

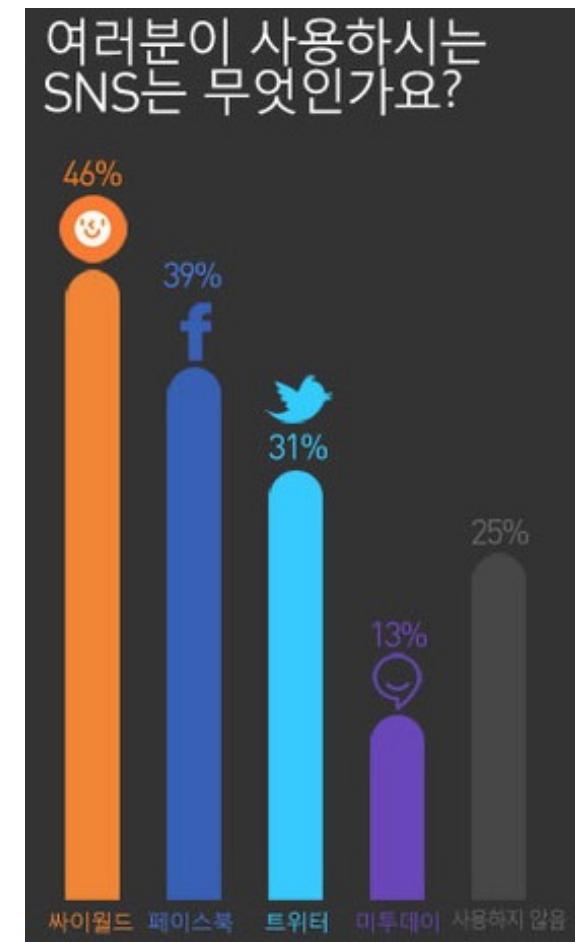
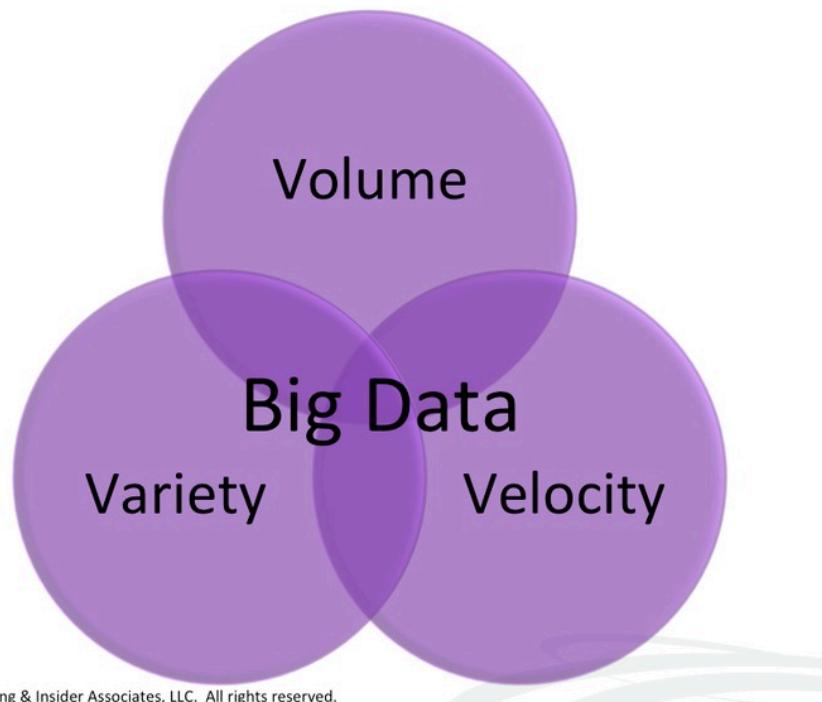
## 빅데이터 활용 - MongoDB프로그래밍



# **1** NoSQL & mongoDB

# 빅데이터 개념

- 기존 데이터에 비해 너무 방대해 이전 방법이나 도구로 수집, 저장, 검색, 분석, 시각화 등이 어려운 정형 또는 비정형 데이터 세트를 의미한다.



# NoSQL

- **OS (Operation System)**
  - Windows
  - Linux
  - Unix 등
- **RDB (Relation-DataBase)**
  - 오라클
  - Mssql
  - Mysql 등
- **NoSQL (Not Only SQL)**
  - Non Relational Operational Database SQL
  - MongoDB
  - Cassandra
  - Hbase 약 150여개



# NoSQL의 제품

- NoSQL 150개 제품 ( 2012년 12월 1일 )
  - <http://nosql-database.org>



Ecosistema NoSQL

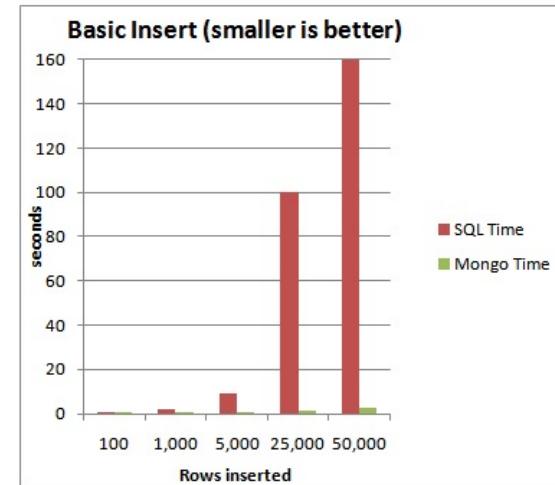
# NoSQL의 장점 / 한계

- **장점**
  - 클라우드 환경에 적합
    - Open Source
    - 하드웨어 확장에 유연한 대처가 가능함
    - 무엇보다 DBMS의 비해 저렴한 비용으로 분산 처리와 병렬처리 가능
  - 유연한 데이터 모델
    - 비정형 데이터 구조 설계로 설계 비용이 감소
    - 관계형 데이터베이스의 Relationship과 Join 구조를 Linking과 Embedded로 구현하여 성능이 빠름
  - 빅데이터 환경에 효과적
    - 메모리 맵핑 기능을 통해 read/write가 빠름
    - 전형적인 OS와 Hardware에 구축할 수 있음
    - 기존 RDB와 동일하게 데이터 처리가 가능
- **한계**
  - 비즈니스 차원에서 현실적으로는 불가능한 시스템이다. 왜냐하면, 태생이 다르기 때문이다.
    - 클라이언트/서버 환경 : 관계형 데이터베이스
    - 클라우드 컴퓨팅 환경 : NoSQL

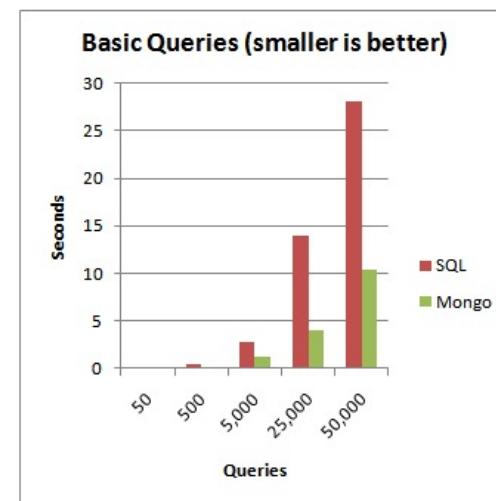
# MongoDB vs SQL Server 성능 분석

- 쓰기 작업 성능 비교

Number of Parallel Clients		Time in seconds					
Basic Insert	Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec	
several columns	100	20	0.19	0.011	526	9,091	
600 bytes per row	1,000	200	1.8	0.02	556	50,000	
	5,000	1,000	9	0.25	556	20,000	
	25,000	5,000	100	1.5	250	16,667	
	50,000	10,000	270	2.5	185	20,000	



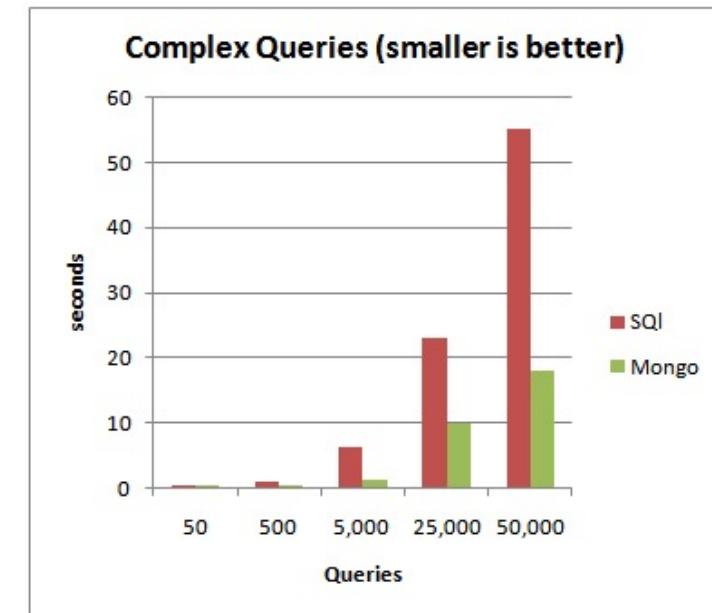
Number of Parallel Clients		Time in seconds					
Basic Query	Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec	
with index	50	10	0.1	0.08	500	625	
	500	100	0.38	0.1	1,316	5,000	
	5,000	1,000	2.8	1.2	1,786	4,167	
	25,000	5,000	14	4	1,786	6,250	
	50,000	10,000	28	10.4	1,786	4,808	



# MongoDB vs SQL Server 성능 분석

- 읽기 / 쓰기 동시 작업 성능 비교

Number of Parallel Clients	5	Time in seconds				
Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec	
Complex / Real w query (2 joins in S embedded doc in MongoDB)	50 500 5,000 25,000 50,000 1,000,000	0.1 1 6.4 23 55 960	0.066 0.19 1.2 9.9 18 398	500 500 781 1,087 909 1,042	758 2,632 4,167 2,525 2,778 2,513	
	10 100 1,000 5,000 10,000 200,000					



참고 : <http://viethip.com/category/db/page/7/>

# 2 MongoDB의 이해 / 설치

# MongoDB의 이해

- **MongoDB란?**

- 10gen 회사에서 기술 개발 및 지원
- JSON 타입으로 데이터를 저장. 자바스크립트 형태의 데이터 표현 방식
  - 데이터 표현 시 괄호를 열고 필드명과 콜론(:) 그리고 데이터 값을 표현
  - 예) { ename : "홍길동" }
- 분산 및 복제 기능 제공. 분산/병렬처리 가능
  - 빅 데이터를 처리하기 위함
  - 장애가 발생하더라도 피해를 최소로 줄이기 위함.
  - 빅데이터에 대한 빠른 추출이 가능.
- 관계형 데이터베이스의 주요 기능을 사용함. CRUD 위주의 다중 트랜잭션 처리.
  - 트랜잭션 위주의 데이터 처리 가능
  - NoSQL은 빅데이터의 빠른 저장과 추출 및 분석을 용이하기 함.

# MongoDB의 이해

- 관계형 데이터베이스와 MongoDB의 논리적 용어 비교

SQL 사용 용어	MongoDB 사용 용어
데이터베이스(database)	데이터베이스(database)
테이블(table)	컬렉션(collection)
행(row)	문서(document) 또는 BSON 문서
열(column)	필드(field)
색인(index)	색인(index)
테이블 조인(table joins)	임베디드 문서 & 링킹(linking)
기본(주) 키(primary key, 유일한 고유 칼럼)	기본(주) 키(primary key, _id 필드 자동 생성)
집합(aggregation, 예: group by)	집합(aggregation) 프레임워크

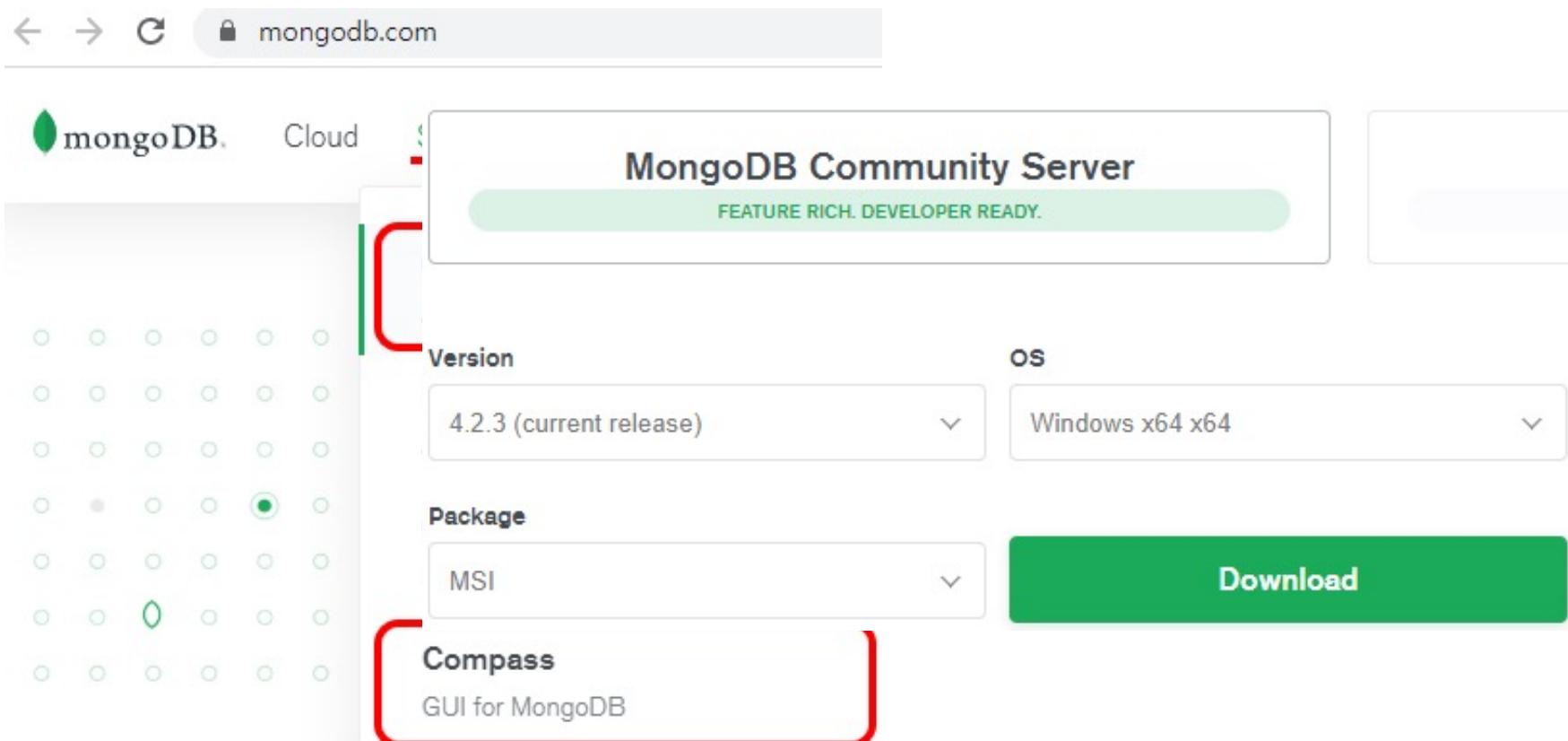
# MongoDB 설치

- 설치 환경 및 지원 드라이버
  - MongoDB 버전 : 2013년 7월 현재 2.4버전
  - 설치 가능한 플랫폼
    - 윈도우 32비트 / 64비트
    - 리눅스 32비트 / 64비트
    - 유닉스 솔라리스 32비트 / 64비트
    - MAC OS X-32비트 / 64비트
  - 지원 드라이버
    - c / c# / c++
    - java / java Script
    - Perl / PHP / Python
    - Ruby / Erlang / Haskell / Scala

# MongoDB 설치

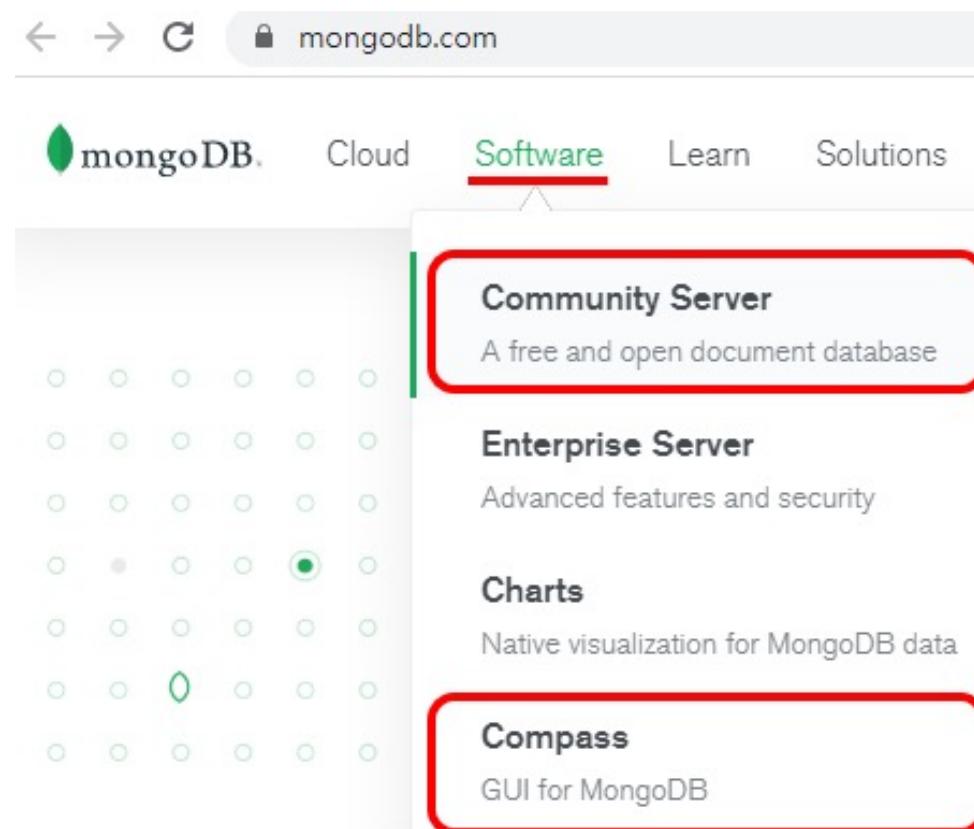
- 다운로드

- [www.mongodb.org](http://www.mongodb.org)



