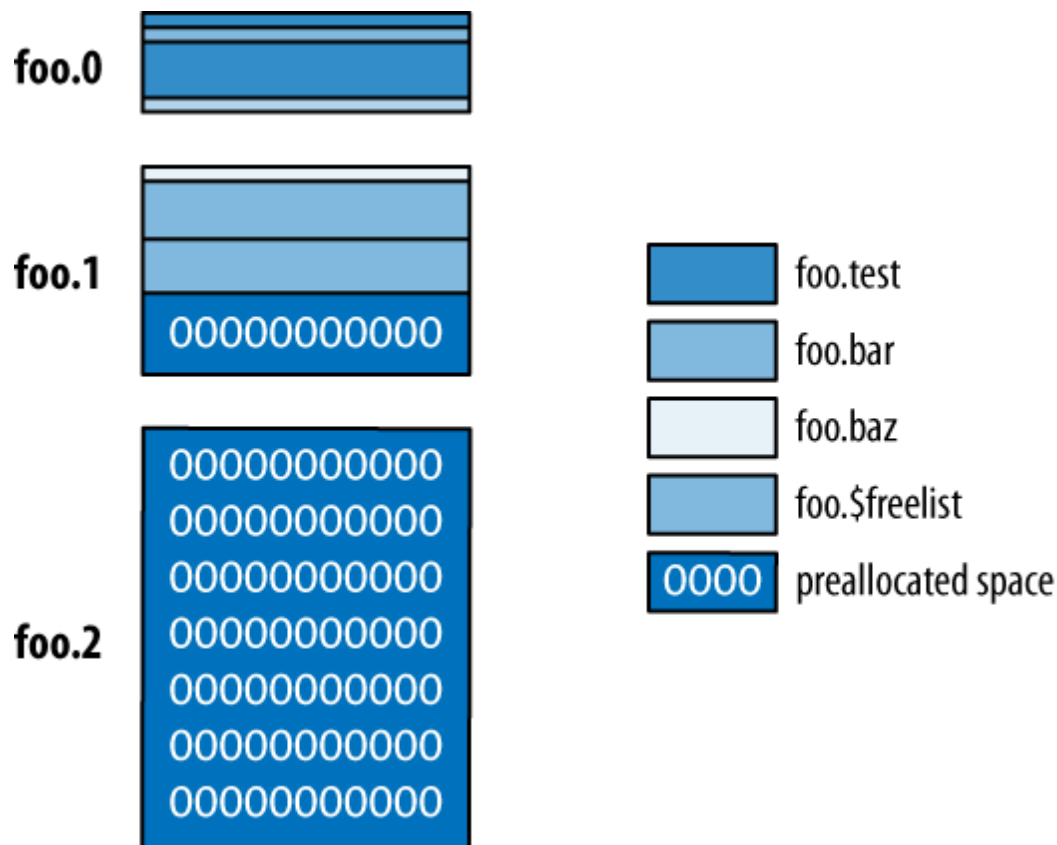
اختصاص دادن در پشت زمینه ی mongoDB و زمانی که فایلی پر شود صورت می گیرد. این بدان معنی است که سرور mongoDB دیتافایل های اضافی و خالی را برای هر دیتابیس نگهداری می کند و از استفاده شدن آنها برای تخصیص دادن فضا جلوگیری می کند

## Namespaces and Extents

در دیتافایل ها هر دیتابیس در قالب یک namespace سازماندهی می شود. اسنادی که به یک مجموعه متعلق دارد دارای فضای نام مخصوص به خودش را دارد

اطلاعات مربوط به هر فضای نام در روی دیسک در section های از دیتافایل ذخیره می شود که extent نام دارد

در شکل زیر در دیتابیس foo سه دیتا فایل وجود دارد که هر یه از قبل اختصاص داده شده اند ولی خالی می باشند. دو تای اول از دیتافایل ها به extents هایی که به چندین فضای نام تعلق دارند تقسیم شده اند



چند نکته ی جالب در مورد شکل بالا وجود دارد:

هر فضای نام می تواند دارای **extents** هایی از نوع متفاوت باشد که ازوما در روی دیسک به صورت پیوسته قرار ندارد

همچنین نشان می دهد که **\$freelist** فضای نامی است که **track** های **extents** هایی که به مدت زمان زیادی استفاده نشده را نگه می دارد

زمانی که به یک فضای نام **extents** تازه ای اختصاص داده می شود ابتدا **freelist** را برای پیدا کردن یک فضای مناسب با **extents** را جستجو می کند

مراجع:

wowebook\_Oreilly.MongoDB  
.The.Definitive.Guide.Sep.20  
10 –

Kyle Banker — MongoDB in  
Action — 2011

MongoDB-Manual