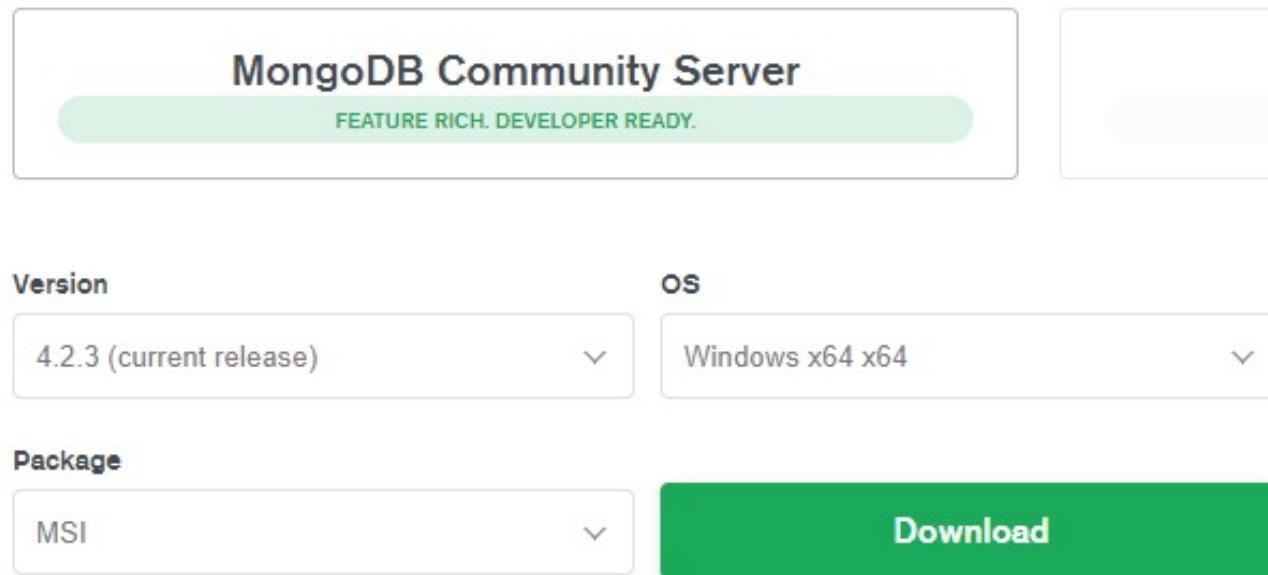
# MongoDB 설치

- 다운로드
  - [www.mongodb.org](http://www.mongodb.org)



# MongoDB 설치

- **다운로드**
  - 주의 : 각 운영체제 비트 확인 후 다운로드



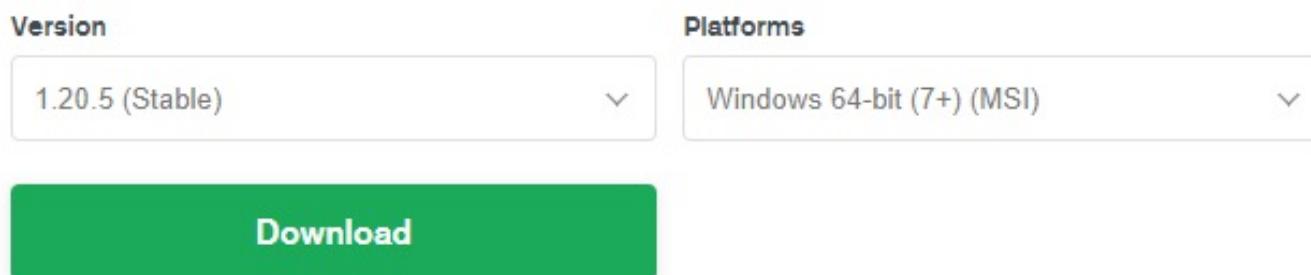
# MongoDB 설치

- **다운로드**
  - 주의 : 각 운영체제 비트 확인 후 다운로드

## MongoDB Compass

As the GUI for MongoDB, MongoDB Compass allows you to make smarter decisions about document structure, querying, indexing, document validation, and more. Commercial subscriptions include technical support for MongoDB Compass.

MongoDB Compass is available in several versions, described below. For more information on version differences, see the [MongoDB Compass Documentation](#).



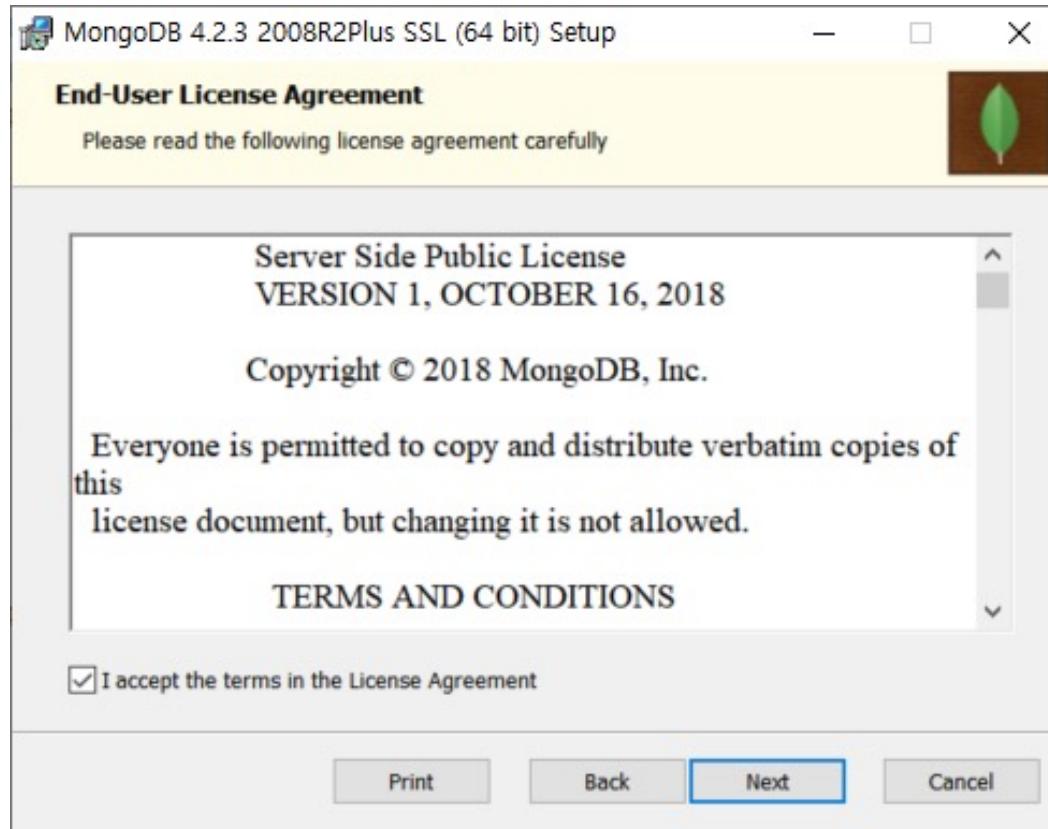
# MongoDB 설치 따라하기

- 설치 요령



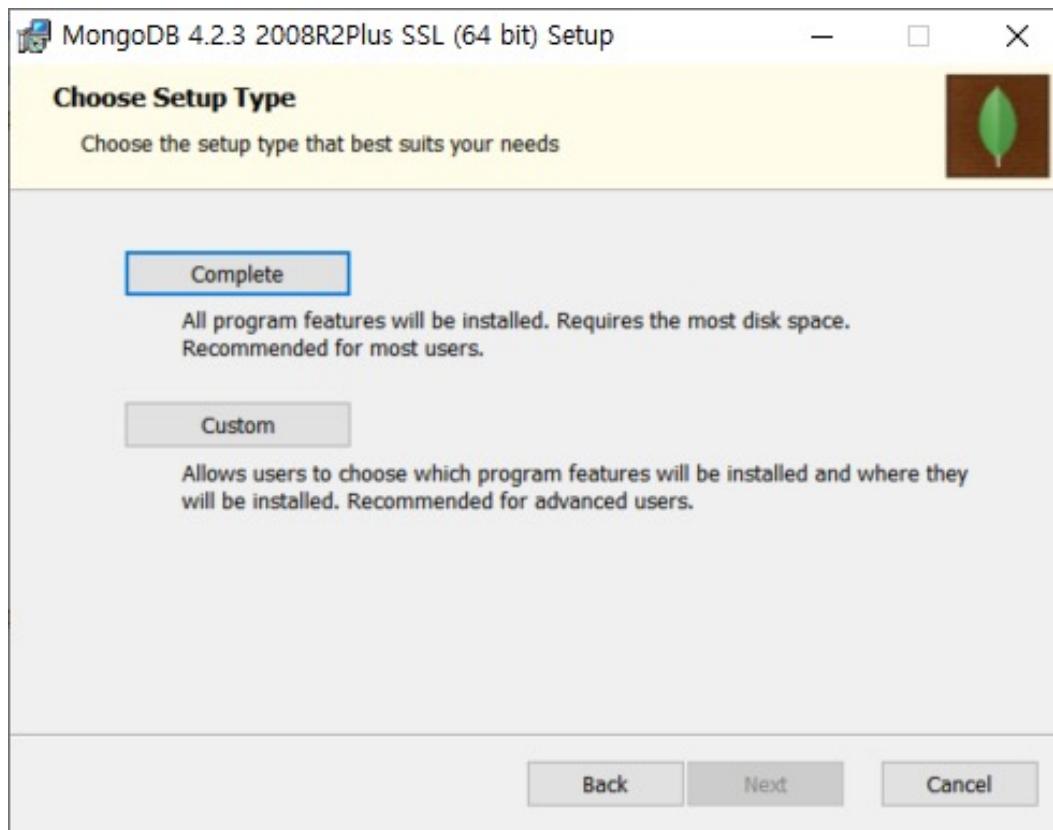
# MongoDB 설치 따라하기

- 설치 요령



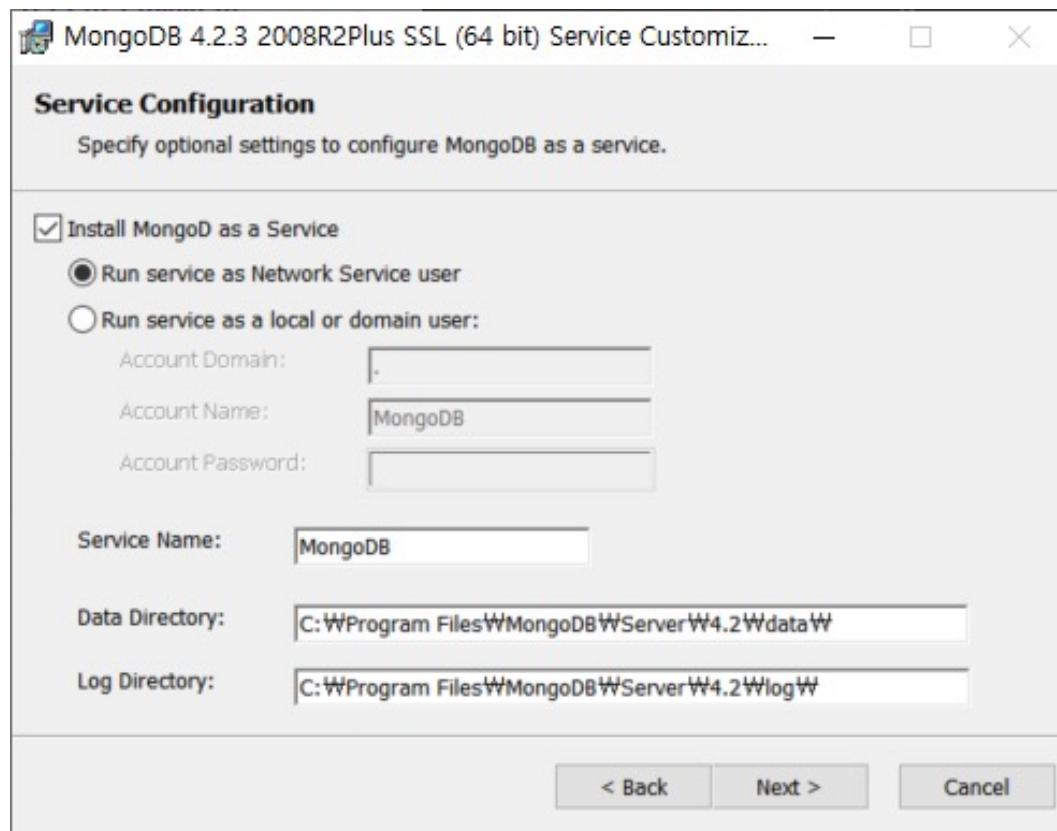
# MongoDB 설치 따라하기

- 설치 요령



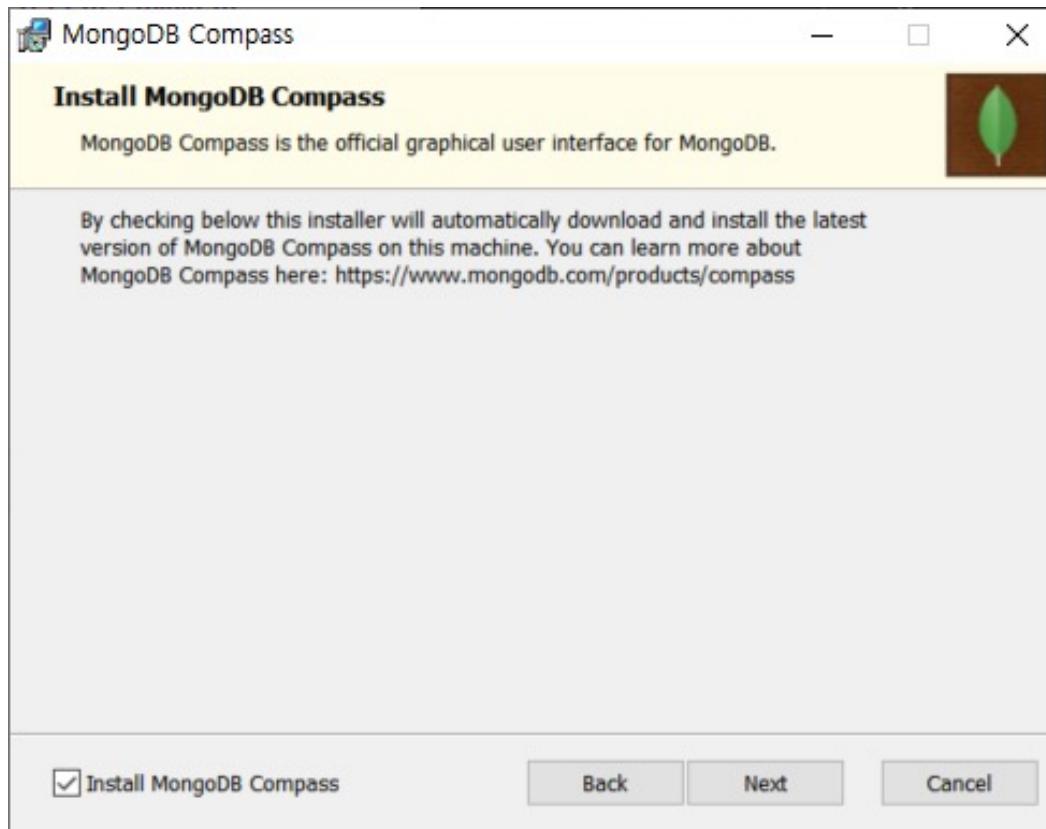
# MongoDB 설치 따라하기

- 설치 요령



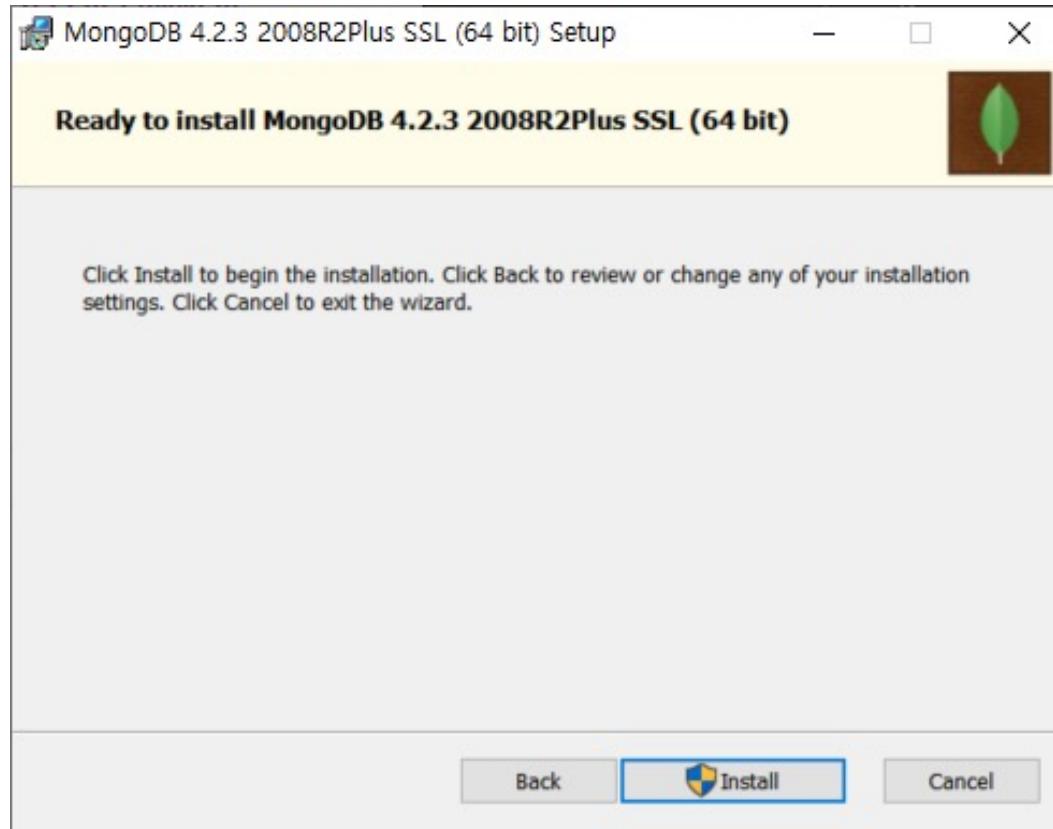
# MongoDB 설치 따라하기

- 설치 요령



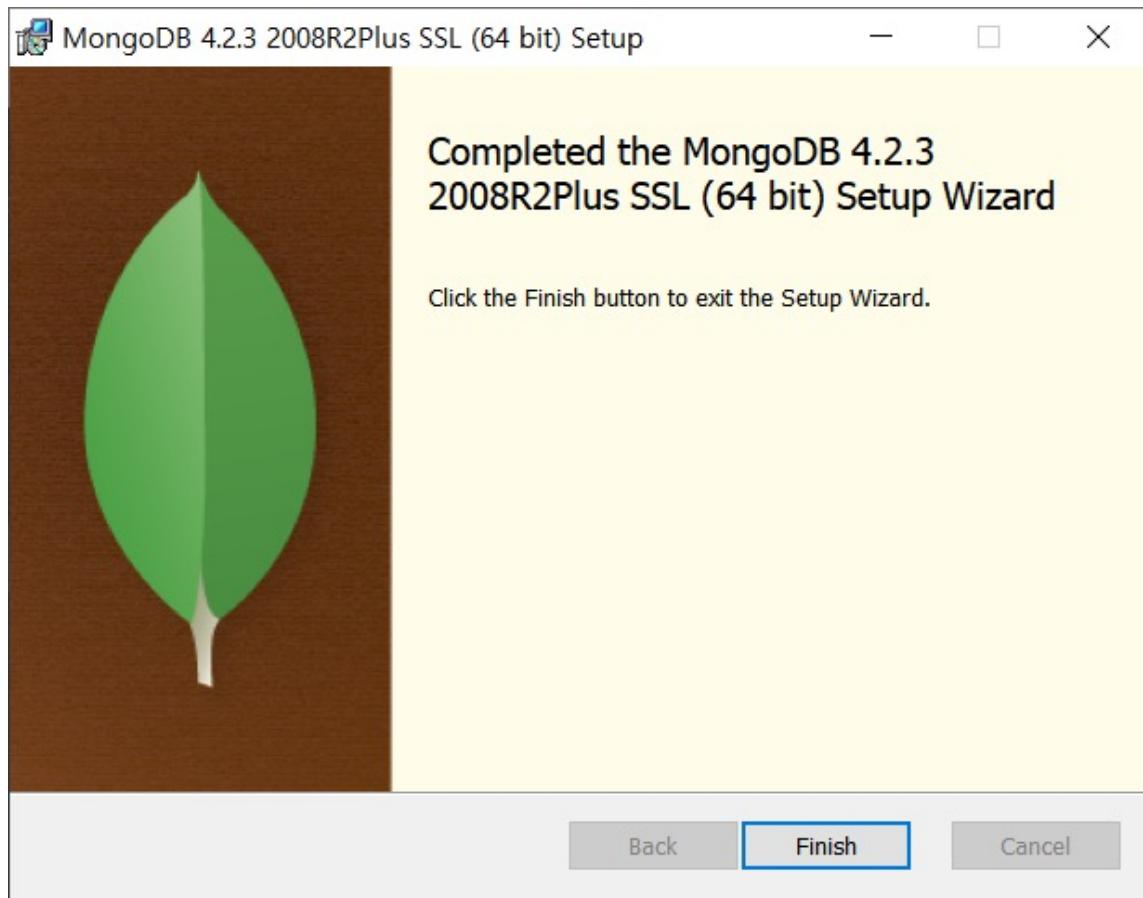
# MongoDB 설치 따라하기

- 설치 요령



# MongoDB 설치 따라하기

- 설치 요령



# MongoDB 설치 따라하기

- 설치 요령

- 시스템 속성 정보 등록
  - 어떤 위치에서도 Mongo Shell을 실행할 수 있도록 System Path에 MongoDB의 bin 폴더 등록
  - [제어판] → [시스템 및 보안] → [시스템] → [고급 시스템 설정]
  - [시스템 속성]에서 [환경변수] 버튼 클릭
  - [시스템 변수]에서 [path] 변수 더블 클릭
  - [시스템 변수 편집]에서 현재 등록되어 있는 [변수값]의 마지막 부분에 다음 내용 추가



# 3

## **mongoDB에서의 데이터처리**

# MongoDB에서의 데이터 처리

- **시작과 종료**

- 메모리 영역, 파일 영역, 프로세서 영역 활성화
  - 데이터가 저장될 물리적인 저장 경로 설정
  - mkdir d:\MongoDB\test
- MongoDB 버전 확인
  - mongod --version 실행

The screenshot shows a Windows Command Prompt window titled "관리자: C:\Windows\system32\cmd.exe". It displays a file listing from a directory containing MongoDB executables. The listing includes files like bsondump.exe, mongo.exe, mongod.exe, mongod.pdb, mongodump.exe, mongoexport.exe, mongofiles.exe, mongoimport.exe, mongooplog.exe, mongoperf.exe, mongorestore.exe, mongos.exe, mongos.pdb, mongostat.exe, and mongotop.exe. The total size of the files is 352,808,448 bytes. Below the file listing, the command "D:\MongoDB\bin>mongod --version" is run, resulting in the output "db version v2.4.5" and the git version information "Tue Jul 16 10:41:13.245 git version: a2ddc68ba7c9cee17bfe69ed840383ec3506602b".

```
D:\MongoDB\bin>mongod --version
db version v2.4.5
Tue Jul 16 10:41:13.245 git version: a2ddc68ba7c9cee17bfe69ed840383ec3506602b
D:\MongoDB\bin>
```

# MongoDB에서의 데이터 처리

- 시작과 종료

- mongoDB 서버 실행

- MONGOD.EXE 파일 실행
    - mongodb/bin 폴더에서 실행
    - 예) mongod --dbpath d:\MongoDB\test
    - 여기서 dbpath는 데이터가 저장될 물리적인 공간 지정함.
    - 32bit 운영체제에서는 반드시 --journal 을 옵션으로 구현

- 클라이언트 프로그램으로 mongoDB에 접속

- MONGO.EXE 파일 실행
    - mongoDB/bin 폴더에서 실행

```
관리자: C:\Windows\system32\cmd.exe - mongod --dbpath d:\MongoDB\test
D:\MongoDB\bin>mongod --dbpath d:\MongoDB\test
Tue Jul 16 10:48:22.652 [initandlisten] MongoDB starting : pid=5528 port=27017 dbpath=d:\MongoDB\test 64-bit host=DESKTOP-RCI
Tue Jul 16 10:48:22.652 [initandlisten] db version v2.4.5
Tue Jul 16 10:48:22.652 [initandlisten] git version: a2ddc68ba7c9cee17bfe69ed840
383ec3506602b
Tue Jul 16 10:48:22.652 [initandlisten] build info: windows sys.getwindowsversion[major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1'] BOOST_LIB_VERSION=1_49
Tue Jul 16 10:48:22.652 [initandlisten] allocator: system
Tue Jul 16 10:48:22.652 [initandlisten] options: { dbpath: "d:\MongoDB\test" }
Tue Jul 16 10:48:22.683 [initandlisten] journal dir=d:\MongoDB\test\journal
Tue Jul 16 10:48:22.683 [initandlisten] recover : no journal files present, no recovery needed
Tue Jul 16 10:48:22.792 [FileAllocator] allocating new datafile d:\MongoDB\test\local.ns, filling with zeroes...
Tue Jul 16 10:48:22.792 [FileAllocator] creating directory d:\MongoDB\test\_tmp
Tue Jul 16 10:48:22.839 [FileAllocator] done allocating datafile d:\MongoDB\test\local.ns, size: 16MB, took 0.04 secs
Tue Jul 16 10:48:22.839 [FileAllocator] allocating new datafile d:\MongoDB\test\local.0, filling with zeroes...
Tue Jul 16 10:48:22.932 [FileAllocator] done allocating datafile d:\MongoDB\test\local.0, size: 64MB, took 0.087 secs
Tue Jul 16 10:48:22.932 [initandlisten] command local.$cmd command: { create: "
```

```
관리자: C:\Windows\system32\cmd.exe - mongo
2013-07-10 오오후후 05:57 173 11.txt
2013-07-10 오오후후 05:05 11,918 2013년년 1기기 프프로로젝트트.xls
2013-07-16 오오전전 10:38 <DIR> mongoDB
2013-07-16 오오전전 10:21 <DIR> mongoDB 자자료료
2013-07-15 오오후후 05:48 <DIR> 공공유유풀폴더더
2013-07-15 오오전전 10:46 <DIR> 네네트트워워크크프프로로그그래밍밍
2013-07-12 오오후후 05:23 <DIR> 이이핵핵서서
2개개 파일파일일 12,091 바바이이트트
5개개 디디렉렉터터리리 462,548,279,296 바바이이트트 남남을을
D:\>cd mongoDB
D:\mongoDB>cd bin
D:\mongoDB\bin>mongo
MongoDB shell version: 2.4.5
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
>
>
>
```

# MongoDB에서의 데이터 처리

- **시작과 종료**

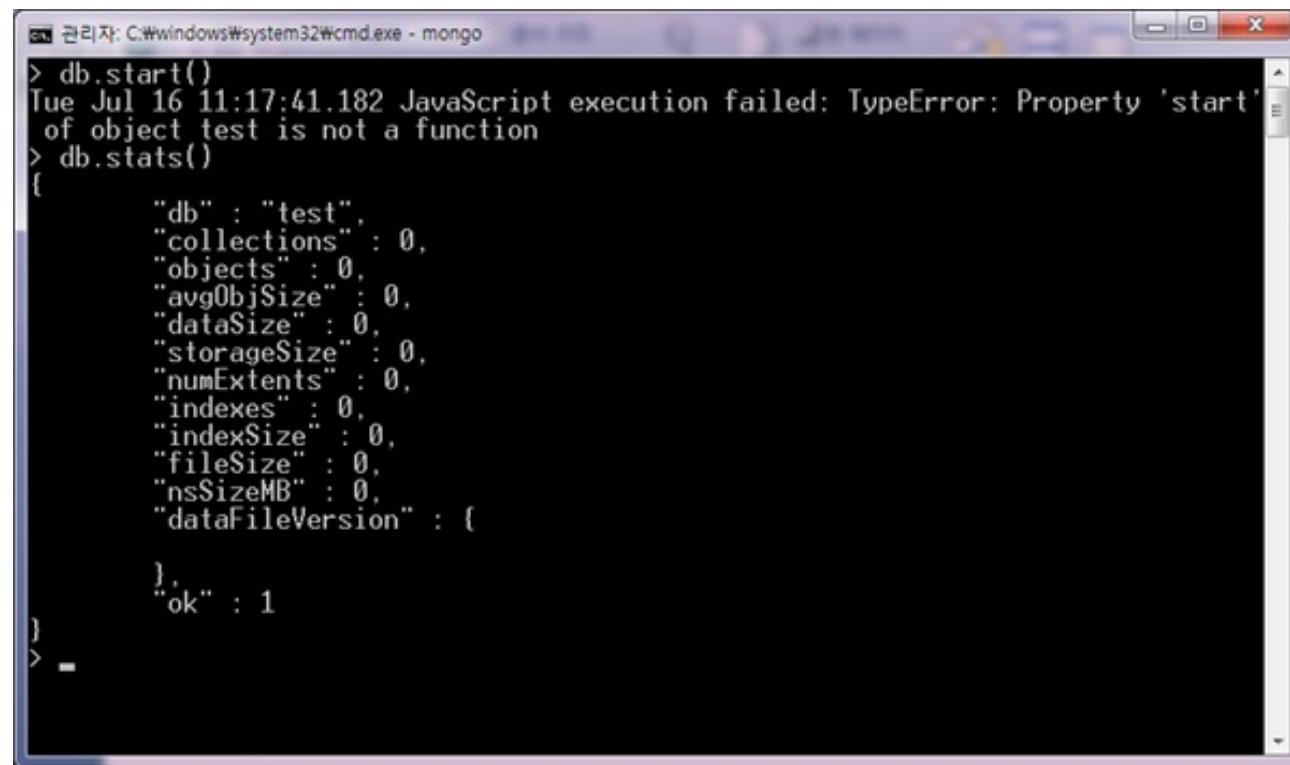
- help : shell 상태에서 실행할 수 있는 명령어 help 기능
- 생성되어 있는 DB 확인
  - show dbs
  - 데이터베이스가 존재하지 않을 경우 첫번째 컬렉션을 생성할 때 자동으로 생성됩니다.

```
> help
      db.help()                      help on db methods
      db.mycoll.help()                help on collection methods
      sh.help()                       sharding helpers
      rs.help()                       replica set helpers
      help admin                      administrative help
      help connect                    connecting to a db help
      help keys                       key shortcuts
      help misc                       misc things to know
      help mr                          mapreduce

      show dbs                        show database names
      show collections                show collections in current database
      show users                      show users in current database
      show profile                    show most recent system.profile entries with
                                     time >= 1ms
      show logs                       show the accessible logger names
      show log [name]                 prints out the last segment of log in memory
                                     'global' is default
      use <db_name>                  set current database
      db.foo.find()                   list objects in collection foo
      db.foo.find( { a : 1 } )        list objects in foo where a == 1
      it                             result of the last line evaluated; use to f
                                     urther iterate
      DBQuery.shellBatchSize = x     set default number of items to display on s-
```

# MongoDB에서의 데이터 처리

- **시작과 종료**
  - 데이터가 저장되어 있는 논리적인 구조 확인
    - db.stats()
    - 데이터베이스명, 컬렉션수, 전체 객체 수, 객체의 평균 길이, 전체 데이터 길이, 데이터베이스에게 할당된 전체 공간, 익스텐드수, 인덱스의 개수, 인덱스 크기, 데이터 파일의 크기, Namespace 크기, 문장의 실행(정상 1, 실패 0)

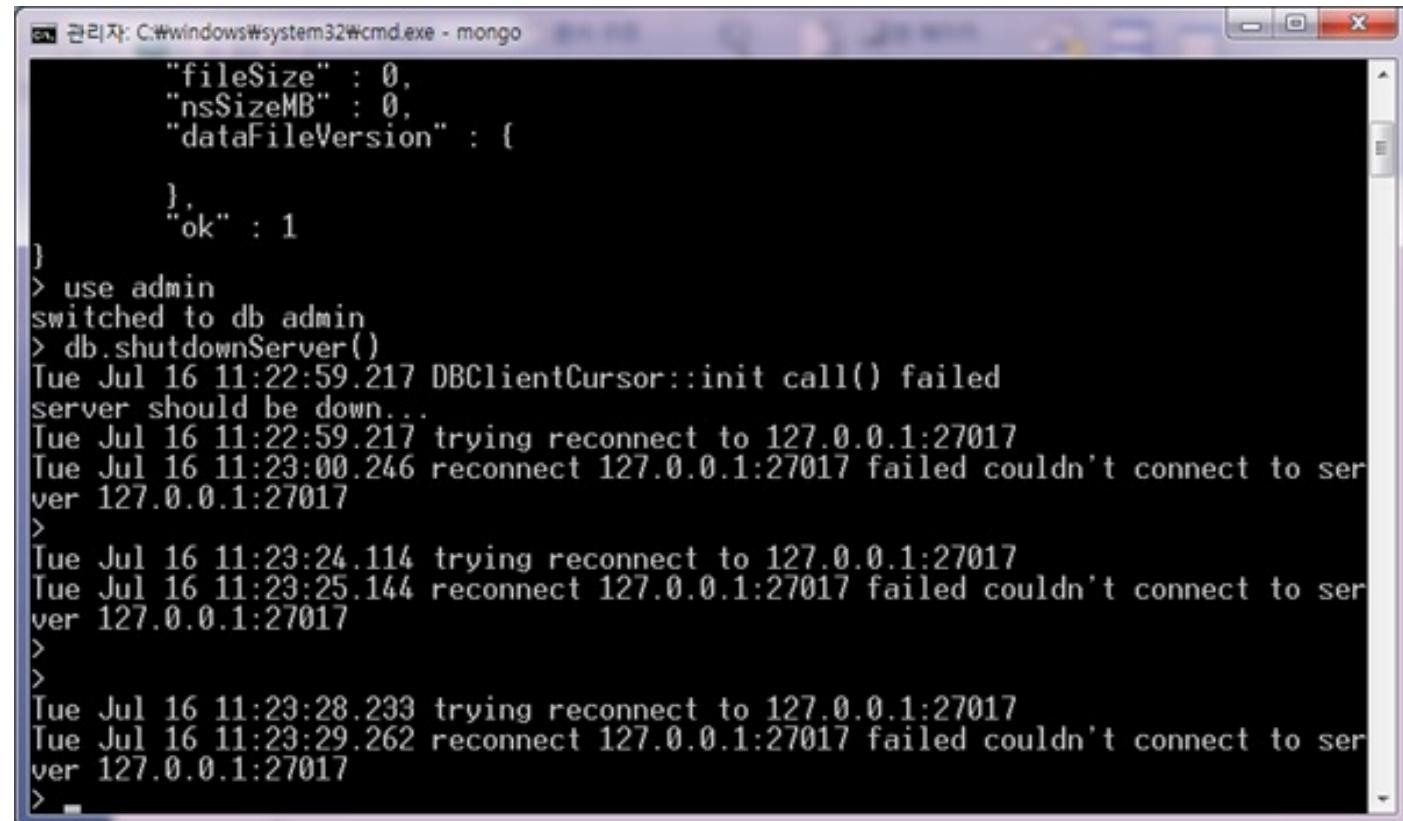


```
관리자: C:\Windows\system32\cmd.exe - mongo
> db.start()
Tue Jul 16 11:17:41.182 JavaScript execution failed: TypeError: Property 'start' of object test is not a function
> db.stats()
{
  "db" : "test",
  "collections" : 0,
  "objects" : 0,
  "avgObjSize" : 0,
  "dataSize" : 0,
  "storageSize" : 0,
  "numExtents" : 0,
  "indexes" : 0,
  "indexSize" : 0,
  "fileSize" : 0,
  "nsSizeMB" : 0,
  "dataFileVersion" : {
    ...
  },
  "ok" : 1
}>
```

# MongoDB에서의 데이터 처리

- **시작과 종료**

- mongoDB 인스턴스 종료
    - shutdown시 반드시 admin 데이터베이스로 이동.
    - db.shutdownServer( ) 명령어 실행

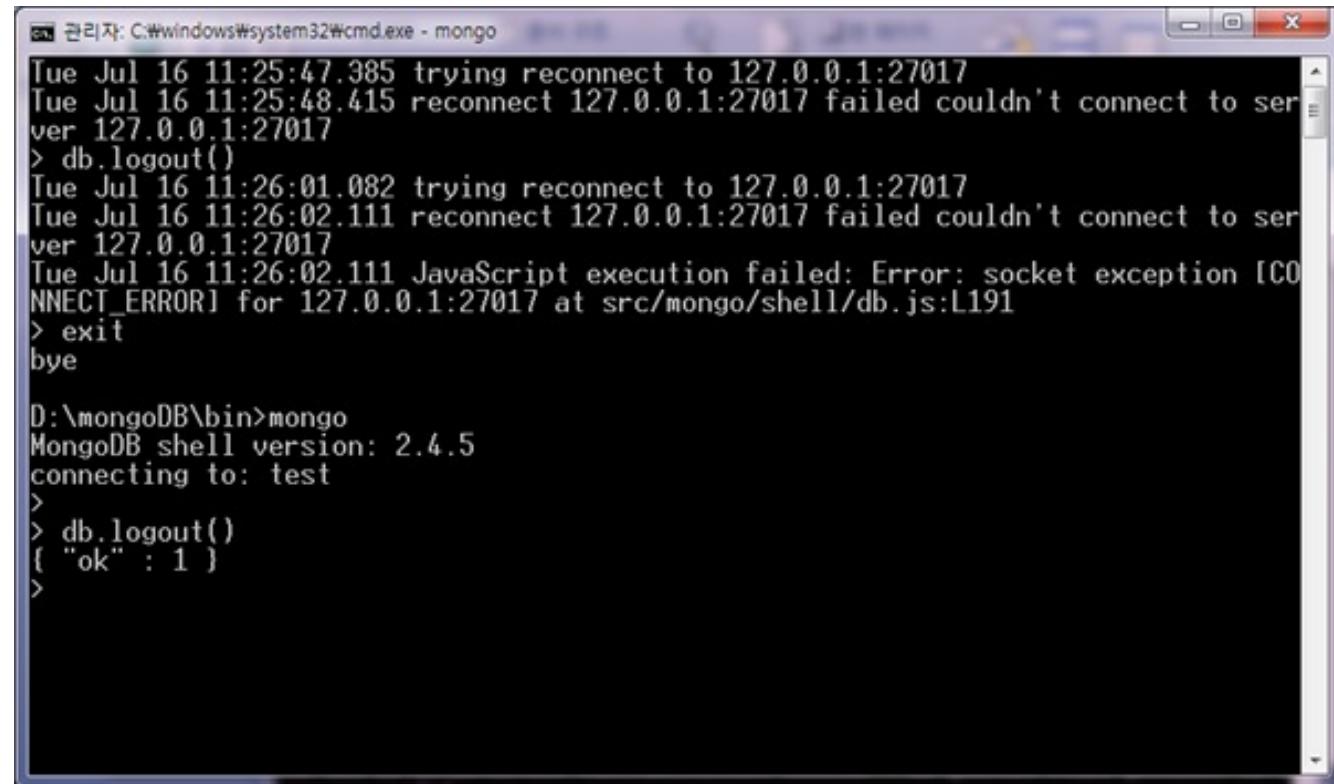


```
관리자: C:\Windows\system32\cmd.exe - mongo
{
    "fileSize" : 0,
    "nsSizeMB" : 0,
    "dataFileVersion" : {
        ...
    },
    "ok" : 1
}
> use admin
switched to db admin
> db.shutdownServer()
Tue Jul 16 11:22:59.217 DBCClientCursor::init call() failed
server should be down...
Tue Jul 16 11:22:59.217 trying reconnect to 127.0.0.1:27017
Tue Jul 16 11:23:00.246 reconnect 127.0.0.1:27017 failed couldn't connect to server 127.0.0.1:27017
>
Tue Jul 16 11:23:24.114 trying reconnect to 127.0.0.1:27017
Tue Jul 16 11:23:25.144 reconnect 127.0.0.1:27017 failed couldn't connect to server 127.0.0.1:27017
>
>
Tue Jul 16 11:23:28.233 trying reconnect to 127.0.0.1:27017
Tue Jul 16 11:23:29.262 reconnect 127.0.0.1:27017 failed couldn't connect to server 127.0.0.1:27017
>
```

# MongoDB에서의 데이터 처리

- **시작과 종료**

- 접속된 클라이언트 프로그램 Logout 방법
  - 단지 클라이언트의 접속만 해제
  - MongoDB 서버가 shutdown 되는 것이 아님.



The screenshot shows a Windows Command Prompt window titled "관리자: C:\windows\system32\cmd.exe - mongo". It displays a MongoDB shell session. The user attempts to log out from the database, but the server remains active. The session ends with the user exiting the shell.

```
Tue Jul 16 11:25:47.385 trying reconnect to 127.0.0.1:27017
Tue Jul 16 11:25:48.415 reconnect 127.0.0.1:27017 failed couldn't connect to server 127.0.0.1:27017
> db.logout()
Tue Jul 16 11:26:01.082 trying reconnect to 127.0.0.1:27017
Tue Jul 16 11:26:02.111 reconnect 127.0.0.1:27017 failed couldn't connect to server 127.0.0.1:27017
Tue Jul 16 11:26:02.111 JavaScript execution failed: Error: socket exception [CONNECT_ERROR] for 127.0.0.1:27017 at src/mongo/shell/db.js:L191
> exit
bye

D:\mongoDB\bin>mongo
MongoDB shell version: 2.4.5
connecting to: test
>
> db.logout()
{ "ok" : 1 }
>
```

# MongoDB에서의 데이터 처리

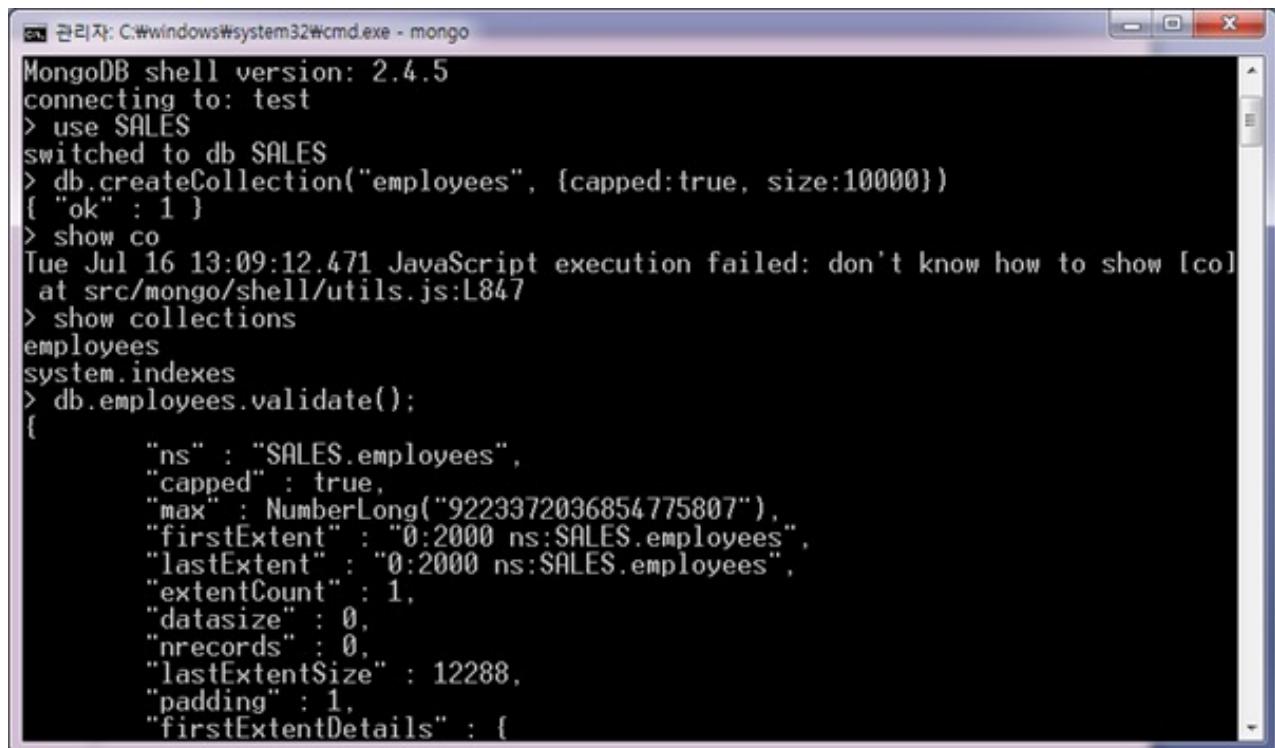
- **Collection 데이터 관리**

- 정형화된 데이터 구성요소(컬럼명, 데이터 타입과 길이, 제약 조건)가 결정되지 않아도 됨.
  - Collection 생성 관리
    - Capped Collection : 최초 제안된 크기로 생성된 공간(익스텐드) 내에서만 데이터 저장 가능하고, 만약 최초의 공간이 모두 사용되면 다시 처음으로 돌아가서 기존공간을 재사용하는 타입의 콜렉션. 기업에서는 로그 데이터 저장 등 일정한 기간 내에서만 저장, 관리가 필요가 있는 데이터에 적용.
    - Non Capped Collection : 관계형 데이터베이스의 테이블처럼 디스크 공간이 허용하는 범위안에서 데이터를 저장 할 수 있는 타입
  - collection 의 생성
    - db.createCollection()

# MongoDB에서의 데이터 처리

- **Collection 데이터 관리**

- collection의 현재 상태 및 정보
  - db.COLLECTION명.validate( )
- collection의 삭제
  - db.COLLECTION명.drop( );



```
관리자: C:\Windows\system32\cmd.exe - mongo
MongoDB shell version: 2.4.5
connecting to: test
> use SALES
switched to db SALES
> db.createCollection("employees", {capped:true, size:10000})
{ "ok" : 1 }
> show co
Tue Jul 16 13:09:12.471 JavaScript execution failed: don't know how to show [co]
at src/mongo/shell/utils.js:L847
> show collections
employees
system.indexes
> db.employees.validate();
{
    "ns" : "SALES.employees",
    "capped" : true,
    "max" : NumberLong("9223372036854775807"),
    "firstExtent" : "0:2000 ns:SALES.employees",
    "lastExtent" : "0:2000 ns:SALES.employees",
    "extentCount" : 1,
    "datasize" : 0,
    "nrecords" : 0,
    "lastExtentSize" : 12288,
    "padding" : 1,
    "firstExtentDetails" : {
```

# MongoDB에서의 데이터 처리

- 데이터의 삽입, 수정, 삭제
  - insert를 이용한 삽입

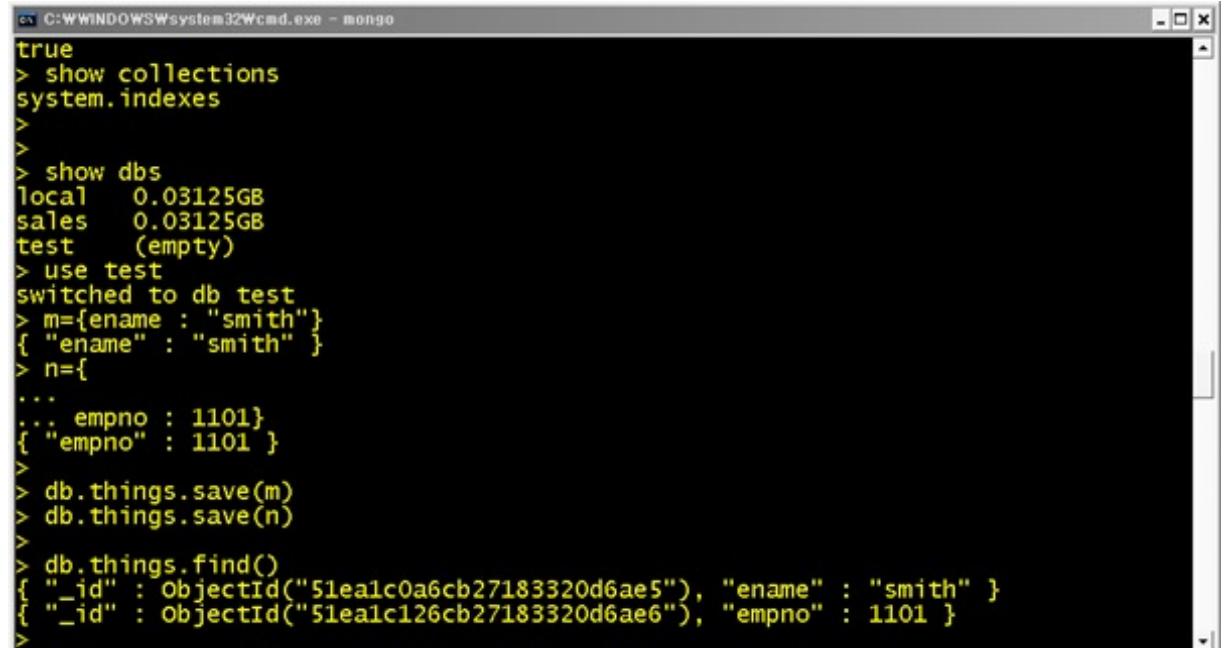
```
use test //Test DB 생성하고 이동
m={ ename : "smith" } // JSON 타입으로 데이터 표현
n={empno : 1101}
db.things.save(m) // 데이터를 저장할때에는 save 메소드 이용
db.things.save(n)
db.things.find( ) //저장된 데이터를 검색할때는 find 메소드 실행
db.things.insert({ empno : 1102, ename : "king" }) // 데이터를
입력할 때 insert 메소드 사용 가능함.
db.things.find( ) //저장된 데이터를 검색할때는 find 메소드 실행
for(var n = 1103; n <=1120; n++) db.things.save({n : n , m : "test" })
db.things.find( ) // for문을 통해 증가된 값을 n필드에 적용하여
데이터를 저장함.
it // 출력 결과가 20개를 초과할 때 쓰는 메소드, 다음 화면으로 이동
```

# 4

# mongoDB 기본적인 쉘

# MongoDB의 기본적인 웰

- 기본 데이터 표현
  - 데이터 표현
    - m = {ename: "smith"}
    - n = {empno : 1101}
  - things collection 생성 및 데이터의 저장
    - db.things.save(m)
    - db.things.save(n)
  - 저장된 데이터를 검색
    - db.things.find()



```
C:\WINDOWS\system32\cmd.exe - mongo
true
> show collections
system.indexes
>
>
> show dbs
local   0.03125GB
sales   0.03125GB
test    (empty)
> use test
switched to db test
> m={ename : "smith"}
{ "ename" : "smith" }
> n={
...
... empno : 1101}
{ "empno" : 1101 }
>
> db.things.save(m)
> db.things.save(n)
>
> db.things.find()
{ "_id" : ObjectId("51ealc0a6cb27183320d6ae5"), "ename" : "smith" }
{ "_id" : ObjectId("51ealc126cb27183320d6ae6"), "empno" : 1101 }
>
```

# MongoDB의 기본적인 웰

- 기본 데이터 표현

- 데이터 생성
    - 데이터를 입력할 때 insert 메소드 사용
    - db.things.insert({empno : 1102, ename : "king"})
  - 저장된 데이터를 검색
    - db.things.find()

```
> db.things.insert({empno : 1102, ename : "king"})
>
> db.things.find()
[{"_id": ObjectId("51ea1c0a6cb27183320d6ae5"), "ename": "smith" },
 {"_id": ObjectId("51ea1c126cb27183320d6ae6"), "empno": 1101 },
 {"_id": ObjectId("51ea29086cb27183320d6ae7"), "empno": 1102, "ename": "king"}
]>
```

- 조회
    - find() : 컬렉션 내의 모든 문서를 반환
    - findOne() : 컬렉션 내의 한 문서를 반환

```
> db.things.findOne()
{ "_id": ObjectId("51ea1c0a6cb27183320d6ae5"), "ename": "smith" }
>
```

# MongoDB의 기본적인 쉘

- 기본 데이터 표현

- 데이터 갱신 1
  - 두 개의 매개변수가 필요함
  - 갱신하려는 문서를 찾는 조건 / 새로운 문서
  - n 데이터를 수정하고, ebirth 키를 추가
    - n.ebirth="1999.10.1"
  - empno이 1101인 m의 새로운 문서를 치환하는 update 수행
    - db.things.update({empno : 1101}, n)
    - db.things.find() // 조회

```
> n.ebirth = "1999.10.1"
1999.10.1
> db.things.update(
... {empno : "1101"}, n)
Sat Jul 20 15:40:05.765 Javascript execution failed: SyntaxError: Unexpected token en }
> db.things.update( {empno : 1101}, n)
Sat Jul 20 15:40:24.609 Javascript execution failed: SyntaxError: Unexpected token en }
> db.things.update( {empno : 1101}, n)
> db.things.find()
{ "_id" : ObjectId("51ea29086cb27183320d6ae7"), "empno" : 1102, "ename" : "king" }
{ "_id" : ObjectId("51ealc0a6cb27183320d6ae5"), "ename" : "smith", "ebirth" : [ ] }
{ "_id" : ObjectId("51ealc126cb27183320d6ae6"), "empno" : 1101, "ebirth" : "1999.10.1" }
>
```

# MongoDB의 기본적인 웰

- 기본 데이터 표현

- 데이터 간접 2
    - dept 키를 추가
      - db.things.update({empno:1102},{\$set: {dept : "human"}})
      - db.things.find( ) //조회

```
> db.things.find()
[{"_id": ObjectId("51ea1c0a6cb27183320d6ae5"), "ename": "smith", "ebirth": []},
 {"_id": ObjectId("51ea1c126cb27183320d6ae6"), "empno": 1101, "ebirth": "1999.10.1"},
 {"_id": ObjectId("51ea29086cb27183320d6ae7"), "dept": "human", "empno": 1102,
 "ename": "king"}]
```

- 데이터 삭제

- ename 0| smith 인 문서만 삭제
      - db.things.remove({ename : "smith"})
    - 조회
      - db.things.find( )

```
> db.things.remove({ename : "smith"})
> db.things.find()
[{"_id": ObjectId("51ea1c126cb27183320d6ae6"), "empno": 1101, "ebirth": "1999.10.1"},
 {"_id": ObjectId("51ea29086cb27183320d6ae7"), "dept": "human", "empno": 1102,
 "ename": "king"}]
```

# MongoDB의 기본적인 쉘

- 기본 데이터 표현
  - 모든 문서 삭제
    - db.things.remove({ })
  - 조회
    - db.things.find( )

```
> db.things.remove({  
... })  
> db.things.find()  
>
```

# MongoDB의 기본적인 쉘

- **기본 데이터 표현**
  - <잠깐!! save, insert, update문의 차이점>
    - MongoDB에서는 한 문서를 저장할 때 save, insert, update 메소드를 사용할 수가 있음
    - 문서를 저장하는 기능은 유사하나, 내부적으로 처리하는 방법이 다름

## 1) insert

콜렉션에 하나의 문서를 최초 저장할 때 일반적으로 사용되는 메소드

## 2) update

하나의 문서에서 특정 필드만을 수정하는 경우 사용되는 메소드

여기서, 하나의 문서가 여러 개의 필드로 구성되어 있더라도 해당 필드만 수정하기 때문에 빠른 시간 내에 효율적으로 데이터를 변경할 수 있음. 빅 데이터의 빠른 수정이 요구되는 경우 가장 적합한 방법임.

## 3) save

하나의 문서에서 특정 필드만 변경하더라도 문서 단위로 데이터를 변경하는 방법

문서 단위로 데이터를 변경하는 경우에는 효율적이지만, 필드 단위로 변경하는 경우에는 update 메소드를 사용하는 것이 좋음

# 5

## **mongoDB에서의 데이터타입**

# MongoDB의 데이터 타입

- **JSON 과 BSON**
  - JSON 타입
    - Document(관계형 데이터베이스에서 ROW에 해당) 중심의 데이터 저장 기술로 구현
    - 반드시 {~~} 통해 표현
    - Java Script Object Notation
  - BSON 타입
    - 저장될때 바이너리 형태의 데이터로 변환 저장
    - Binary Serial Object Notation

# MongoDB의 데이터 타입

- **Data type 종류**
  - OBJECT\_ID 타입
    - 유일한 값
    - BSON Object ID는 12 Byte의 바이너리 값으로 구성
    - 하나의 Collection에 하나의 Document를 입력하면 반드시 유일한 값인 OBJECT ID가 부여됨.
    - 사용자가 별도로 직접 Object ID를 부여할 수 있음.
    - ObjectId ("5250bbef f178e7 7ff8 b71009") (타임스탬프/서버ID/프로세서ID/로컬 카운터)

```
> p={ eno : 1101, fname : "adam", lname : "kroll", job : "manager", salary : 10000, dept_name : "sales" }
{
  "eno" : 1101,
  "fname" : "adam",
  "lname" : "kroll",
  "job" : "manager",
  "salary" : 10000,
  "dept_name" : "sales"
}
> p
{
  "eno" : 1101,
  "fname" : "adam",
  "lname" : "kroll",
  "job" : "manager",
  "salary" : 10000,
  "dept_name" : "sales"
}
> db.emp.save(p)
```

# MongoDB의 데이터 타입

- 실습 (OBJECT\_ID 타입)

```
use test
P = { eno : 1101, fname: "adam", lname : "kroll", job : "manager", salary :
10000, dept_name : "sales" }
db.emp.save(p)
p
db.emp.findOne({_id : ObjectId("id값")})
db.emp.findOne({_id : new ObjectId("새로운 id값")})
db.emp.drop()
```

# MongoDB의 데이터 타입

- **Data type** 종류
  - JSON 타입
    - 문자, 숫자, 바이너리 데이터를 저장할 수 있는 타입

```
> x = { "_id" : ObjectId("4dcd3ebc9278000000005158"), "d" : ISODate("2013-08-20T14:22:46.777Z"), "b" : BinData(0,""), "c" : "aa", "n" : 3, "e" : [ ], "n2" : NumberLong(33) }
{
    "_id" : ObjectId("4dcd3ebc9278000000005158"),
    "d" : ISODate("2013-08-20T14:22:46.777Z"),
    "b" : BinData(0,""),
    "c" : "aa",
    "n" : 3,
    "e" : [ ],
    "n2" : NumberLong(33)
}
> db.datatype.save(x)
>
> db.datatype.find()
{ "_id" : ObjectId("4dcd3ebc9278000000005158"), "d" : ISODate("2013-08-20T14:22:46.777Z"), "b" : BinData(0,""), "c" : "aa", "n" : 3, "e" : [ ], "n2" : NumberLong(33) }
```

# MongoDB의 데이터 타입

- **Data type 종류**

- 배열 데이터 타입

- for(var n = 1103; n <= 1120; n++) db.things.save({empno:n; ename:"test", sal: 1000})

```
> for (var n = 1103; n <= 1120; n++) db.things.save({empno:n, ename:"test", sal: 1000})
> db.things.find()
{ "_id" : ObjectId("51ec389c69710dc7169a30ce"), "empno" : 1103, "ename" : "test"
, "sal" : 1000 }
{ "_id" : ObjectId("51ec389c69710dc7169a30cf"), "empno" : 1104, "ename" : "test"
, "sal" : 1000 }
{ "_id" : ObjectId("51ec389c69710dc7169a30d0"), "empno" : 1105, "ename" : "test"
, "sal" : 1000 }
{ "_id" : ObjectId("51ec389c69710dc7169a30d1"), "empno" : 1106, "ename" : "test"
, "sal" : 1000 }
{ "_id" : ObjectId("51ec389c69710dc7169a30d2"), "empno" : 1107, "ename" : "test"
, "sal" : 1000 }
{ "_id" : ObjectId("51ec389c69710dc7169a30d3"), "empno" : 1108, "ename" : "test"
, "sal" : 1000 }
{ "_id" : ObjectId("51ec389c69710dc7169a30d4"), "empno" : 1109, "ename" : "test"
```

- 배열 변수에 결과 저장
    - varcursor = db.things.find()
  - 배열 데이터를 반복문을 통해 출력
    - while(cursor.hasNext()) printjson(cursor.next())

# MongoDB의 데이터 타입

- **Data type 종류**

- 배열 변수에 결과 저장
  - var cursur = db.things.find( )
- 배열 데이터를 반복문을 통해 출력
  - while(cursur.hasNext( )) printjson(cursur.next( ))

```
> sal = 1000 /  
> var cursur = db.things.find()  
> while(cursur.hasNext()) printjson(cursur.next())  
>  
  "_id" : ObjectId("51ec389c69710dc7169a30ce"),  
  "empno" : 1103,  
  "ename" : "test",  
  "sa1" : 1000  
>  
  "_id" : ObjectId("51ec389c69710dc7169a30cf"),  
  "empno" : 1104,  
  "ename" : "test",  
  "sa1" : 1000  
>  
  "_id" : ObjectId("51ec389c69710dc7169a30d0"),  
  "empno" : 1105,  
  "ename" : "test",  
  "sa1" : 1000  
>
```

# MongoDB의 데이터 타입

- **Data type 종류**

- 배열에 데이터를 저장할 수 있음.
  - var cursur = db.things.find()
- 17번째 배열에 저장된 데이터만 출력
  - printjson(cursur[17])
- 17번째 배열에 저장된 데이터만 출력
  - var arr = db.things.find( ).toArray( )
  - arr[17]

```
>
> var cursur = db.things.find()
> printjson(cursur[17])
{
    "_id" : ObjectId("51ec389c69710dc7169a30df"),
    "empno" : 1120,
    "ename" : "test",
    "sa1" : 1000
}
>
> var arr = db.things.find().toArray()
> arr[17]
{
    "_id" : ObjectId("51ec389c69710dc7169a30df"),
    "empno" : 1120,
    "ename" : "test",
    "sa1" : 1000
}
```

# MongoDB의 데이터 타입

- **Date 타입**

- Date 함수를 통해 현재 날짜 정보를 DATE 타입으로 출력합니다.
  - x = new Date( )
- 수립된 DATE 값을 문자형으로 출력합니다.
  - x.toString( )
- ISODate 함수를 통해 현재 날짜정보를 DATE 타입으로 출력합니다.
  - d=ISODate( )
- 수집된 DATE 값 중에 해당월을 출력합니다.
  - d.getMonth( )

```
> x= new Date()
Mon Jul 22 05:02:28.671 JavaScript execution failed: ReferenceError: Data is not
defined
> x= new Date()
ISODate("2013-07-21T20:02:37.781Z")
>
> x.toString()
Mon Jul 22 2013 05:02:37 GMT+0900 <한국시간>
>
> d=ISODate()
ISODate("2013-07-21T20:03:08.406Z")
>
> d.getMonth()
6
>
```

# MongoDB의 데이터 타입

- **Timestamp타입**
  - 싱글노드에서 MongoDB의 Timestamp 타입은 64bit 값으로 저장되며, 2개의 필드로 구성된 값을 리턴함.
  - Timestamp 함수를 통해 현재 시간정보를 저장.
    - db.foo.insert({ x : 1, y : new Timestamp() })
  - 문장이 실행된 시점의 Timestamp 정보
    - db.foo.find( )

```
> db.foo.insert({ x : 1, y : new Timestamp() })
> db.foo.find()
{ "_id" : ObjectId("51ec40c569710dc7169a30e0"), "x" : 1, "y" : Timestamp(0, 0) }

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

# MongoDB의 데이터 타입

- **Timestamp타입**

- Timestamp 필드 정보를 기준으로 실행
    - db.foo.insert({y : new Timestamp( ), x : 3})
    - db.foo.find({}, {\_id : 0})

```
> db.foo.insert({ y : new Timestamp(), x : 3 })
>
> db.foo.find({})._id
Mon Jul 22 05:18:37.046 JavaScript execution failed: SyntaxError: Unexpected token
en {
> db.foo.find({}._id)
{ "y" : Timestamp(1374437834, 1), "x" : 3 }
>
>
```

# MongoDB의 데이터 타입

- **Sequence Number 타입**

- 관계형 데이터베이스에는 제품에 따라 다르지만, 연속적인 값을 생성하기 위한 기능들이 제공
  - MongoDB에서는 이러한 기능이 제공되지 않지만, 다음과 같은 함수를 이용하여 직접 생성할 수 있음.

```
> function seq_no(name) {var ret = db.seq_no.findAndModify({query:{_id:name}, update:{$inc:{next:1}}, "new" : true, upsert:true}); return ret.next; }
> db.order_no.insert({_id:seq_no("order_no"), name : "jimmy"})
> db.order_no.insert({_id:seq_no("order_no"), name: "Chad"})
>
> db.order_no.find()
{ "_id" : 1, "name" : "jimmy" }
{ "_id" : 2, "name" : "Chad" }
```

# MongoDB의 연산자

- **비교연산자 & Boolean 연산자**

- 관계형 데이터베이스에는 제품에 따라 다르지만, 연속적인 값을 생성하기 위한 기능들이 제공
- MongoDB에서는 이러한 기능이 제공되지 않지만, 다음과 같은 함수를 이용하여 직접 생성할 수 있음.

```
> db.employees.find({empno:7900}, {ename:""}).forEach(printjson)
{ "_id" : ObjectId("525eabb6a3c1f6b9931b3f13"), "ename" : "JAMES" }
>
> db.employees.ensureIndex({ename:1})
>
> db.employees.find({}, {_id:0, empno:1, ename:1}).min({ename:"ALLEN"}).max({ename:"SCOTT"})
{ "empno" : 7499, "ename" : "ALLEN" }
{ "empno" : 7698, "ename" : "BLAKE" }
{ "empno" : 7782, "ename" : "CLARK" }
{ "empno" : 7934, "ename" : "CLERK" }
{ "empno" : 7902, "ename" : "FORD" }
{ "empno" : 7900, "ename" : "JAMES" }
{ "empno" : 7566, "ename" : "JONES" }
{ "empno" : 7654, "ename" : "MARTIN" }
{ "empno" : 7839, "ename" : "PRESIDENT" }
>
```

# MongoDB의 연산자

- 비교연산자 & Boolean 연산자

```
> db.employees.ensureIndex({deptno:1})
>
> db.employees.find({$min : {deptno:20}, $max:{deptno:30}, $query:{}}, {_id:0, empno:1, ename:1, hiredate:1})
{ "empno" : 7369, "ename" : "SMITH", "hiredate" : "17-12-1980" }
{ "empno" : 7566, "ename" : "JONES", "hiredate" : "02-04-1981" }
{ "empno" : 7788, "ename" : "SCOTT", "hiredate" : "13-06-1987" }
{ "empno" : 7876, "ename" : "ADAMS", "hiredate" : "13-06-1987" }
{ "empno" : 7902, "ename" : "FORD", "hiredate" : "03-12-1981" }
>
> db.employees.find({empno:{$gt:7500, $lte:7600}}, {_id:0, empno:1, ename:1})
{ "empno" : 7521, "ename" : "WARD" }
{ "empno" : 7566, "ename" : "JONES" }
>
> db.employees.count()
14
>
> db.employees.find({empno:{'$gt':7900}}).count()
2
>
> db.employees.distinct("deptno")
[ 10, 20, 30 ]
```

# MongoDB의 연산자

- **비교연산자 & Boolean 연산자**
  - \$lt(미만)
    - 두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 작으면 true, 크거나 같으면 false를 리턴
  - \$lte(이하)
    - 두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 작거나 같으면 true, 크면 false를 리턴
  - \$gt(초과)
    - 두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 크면 true, 작으면 false를 리턴
  - \$gte(이상)
    - 두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 크거나 같으면 true, 작으면 false를 리턴
  - \$ne(같지 않음)
    - 두 개의 값을 비교하여 같지 않으면 true, 같으면 false를 리턴
  - \$eq(같음)
    - 두 개의 값을 비교하여 동일하면 true, 동일하지 않으면 false를 리턴
  - \$and(쿼리가 다 만족)
    - 여러 개의 조건이 모두 true인 조건 검색
  - \$or(쿼리 중 하나가 만족)
    - 여러 개의 조건 중에서 하나라도 만족되는 조건을 검색
  - \$not
    - 검색 조건이 아닌 조건을 검색

# MongoDB의 연산자

- **산술 연산자**
  - \$add : 두 개의 값을 합산한 결과를 리턴
  - \$devide : 두 개의 값을 나눈 결과를 리턴
  - \$mod : 첫번째 값을 두번째 값으로 나눈 후 나머지 값을 리턴
  - \$multiply : 첫번째 값과 두 번째 값을 곱한 결과를 리턴
  - \$subtract : 첫번째 값에서 두 번째 값을 뺀 결과를 리턴
- **문자 연산자**
  - \$toUpperCase : 해당 문자열의 값을 소문자로 변환
  - \$toLowerCase : 해당 문자열의 값을 대문자로 변환

# 6

# mongoDB와 SQL 비교

# SQL 문법과 MongoDB 문법 비교 분석

- **개요**
  - 대부분의 개발자들은 기존의 관계형 데이터베이스의 SQL문법에 익숙해 있을 것이다. 문법을 비교하여 본다.
- **사전 작업**

```
use test
db.members.insert({_id: ObjectId("2013010100000000001"), mem_no : "T20130001", age : 49, type : "ACE"})
```

# SQL 문법과 MongoDB 문법 비교 분석

- **Create & Alter 문**

SQL 문장	MongoDB 스키마 문장
CREATE TABLE members( mem_no varchar(30), age number, type char(1), PRIMARY KEY (mem_no) );	db.members.insert({_id: ObjectId("20130101000000000001"), mem_no : "T20130001", age : 49, type : "ACE"}) 또는 Db.createCollection("members")
ALTER TABLE members ADD regist_date DATE;	
ALTER TABLE members DROP COLUMN regist_date ;	
CREATE INDEX i_members_type ON members(type);	db.members.ensureIndex({type : 1})
CREATE INDEX I_members_type_no ON member(type, mem_no DESC);	db.memebrs.ensureIndex({type:1,mem_no:-1})
DROP TABLE members;	db.members.drop()

# SQL 문법과 MongoDB 문법 비교 분석

- **Insert 문**

SQL 문장	MongoDB 스키마 문장
INSERT INTO members ( Mem_no, Age, Type) VALUES ("T20130102",35, "GOLD")	db.members.insert({mem_no : "T20130001", age : 35, type : "GOLD"})

- **Update Records 문**

SQL 문장	MongoDB 스키마 문장
UPDATE members SET type = "GOLD" WHERE age > 45;	db.members.update( {age : {\$gt : 45 }}, {\$set : { type : "GOLD"}}, {multi : true } )
UPDATE members SET age = age + 3 WHERE type = "ACE";	db.members.update( { type : "ACE"}, {\$inc : {age : 3 }}, {multi : true } )

# SQL 문법과 MongoDB 문법 비교 분석

- **Select문**

SQL 문장	MongoDB 스키마 문장
SELECT * FROM members;	db.members.find()
SELECT rowid, mem_no, type FROM memebrs,	db.memebers.find( { }, {mem_no : 1, type : 1})
SELECT mem_no, type FROM memebrs,	db.memebers.find( { }, {mem_no : 1, type : 1, _id : 0 })
SELECT * FROM memebrs WHERE type = "ACE";	db.members.find( {type : "ACE"}, )
SELECT mem_no, type FROM memebrs WHERE type = "ACE";	db.members.find( {type : "ACE"}, {mem_no : 1, type : 1, _id : 0 } )
SELECT * FROM memebrs WHERE type != "ACE";	db.members.find( {type : { \$ne:"ACE"}}, )

# SQL 문법과 MongoDB 문법 비교 분석

- **Select문**

SQL 문장	MongoDB 스키마 문장
SELECT * FROM members WHERE type = "ACE" AND age =49;	db.members.find( {type : "ACE", age : 49} )
SELECT * FROM members WHERE type = "ACE" OR age =49;	db.members.find( {\$or : [ { type : "ACE"} , {age : 49} ] } )
SELECT * FROM members WHERE age > 45;	db.memebers.find( {age : { \$gt : 45}} )
SELECT * FROM members WHERE age < 55;	db.memebers.find( {age : { \$lt : 45}} )
SELECT * FROM members WHERE age > 45 AND age <=55;	db.memebers.find( {age : { \$gt : 45, \$lte : 55}} )

# SQL 문법과 MongoDB 문법 비교 분석

- **Select문**

SQL 문장	MongoDB 스키마 문장
SELECT * FROM members WHERE mem_no like "%2013%";	db.members.find( {mem_no: /2013/} )
SELECT * FROM members WHERE mem_no like "T%";	db.members.find( {mem_no: /^T/} )
SELECT * FROM members WHERE type = "ACE" ORDER BY mem_no ASC;	db.memebers.find( { type : "ACE"}).sort({mem_no : 1})
SELECT * FROM members WHERE type = "ACE" ORDER BY mem_no DES;	db.memebers.find( { type : "ACE"}).sort({mem_no : -1})
SELECT COUNT (*) FROM members;	db.memebers.count() db.memebers.fond().count()

# SQL 문법과 MongoDB 문법 비교 분석

- **Select문**

SQL 문장	MongoDB 스키마 문장
SELECT COUNT(mem_no) FROM members;	db.members.count( { mem_no : {\$exists : true}})
SELECT COUNT(*) FROM members WHERE age > 45;	db.members.count( { age : {\$gt : 45}} )
SELECT DISTINCT type FROM members;	db.memebers.distinct("type")
SELECT * FROM members WHERE rownum = 1;	db.memebers.findOne( )
EXPLAIN PLAN SELECT * FROM members WHERE type = "ACE";	db.memebers.find( { type : "ACE"} ).explain( )

# SQL 문법과 MongoDB 문법 비교 분석

- **DELETE문**

SQL 문장	MongoDB 스키마 문장
DELETE FROM members WHERE type = "ACE";	db.members.remove({type : "ACE"})
DELETE FROM members;	db.members.remove( )

# MongoDB의 find( ) 명령어

- **Find 명령어의 구조**

- emp 컬렉션에서 score가 60점이 넘는 학생 5명을 가져오는 질의문

```
db.emp.find(  
    {score : {$gt:60}},  
    {name:1, score:1}  
).limit(5)          // 컬렉션 이름  
                    // 질의 조건  
                    // 프로젝션  
                    // 수식어
```

- record 컬렉션에서 user\_id 가 42보다 작은 사용자 중 \_id 필드는 가져오지 않고, name과 email 필드만 가져오는 질의문

```
db.record.find({user_id : {$lt:42}},{_id:0, name:1, email:1})
```

# MongoDB의 find( ) 명령어

- **프로젝션**
  - 결과 문서를 반환할 필드를 지정
  - 종류
    - 배열 필드 : \$elemMatch, \$slice
    - 집합 프레임 워크 : \$project

**\$elemMatch** : 하나의 문서 내 배열 필드에서 \$elemMatch 조건과 일치하는 첫번째 배열요소만 반환

```
db.school.find({zipcode : 63109},{student : {$elemMatch : {school : 102, age : {$gt : 10}}}})
```

**\$slice** : 배열 필드에서 반환하는 요소의 개수를 지정

```
db.school.find({zipcode : 63109},{student : {$slice : -5}})
```

# 7

## 사용자 관리

# MongoDB의 사용자 관리

- 사용자 생성
  - MongoDB는 기본적인 스키마를 제공하지 않는다.
  - 사용자 계정은 데이터베이스에 접속한 사용자가 인가된 사용자인지 아니면 비인가된 사용자인지에 대한 판단은 인증 기준으로만 사용될 뿐 객체의 이름과는 전혀 상관이 없다.
  - 사용법 : db.addUser (ID, password)
    - use admin
    - db.addUser("system", "manager")

```
> use admin
switched to db admin
> db.addUser("system", "manager")
{
    "user" : "system",
    "readOnly" : false,
    "pwd" : "415d65ef1a3a4116eaf0a6fae3e49278",
    "_id" : ObjectId("524395f8e7f7edac7d988dfb")
}
```

# MongoDB의 사용자 관리

- 사용자의 삭제

- 사용법 : db.removeUser(ID)
    - db.removeUser("system")

```
> db.removeUser("system")
>
>
```

- 암호 변경

```
> db.addUser("system", "m")
{
  "_id" : ObjectId("5243dbcbe7f7edac7d988dfc"),
  "user" : "system",
  "readOnly" : false,
  "pwd" : "8a8028d361e4d113238a8dd1f1fe10a6"
}
```

# MongoDB의 사용자 관리

- 사용자의 인증 방법
  - MongoDB 인스턴스가 시작된 후 DB에 접속할 때 인증 확인
    - db.auth(ID, password)

```
> db.auth("system", "manager")
1
>
```

1: 인증 완료, 0 : 인증 거부

# MongoDB의 사용자 관리

- 사용자의 인증 방법

- MongoDB 인스턴스를 시작할 때 인증 여부를 확인
  - 반드시 MongoDB 구동시 —auth 파라메터를 이용하여 시작

```
D:\>mongod --dbpath d:\mongodb\test --auth
Thu Sep 26 17:08:12.234
Thu Sep 26 17:08:12.234 warning: 32-bit servers don't have journaling enabled by
default. Please use --journal if you want durability.
Thu Sep 26 17:08:12.234
Thu Sep 26 17:08:12.421 [initandlisten] MongoDB starting : pid=4132 port=27017 c
bpath=d:\mongodb\test 32-bit host=mainpc
Thu Sep 26 17:08:12.421 [initandlisten]
Thu Sep 26 17:08:12.421 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Thu Sep 26 17:08:12.437 [initandlisten] **          32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Thu Sep 26 17:08:12.437 [initandlisten] **          Note that journaling defaults t
o off for 32 bit and is currently off.
Thu Sep 26 17:08:12.437 [initandlisten] **          See https://dochub.mongodb.org/c
```

# MongoDB의 사용자 관리

- 사용자의 인증 방법

- 클라이언트 접속 후 인증이 확인되지 않았기 때문에 에러 발생

```
> use test
switched to db test
>
> show collections
Thu Sep 26 17:12:58.093 JavaScript execution failed: error: {
    "$err" : "not authorized for query on test.system.namespaces",
    "code" : 16550
} at src/mongo/shell/query.js:L128
```

- 인증이 완료 되었기 때문에 정상적인 데이터 조회 가능

```
> db.auth("sales", "sales123")
1
>
> show collections
system.indexes
system.users
things
>
```

# MongoDB의 사용자 관리

- 사용자의 인증 방법

- 명령어로 직접 접속 시도했으나 잘못된 암호 입력으로 접속 실패

```
C:\Documents and Settings\MyHome>mongo --username sales --password sales456 test  
MongoDB shell version: 2.4.5  
connecting to: test  
Thu Sep 26 17:17:21.953 JavaScript execution failed: Error: 18 { code: 18, ok: 0  
.0, errmsg: "auth fails" } at src/mongo/shell/db.js:L228  
exception: login failed
```

- 명령어로 직접 접속 시도

```
C:\Documents and Settings\MyHome>mongo --username sales --password sales123 test  
MongoDB shell version: 2.4.5  
connecting to: test  
> show collections  
system.indexes  
system.users  
things  
>  
> db.removeUser("sales")  
> show users
```

# 8

## 인덱싱 (indexing)

# MongoDB의 인덱싱

- 인덱스 생성과 관리
  - 인덱스의 사용은 빠른 데이터 검색을 위해서이다. 관계형 데이터 베이스에서 사용하고 있는 인덱스와 유사한 기능을 하는 것이다. NoSQL 관련된 제품 중에서 가장 대표적인 제품 중에 하나가 MongoDB이다.

**mongoDB의 인덱스의 특징**은 다음과 같다.

- (1) 인덱스명은 대소문자를 구분한다.
- (2) 도큐먼트를 변경할 시에 해당 인덱스 키만 변경되지만, 도큐먼트 전체를 변경해야 하는 경우 변경된 도큐먼트 크기가 기존의 EXTENT 공간 크기보다 큰 경우에는 더 큰 EXTENT 공간으로 마이그레이션이 될 수 있어서 처음 설계시 충분한 EXTENT 공간을 확보해야 한다.
- (3) 불필요한 데이터 검색을 감소시키기 위해 `sort()`와 `limit()`을 같이 사용하면 성능 향상에 좋다.

# MongoDB의 인덱싱

- **인덱스 생성**
  - 방식 : db.컬렉션명.ensureIndex 메소드
    - db.emp.ensureIndex({eno:1})
- **인덱스 출력**
  - 현재 컬렉션에 생성되어 있는 인덱스와 종류와 개수를 확인
  - 방식 : db.컬렉션명.getIndexes()
    - db.emp.getIndexes()

```
> db.emp.ensureIndex({eno:1})
>
> db.emp.getIndexes()
[
    {
        "v": 1,
        "key": {"_id": 1},
        "ns": "test.emp",
        "name": "_id_"
    },
    {
        "v": 1,
        "key": {"eno": 1},
        "ns": "test.emp",
        "name": "eno_1"
    }
]
```

# MongoDB의 인덱싱

- **인덱스 삭제**

- 방식 : db.컬렉션명.dropIndex
  - db.emp.dropIndex({eno:1})
- 방식 : db.runCommand
  - db.runCommand({dropIndexes: 'emp', index:{eno:1}})
- eno 인덱스가 삭제되어 출력되지 않는다.

```
> db.emp.dropIndex({eno:1})
{ "nIndexesWas" : 3, "ok" : 1 }
> db.emp.getIndexes()
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "ns" : "test.emp",
        "name" : "_id_"
    },
    {
        "v" : 1,
        "key" : {
            "deptno" : 1
        },
        "ns" : "test.emp",
        "name" : "deptno_1"
    }
]
```

# MongoDB의 인덱싱

- **인덱스 삭제**
  - deptno 인덱스가 삭제되어서 출력되지 않는다.

```
> db.runCommand({dropIndexes:'emp', index:{deptno:1}})  
{ "nIndexesWas" : 2, "ok" : 1 }  
> db.emp.getIndexes()  
[  
    {  
        "v" : 1,  
        "key" : {  
            "_id" : 1  
        },  
        "ns" : "test.emp",  
        "name" : "_id_"  
    }  
]  
>
```

- **인덱스의 재구성**
  - MongoDB에서 제공하는 Balance\*Tree 인덱스를 사용하면 데이터의 빠른 검색이 가능하지만 입출력이 다수가 발생하면 성능 저하가 발생할 수 있다. 이것은 Balance\*Tree 인덱스의 일반적은 사용상의 문제점이다. 이 경우의 해결책은 인덱스 재구성을 통하여 진행할 수 있다.
  - 방식 : db.컬렉션명.reIndex()
    - db.emp.reIndex()
  - 방식 : db.runCommand
    - db.runCommand({reIndex: 'emp'})

# MongoDB의 인덱싱

- 인덱스의 재구성

- 기존의 인덱스를 삭제하고 재구성한다.

```
> db.emp.ensureIndex({eno:1})
> db.emp.ensureIndex({deptno:1})
> db.emp.reIndex()
{
    "nIndexesWas" : 3,
    "msg" : "indexes dropped for collection",
    "nIndexes" : 3,
    "indexes" : [
        {
            "key" : { "_id" : 1 },
            "ns" : "test.emp",
            "name" : "_id_"
        },
        {
            "key" : { "eno" : 1 },
            "ns" : "test.emp",
            "name" : "eno_1"
        },
        {
            "key" : { "deptno" : 1 },
            "ns" : "test.emp",
            "name" : "deptno_1"
        }
    ],
    "ok" : 1
}
```

# MongoDB의 인덱스의 종류

- **Single-key 인덱스와 Compound key 인덱스**
  - Single-key 인덱스 : 하나의 필드로 생성
    - db.emp.ensureIndex({ename:1})
    - 싱글키 인덱스 (ASC : 1 / DESC : -1)
  - Compound key 인덱스 : 여러 개의 필드로 생성
    - db.emp.ensureIndex({ename:1, comm : -1})
    - 복합키 인덱스
- **싱글 키 인덱스와 복합키 인덱스**

```
> db.emp.ensureIndex({ename:1})
> db.emp.ensureIndex({ename:1, comm : -1})
>
```

# MongoDB의 인덱스의 종류

- 싱글키 인덱스(ename)와 복합키 인덱스(ename/comm)의 삽입 확인

```
> db.emp.getIndexes()
[  
    {  
        "v" : 1,  
        "key" : {  
            "_id" : 1  
        },  
        "ns" : "test.emp",  
        "name" : "_id_"  
    },  
    {  
        "v" : 1,  
        "key" : {  
            "eno" : 1  
        },  
        "ns" : "test.emp",  
        "name" : "eno_1"  
    },  
    {  
        "v" : 1,  
        "key" : {  
            "deptno" : 1  
        },  
        "ns" : "test.emp",  
        "name" : "deptno_1"  
    },  
    {  
        "v" : 1,  
        "key" : {  
            "ename" : 1  
        },  
        "ns" : "test.emp",  
        "name" : "ename_1"  
    },  
    {  
        "v" : 1,  
        "key" : {  
            "ename" : 1,  
            "comm" : -1  
        },  
        "ns" : "test.emp",  
        "name" : "ename_1_comm_-1"  
    }  
]
```

# MongoDB의 인덱스의 종류

- **Non-Unique 인덱스와 Unique 인덱스**
  - Unique 인덱스 ; 유일한 값들이 저장되어 있는 필드에 생성
    - db.emp.ensureIndex({eno : 1}, { unique : true})
  - Non-Unique 인덱스 : 중복된 값이 저장되어 있는 필드에 생성
    - db.emp.ensureIndex({ename:1})
- **Unique 인덱스(eno)와 Non-Unique 인덱스(ename)**

```
> db.emp.ensureIndex({eno:1},{unique:true})
> db.emp.ensureIndex({ename:1})
>
```

# MongoDB의 인덱스의 종류

- Unique 인덱스(eno)와 Non-Unique 인덱스(ename) 확인

```
        "name" : "ename_1_composite"
    },
    {
        "v" : 1,
        "key" : {
            "eno" : 1
        },
        "unique" : true,
        "ns" : "test.emp",
        "name" : "eno_1"
    },
    {
        "v" : 1,
        "key" : {
            "ename" : 1
        },
        "ns" : "test.emp",
        "name" : "ename_1"
    }
]
```

# MongoDB의 인덱스의 종류

- **DropDups 인덱스**
  - 동일한 값이 여러 개 저장되어 있는 필드에 DropDups 인덱스를 생성하면 최초 입력된 도큐먼트만 남고 나머지 도큐먼트는 제거된다.
  - 방식
    - db.employees.ensureIndex({eno:1}, {unique:true, dropDups : true})
- 데이터 중복이 발생하도록 eno : 5678 입력하여 두 개가 저장된 상태

```
> db.emp.dropIndex({eno:1})
{ "nIndexesWas" : 5, "ok" : 1 }

>
>
> db.emp.insert({eno:5678, ename:"smith"})
>

> db.emp.insert({eno:5678, ename:'ADAM'})
>

> db.emp.find({eno:5678})
[{"_id": ObjectId("524670cdf1a9de6b99fff6dd"), "eno": 5678, "ename": "smith"}, {"_id": ObjectId("52467101f1a9de6b99fff6de"), "eno": 5678, "ename": "ADAM"}]
```

# MongoDB의 인덱스의 종류

- dropDups 인덱스 설정 후 최초 입력된 도큐먼트만 남고 나머지 도큐먼트는 삭제된 상태

```
> db.emp.ensureIndex({eno:1}, {unique:true, dropDups : true})
>
> db.emp.find({eno:5678})
{ "_id" : ObjectId("524670cdf1a9de6b99ffff6dd"), "eno" : 5678, "ename" : "smith"
}
```

- **Sparse** 인덱스

- 전체를 대상으로 full collection 검색하는 것보다 해당 조건을 만족하는 도큐먼트로만 Sparse 인덱스를 생성하고 검색하기 때문에 빠른 성능을 기대할 수 있다. 일반적인 Balance Tree 인덱스의 경우에는 반드시 인덱스가 생성된 검색 조건으로 데이터를 검색했을 때 인덱스 스캔이 가능하지만, Sparse 인덱스의 경우에는 검색 조건이 없이 전체 컬렉션에 대한 검색 문장 만으로도 인덱스 스캔이 가능하다.
- 즉, 데이터 양과 넓은 조건의 데이터라면 Sparse 인덱스가 적합하지 않다. 데이터 양과 밀도가 낮은 경우에는 IO를 줄일 수 있기 때문에 Sparse 인덱스 사용 권장한다.
- Comm 필드의 값이 존재하는 도큐먼트들만으로 인덱스가 생성된다.
  - db.employees.ensureIndex({comm : 1},{sparse: true})
  - db.employees.find().sort({comm:-1})

# MongoDB의 인덱스의 종류

- 전체 검색시 sparse 인덱스에 생성되어 있는 도큐먼트들만 출력

```
> db.emp.ensureIndex({comm:1}, {sparse:true})
>
> db.emp.find().sort({comm:-1})
{ "_id" : ObjectId("524682f456229da245d5adc5"), "empno" : 7654, "ename" : "MARTIN", "job" : "SALESMAN", "hiredate" : "28-09-1981", "sal" : 1250, "comm" : 1400, "deptno" : 30 }
{ "_id" : ObjectId("524682f456229da245d5adc3"), "empno" : 7521, "ename" : "WARD", "job" : "SALESMAN", "hiredate" : "22-02-1981", "sal" : 1250, "comm" : 500, "deptno" : 30 }
{ "_id" : ObjectId("524682f456229da245d5adc2"), "empno" : 7499, "ename" : "ALLEN", "job" : "SALESMAN", "hiredate" : "20-02-1981", "sal" : 1600, "comm" : 300, "deptno" : 30 }
```

- sparse 인덱스를 삭제하고 전체 검색시 14개 도큐먼트 출력

```
> db.emp.dropIndex({comm:1})
{ "nIndexesWas" : 2, "ok" : 1 }
>
> db.emp.find().sort({comm:-1})
{ "_id" : ObjectId("524682f456229da245d5adc5"), "empno" : 7654, "ename" : "MARTIN", "job" : "SALESMAN", "hiredate" : "28-09-1981", "sal" : 1250, "comm" : 1400, "deptno" : 30 }
{ "_id" : ObjectId("524682f456229da245d5adc3"), "empno" : 7521, "ename" : "WARD", "job" : "SALESMAN", "hiredate" : "22-02-1981", "sal" : 1250, "comm" : 500, "deptno" : 30 }
{ "_id" : ObjectId("524682f456229da245d5adc2"), "empno" : 7499, "ename" : "ALLEN", "job" : "SALESMAN", "hiredate" : "20-02-1981", "sal" : 1600, "comm" : 300, "deptno" : 30 }
{ "_id" : ObjectId("524682f456229da245d5adc1"), "empno" : 7369, "ename" : "SMITH", "job" : "CLERK", "hiredate" : "17-12-1980", "sal" : 800, "deptno" : 20 }
{ "_id" : ObjectId("524682f456229da245d5adc4"), "empno" : 7566, "ename" : "JONES", "job" : "MANAGER", "hiredate" : "02-04-1981", "sal" : 2975, "deptno" : 20 }
{ "_id" : ObjectId("524682f456229da245d5adc6"), "empno" : 7698, "ename" : "BLAKE", "job" : "MANAGER", "hiredate" : "01-05-1981", "sal" : 2850, "deptno" : 30 }
{ "_id" : ObjectId("524682f456229da245d5adc7"), "empno" : 7782, "ename" : "CLARK", "job" : "MANAGER", "hiredate" : "09-06-1981", "sal" : 2450, "deptno" : 10 }
{ "_id" : ObjectId("524682f456229da245d5adc8"), "empno" : 7788, "ename" : "SCOTT", "job" : "ANALYST", "hiredate" : "13-06-1987", "sal" : 3000, "deptno" : 20 }
{ "_id" : ObjectId("524682f456229da245d5adc9"), "empno" : 7839, "ename" : "PRESIDENT", "job" : "CEO", "hiredate" : "17-11-1981", "sal" : 5000, "deptno" : 10 }
{ "_id" : ObjectId("524682f456229da245d5adca"), "empno" : 7844, "ename" : "TURNER", "job" : "SALESMAN", "hiredate" : "08-09-1981", "sal" : 1500, "deptno" : 30 }

{ "_id" : ObjectId("524682f456229da245d5adcb"), "empno" : 7876, "ename" : "ADAMS", "job" : "CLERK", "hiredate" : "13-06-1987", "sal" : 1100, "deptno" : 20 }
{ "_id" : ObjectId("524682f456229da245d5adcc"), "empno" : 7900, "ename" : "JAMES", "job" : "SALESMAN", "hiredate" : "03-12-1981", "sal" : 1500, "deptno" : 30 }
```

# MongoDB의 인덱스의 종류

- **Background 인덱스**
  - 빅데이터를 처리하는데 인덱스를 생성하면 그 크기는 대용량의 인덱스가 생성이 될 것이며, 이 인덱스를 생성하는데 많은 시스템 자원이 필요할 것이다. 이는 시스템의 성능의 저하를 가지고 올 수 있다.
  - 이런 문제점을 해결할 수 있는 것은 Background 인덱스이다.
  - Background 인덱스 자원이 충분할 때에는 인덱스를 즉시 생성한다.
  - Background 인덱스 시스템 자원이 부족할 경우 인덱스의 생성을 잠시 보류했다가 일정 공간 이상의 자원이 확보되면 인덱스 생성 작업을 이어서 수행한다.
  - 사용버전 : 1.3.2 버전 이후
  - Background 인덱스 사용
    - db.employees.ensureIndex({hiredate:1},{background:true})

```
> db.emp.ensureIndex({hiredate:1}, {background:true})
>
> db.emp.find({hiredate:"20-02-1981"})
{ "_id" : ObjectId("524682f456229da245d5adc2"), "empno" : 7499, "ename" : "ALLEN",
  "job" : "SALESMAN", "hiredate" : "20-02-1981", "sal" : 1600, "comm" : 300, "deptno" : 30 }
>
```

# MongoDB의 인덱스의 종류

- **Coverd 인덱스**

- 인덱스 스캔의 경우, 인덱스를 통해 조건을 만족하는 데이터를 검색한 후 컬렉션을 추가 검색을 하여 데이터를 검색하는데, 빅데이터 검색시에는 불필요한 IO 문제로 성능 저연이 나올 수 있다.
- 조건을 만족하는 대상 필드만을 Covered 인덱스를 생성하고 그 인덱스만을 검색하게 하면 된다.

```
> db.emp.ensureIndex({deptno:1, ename:1})
> db.emp.find({deptno : 10, ename:"CLARK"}, {_id:0, ename:1})
[{"ename" : "CLARK" }

> db.emp.find({deptno : 10, ename:"CLARK"}, {_id:0, ename:1}).explain()
{
    "cursor" : "BtreeCursor deptno_1_ename_1",
    "isMultiKey" : false,
    "n" : 1,
    "nscannedObjects" : 0,
    "nscanned" : 1,
    "nscannedObjectsAllPlans" : 0,
    "nscannedAllPlans" : 1,
    "scanAndOrder" : false,
    "indexOnly" : true,
    "nYields" : 0,
    "nchunkskips" : 0,
    "millis" : 0,
    "indexBounds" : {
        "deptno" : [
            [
                10,
                10
            ]
        ],
        "ename" : [
            [
                "CLARK",
                "CLARK"
            ]
        ]
    },
    "server" : "mainpc:27017"
}
```

"indexOnly" : true

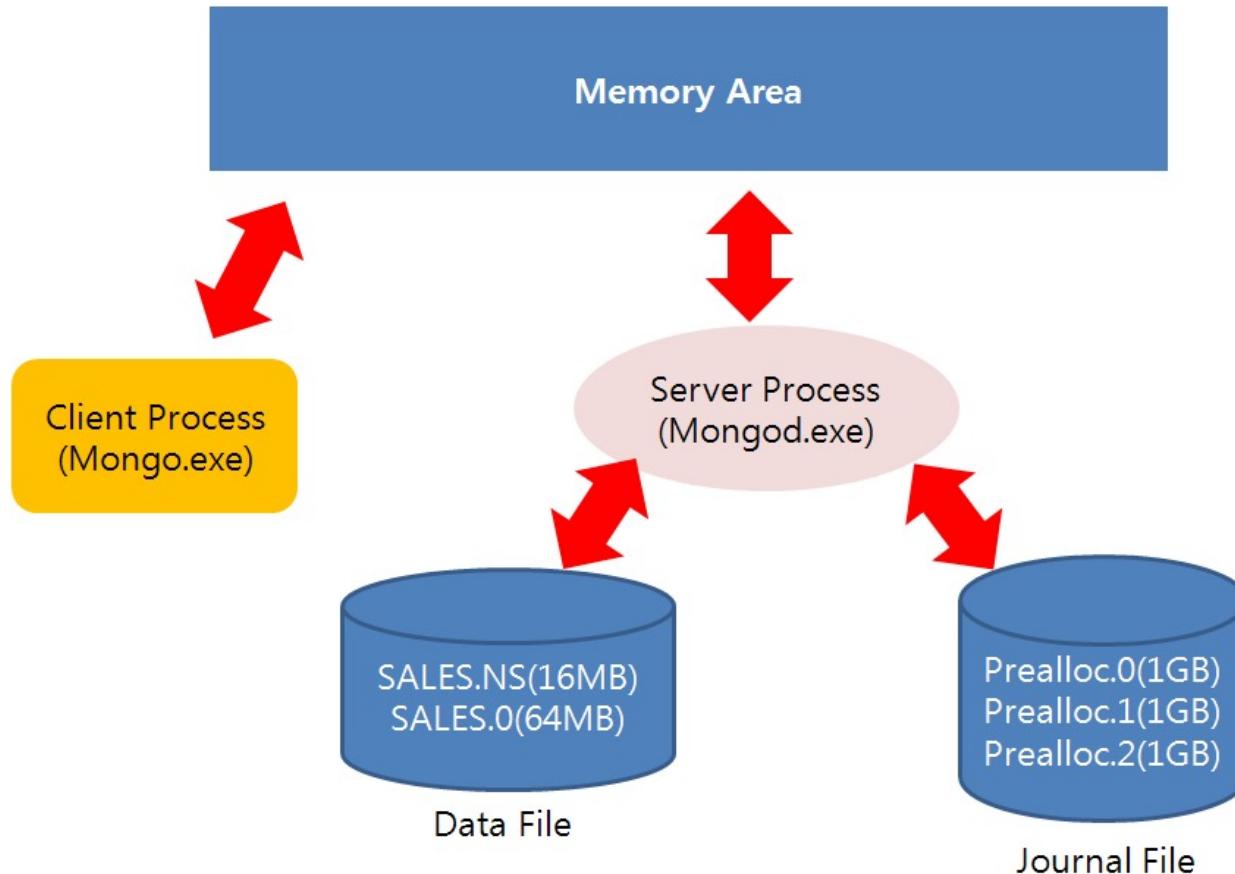
// 인덱스 만으로 조건을  
만족하는 데이터 검색

# 9

## **mongoDB의 구조**

# MongoDB의 구조

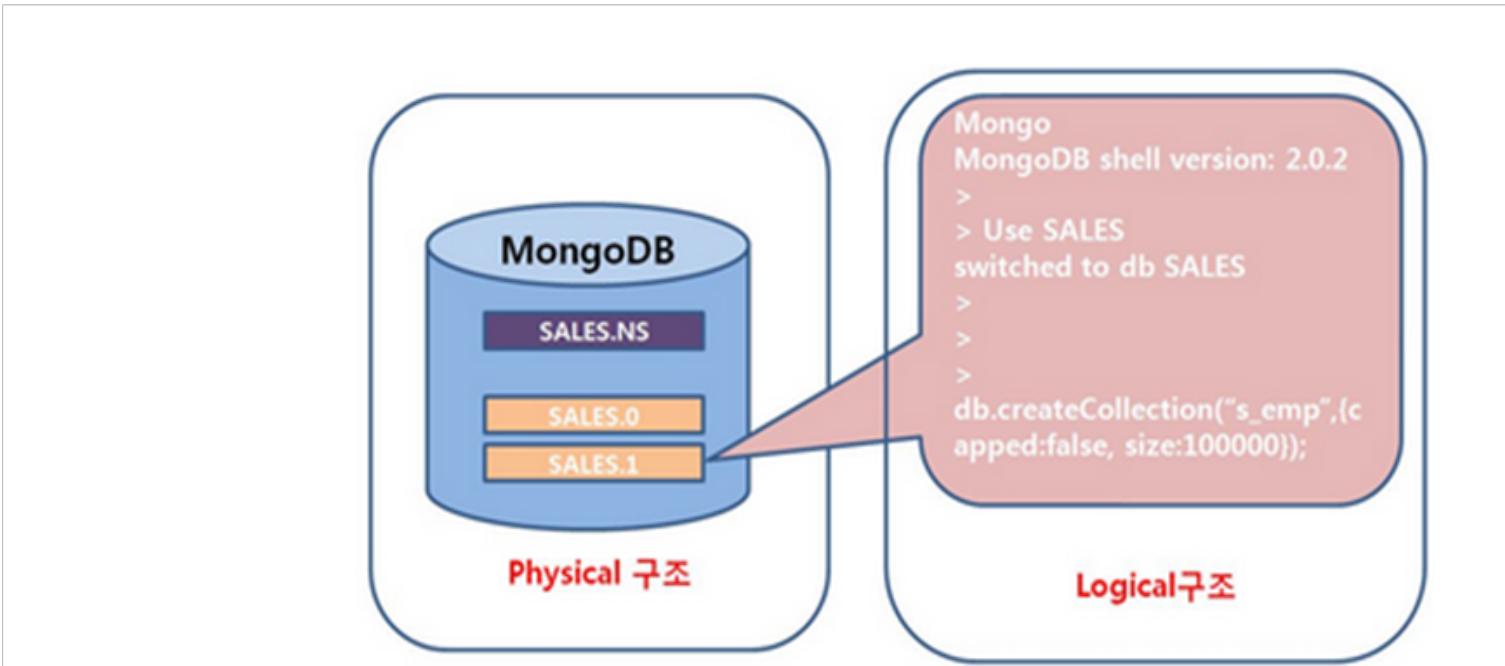
- MongoDB 구조
  - 기본적으로 3가지 영역으로 나눌 수 있다.



# MongoDB의 구조

- **MongoDB 구조**
  - 프로세스 영역
    - 클라이언트 프로세스 : Mongo.exe를 통해 MongoDB에 접속할 때 사용되는 프로세스
    - 서버 프로세스 : Mongod.exe를 통해 MongoDB의 시작과 종료를 시킬 때 사용하는 프로세스
  - 메모리 영역
    - Memory Cache 영역 : MongoDB를 위한 전용 메모리 영역이며 데이터의 쓰기와 읽기 작업이 수행되는 메모리 영역
    - Journal 영역 : 읽기와 쓰기 작업이 발생할 때 예기치 못한 장애가 발생하는 경우를 대비하여 데이터를 백업 시켜 놓는 전용 메모리 영역
    - Resident(WorkingSet) : 실제 데이터가 처리되는 물리적 메모리 영역
  - 파일 영역
    - 데이터 파일 : SALES.NS, SALES.0 파일을 데이터 파일이라고 한다. 사용자가 입력한 데이터가 저장되는 실제 물리적 공간
    - JOURNAL 파일 : 데이터가 입력, 수정, 삭제 될 때 가장 먼저 저널 메모리 영역에 데이터가 백업되는데 예기치 못한 장애로 인해 시스템이 종료되면 데이터의 소실이 일어납니다. 이 메모리 영역의 백업 데이터는 디스크 상의 저널 파일에 저장된다.
    - LOG 파일 : 서버가 실행이 되면 내부적으로 발생하는 로그 정보를 파일로 저장할 수 있다. 이를 실행 할 때에는 서버 실행 시 추가 설정이 필요하다.

# MongoDB의 물리적 구조



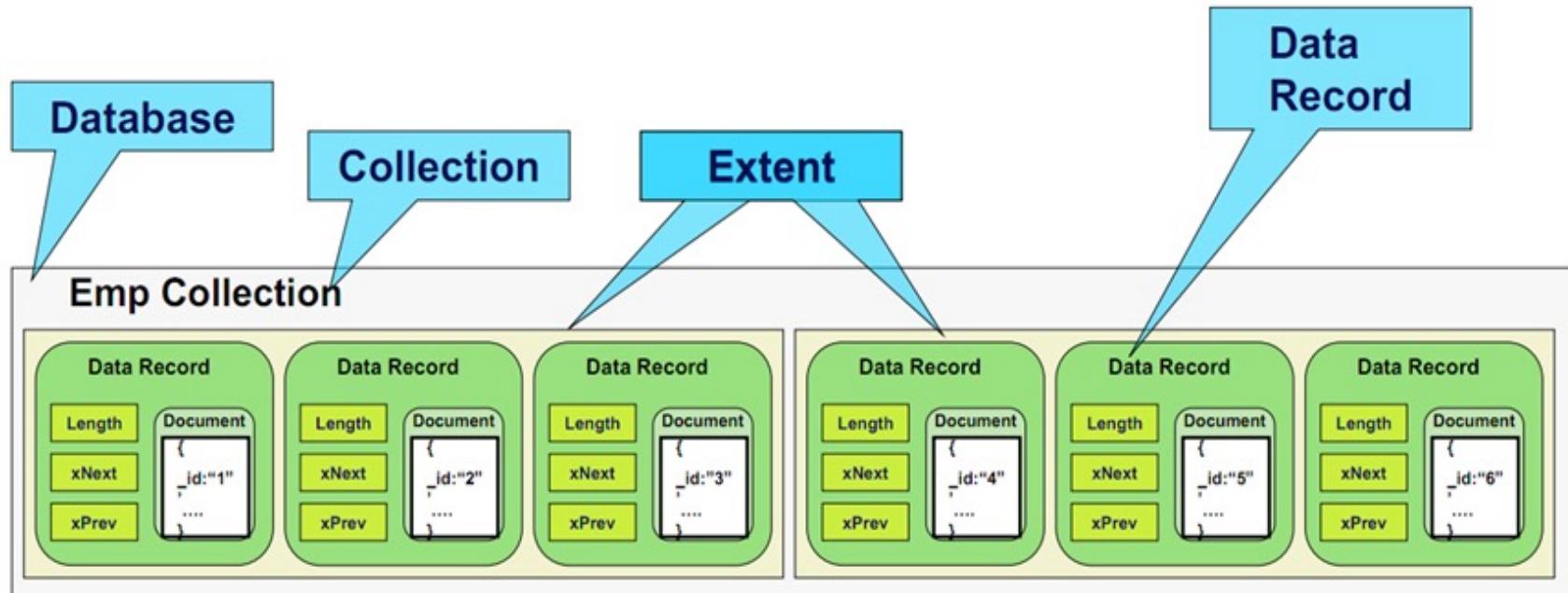
- 1) 32 bit에서 네임스페이스의 최초 크기는 16MB가 할당되며 DataFile은 32MB 크기로 생성된다. Next 크기는 32MB의 배수로 증가하되 최대 2GB입니다.
- 2) 네임스페이스에는 collection의 first Extent와 LastExtent에 대한 정보와 Meta Data, 인덱스, FreeList 정보가 저장된다.

# MongoDB의 물리적 구조

- 물리적 구조
  - 데이터 파일 및 저널 파일
    - 데이터 파일 확장자 : .0, .1, .2 .... 형식으로 생성
    - 32bit 시스템 환경에서 .ns 네임스페이스 최초 크기는 16MB가 할당
    - 데이터 파일과 네임스페이스 파일의 크기는 32MB 배수로 증가
    - 최대 2GB
    - 저널 파일의 기본 크기는 1GB이며 최대 크기에 도달하면 새로운 파일이 1GB 단위로 추가

```
>
> show dbs
admin      0.0625GB
local      0.03125GB
mdb        0.0625GB
sales      0.03125GB
test        0.0625GB
>
```

# MongoDB의 논리적 구조



- 1) MongoDB의 논리적 구조는 Database -> Collections -> Extents -> Data Records -> Documents로 구성
- 2) Collection 크기는 최초 생성 시점에 결정되며 Extent 기본 크기는 8k이며 데이터 크기가 작은 경우에는 4k로 생성

# MongoDB의 논리적 구조

- **논리적 구조**

- MongoDB에서 데이터를 저장하는 가장 작은 단위는 도큐먼트이다.
- 켈랙션 크기 결정 및 생성
  - db.createCollection("s\_emp", {capped:**true**, size:8192})
- Capped collection을 최초 extent 8192 byte 크기로 생성

```
> db.createCollection("s_emp", {capped:false, size:8192})
{ "ok" : 1 }
>
> db.s_emp.validate()
{
    "ns" : "sales.s_emp",
    "firstExtent" : "0:10000 ns:sales.s_emp",
    "lastExtent" : "0:10000 ns:sales.s_emp",
    "extentCount" : 1,
    "datasize" : 0,
    "nrecords" : 0,
    "lastExtentSize" : 8192,
    "padding" : 1,
    "firstExtentDetails" : {
        "loc" : "0:10000",
        "xnext" : "null",
        "xprev" : "null",
        "nsdiag" : "sales.s_emp",
        "size" : 8192,
        "firstRecord" : "null",
        "lastRecord" : "null"
    }
}
```

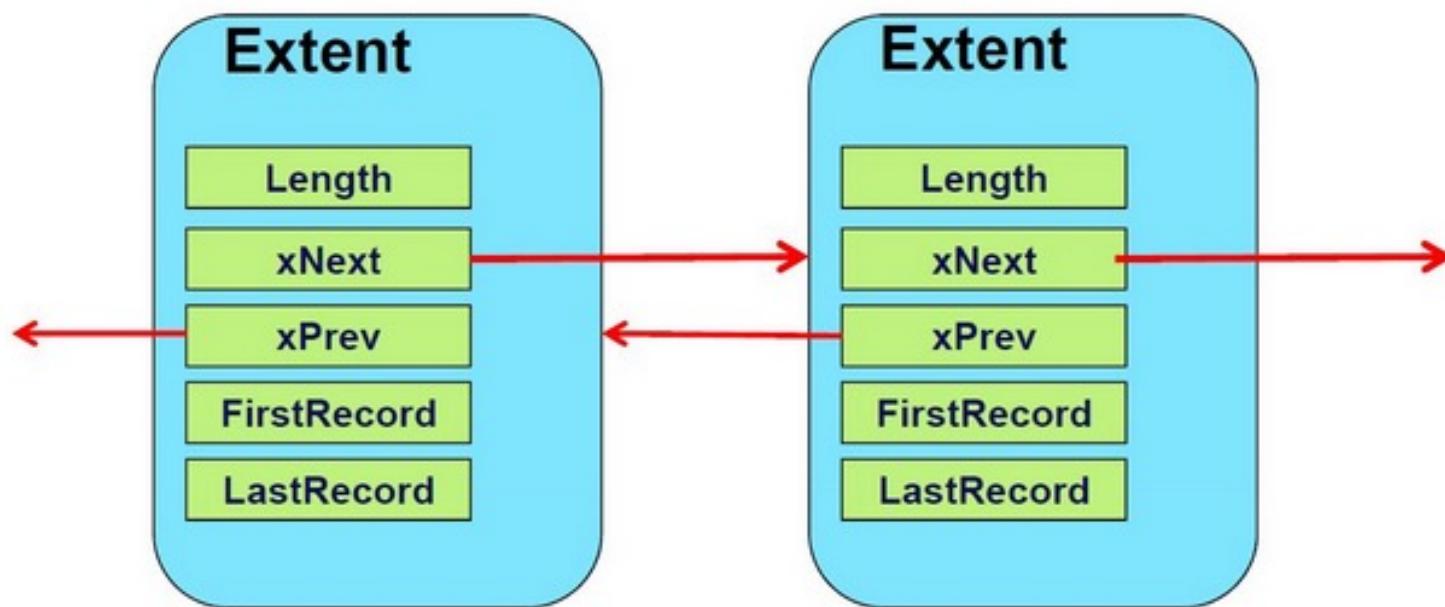
# MongoDB의 논리적 구조

- 논리적 구조

```
"extentCount" : 1,           //활성화된 Extent 개수
"datasize" : 0,             //collection에 저장된 데이터 크기
"nrecords" : 0,             // 저장된 Document 수
"lastExtentSize" : 8192,     // 마지막 Extent의 크기
"padding" : 1,
"firstExtentDetails" : {
  "loc" : "0:10000",
  "xnext" : "null",
  "xprev" : "null",
  "nsdiag" : "sales.s_emp",
  "size" : 8192,           // 활성화된 Extent 크기
  "firstRecord" : "null",
  "lastRecord" : "null"
},
```

# MongoDB의 논리적 구조

- 논리적 구조
  - Extent와 data record



# MongoDB의 논리적 구조

- **논리적 구조**

- Extent와 data record

- MongoDB에서는 Extent 데이터가 저장되는 논리적인 단위로, 하나의 extent에 얼마나 적절하게 데이터를 잘 저장하는냐는 읽기/쓰기에 있어 성능을 좌우하는 요소가 될 수 있다.
    - 하나의 컬렉션을 위한 가장 적절한 익스텐트 크기를 결정하기 위해서는 적절한 분석과 논리적 설계가 요구된다.
    - 하나의 Data Record는 하나의 Document 정보와 이전, 이후 Document가 저장되어 있는 주소 정보를 함께 저장하고 있다.

```
> for(var n=201300001; n <=201312345; n++) db.s_emp.save({empno : n , m :"test"}  
)  
> db.s_emp.count()  
12346
```

```
> for(var n=201300001; n <=201312345; n++) db.s_emp.save({empno : n ,m :"test"})  
//빅 데이터를 입력하고 Extent 활성상태를 확인
```

```
> db.s_emp.count()  
12346
```

# MongoDB의 논리적 구조

- 논리적 구조

- Validate() : 모든 속성을 나타내 주는 메소드

```
> db.s_emp.validate()
{
    "ns" : "sales.s_emp",
    "firstExtent" : "0:10000 ns:sales.s_emp",
    "lastExtent" : "0:179000 ns:sales.s_emp",
    "extentCount" : 5,
    "datasize" : 691360,
    "nrecords" : 12346,
    "lastExtentSize" : 2097152,
    "padding" : 1,
    "firstExtentDetails" : {
        "loc" : "0:10000",
        "xnext" : "0:1d000",
        "xprev" : "null",
        "nsdiag" : "sales.s_emp",
        "size" : 8192,
        "firstRecord" : "0:100b0",
        "lastRecord" : "0:11f90"
    },
    "lastExtentDetails" : {
        "loc" : "0:179000",
        "xnext" : "null",
        "xprev" : "0:69000",
        "nsdiag" : "sales.s_emp",
        "size" : 2097152,
        "firstRecord" : "0:1790b0",
        "lastRecord" : "0:1a83d8"
    }
}
```

# MongoDB의 논리적 구조

- 논리적 구조

- Validate() : 모든 속성을 나타내 주는 메소드

```
"ns" : "sales.s_emp",                                // 데이터베이스명과 컬렉션명
"firstExtent" : "0:10000 ns:sales.s_emp",
"lastExtent" : "0:179000 ns:sales.s_emp",
"extentCount" : 5,                                    // 현재 활성화된 Extent 수
"datasize" : 691360,                                 // 전체 데이터 크기
"nrecords" : 12346,                                 // 입력된 document 수
"lastExtentSize" : 2097152,
"padding" : 1,
"firstExtentDetails" : {
  "loc" : "0:10000",
  "xnext" : "0:1d000",
  "xprev" : "null",
  "nsdiag" : "sales.s_emp",
  "size" : 8192,
  "firstRecord" : "0:100b0",
  "lastRecord" : "0:11f90"
},
```

# MongoDB의 논리적 구조

- 논리적 구조
  - s\_emp에 확장된 모든 Extent 상태를 출력한다.

```
> db.s_emp.validate({full : true})
{
  "ns" : "sales.s_emp",
  "firstExtent" : "0:10000 ns:sales.s_emp",
  "lastExtent" : "0:179000 ns:sales.s_emp",
  "extentCount" : 5,
  "extents" : [
    {
      "loc" : "0:10000",
      "xnext" : "0:1d000",
      "xprev" : "null",
      "nsdiag" : "sales.s_emp",
      "size" : 8192,
      "firstRecord" : "0:100b0",
      "lastRecord" : "0:11f90"
    },
    {
      "loc" : "0:1d000",
      "xnext" : "0:25000",
      "xprev" : "0:10000",
      "nsdiag" : "sales.s_emp",
      "size" : 32768,
      "firstRecord" : "0:1d0b0",
      "lastRecord" : "0:24f88"
    },
    {
      "loc" : "0:25000",
      "xnext" : "0:69000",
      "xprev" : "0:1d000",
      "nsdiag" : "sales.s_emp",
      "size" : 131072,
      "firstRecord" : "0:250b0",
      "lastRecord" : "0:44fb8"
    },
    {
      "loc" : "0:69000",
      "xnext" : "0:179000",
      "xprev" : "0:25000",
      "nsdiag" : "sales.s_emp",
      "size" : 131072,
      "firstRecord" : "0:690b0",
      "lastRecord" : "0:99fb8"
    }
  ]
}
```

# MongoDB의 논리적 구조

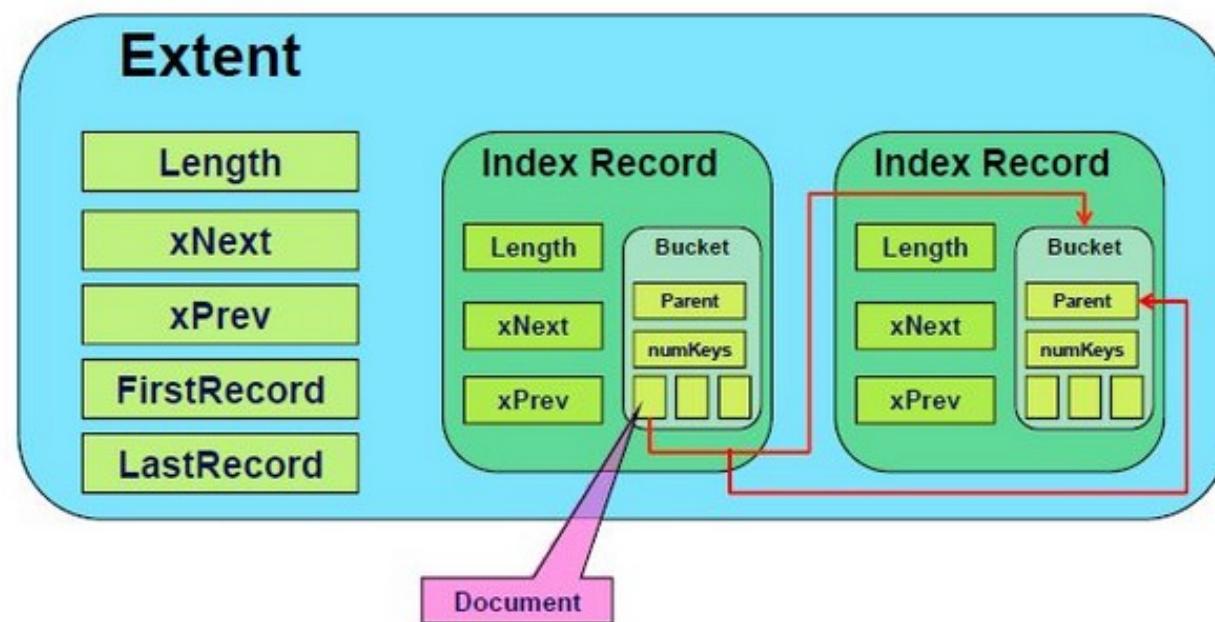
- 논리적 구조
  - s\_emp에 확장된 모든 Extent 상태를 출력한다.

```
{  
  "loc" : "0:10000",    // 첫 번째 Extent 정보  
  "xnext" : "0:1d000", // 두 번째 Extent와 연결  
  "xprev" : "null",  
  "nsdiag" : "sales.s_emp",  
  "size" : 8192,        // 첫 번째 Extent 크기  
  "firstRecord" : "0:100b0",  
  "lastRecord" : "0:11f90"  
},  
{  
  "loc" : "0:1d000",    // 두 번째 Extent 정보  
  "xnext" : "0:25000", // 세 번째 Extent와 연결  
  "xprev" : "0:10000", // 첫 번째 Extent와 연결  
  "nsdiag" : "sales.s_emp",  
  "size" : 32768,  
  "firstRecord" : "0:1d0b0",  
  "lastRecord" : "0:24f88"  
},
```

```
{  
  "loc" : "0:25000",    // 세 번째 Extent 정보  
  "xnext" : "0:69000",  
  "xprev" : "0:1d000", // 두 번째 Extent와 연결  
  "nsdiag" : "sales.s_emp",  
  "size" : 131072,      // 세 번째 Extent 크기  
  "firstRecord" : "0:250b0",  
  "lastRecord" : "0:44fb8"  
},
```

# MongoDB의 논리적 구조

- 논리적 구조
  - Extent와 Index record
    - 인덱스는 콜렉션과 동일하게 여러 개의 Extent로 구성되며 하나의 Extent는 여러 개의 Index Records로 구성된다.
    - Index Record 데이터 파일 중에 .ns 확장자를 가진 네임스페이스 파일에 저장되기 때문에 인덱스의 개수와 크기에 따라 네임스페이스의 크기도 결정되어어야 한다.



# MongoDB의 논리적 구조

- 논리적 구조
  - 인덱스에 확장된 모든 extent 상태를 출력

```
> db.s_emp.ensureIndex({empno:1})
>
> db.s_emp.$_id_.validate({full:true})
{
    "ns" : "sales.s_emp.$_id_",
    "firstExtent" : "0:14000 ns:sales.s_emp.$_id_",
    "lastExtent" : "0:e9000 ns:sales.s_emp.$_id_",
    "extentCount" : 3,
    "extents" : [
        {
            "loc" : "0:14000",
            "xnext" : "0:45000",
            "xprev" : "null",
            "nsdiag" : "sales.s_emp.$_id_",
            "size" : 36864,
            "firstRecord" : "0:15000",
            "lastRecord" : "0:1b000"
        },
        {
            "loc" : "0:45000",
            "xnext" : "0:e9000",
            "xprev" : "0:14000",
            "nsdiag" : "sales.s_emp.$_id_",
            "size" : 147456,
            "firstRecord" : "0:46000",
            "lastRecord" : "0:66000"
        },
        {
            "loc" : "0:e9000",
            "xnext" : "null",
            "xprev" : "0:45000",
            "nsdiag" : "sales.s_emp.$_id_",
            "size" : 589824,
            "firstRecord" : "0:ea000",
            "lastRecord" : "0:122000"
        }
    ]
}
```

# MongoDB의 논리적 구조

- 논리적 구조
  - 인덱스에 확장된 모든 익스텐츠 상태를 출력

```
"ns" : "sales.s_emp.$_id_",
"firstExtent" : "0:14000 ns:sales.s_emp.$_id_",
"lastExtent" : "0:e9000 ns:sales.s_emp.$_id_",
"extentCount" : 3,      // 활성화된 Extent 수
"extents" : [
{
  "loc" : "0:14000",    // 첫 번째 Extent 정보
  "xnnext" : "0:45000",
  "xprev" : "null",
  "nsdiag" : "sales.s_emp.$_id_",
  "size" : 36864,
  "firstRecord" : "0:15000",
  "lastRecord" : "0:1b000"
},
{
  "loc" : "0:45000",    // 두 번째 Extent 정보
  "xnnext" : "0:e9000",
```

```
  "xprev" : "0:14000",
  "nsdiag" : "sales.s_emp.$_id_",
  "size" : 147456,
  "firstRecord" : "0:46000",
  "lastRecord" : "0:66000"
},
{
  "loc" : "0:e9000",    // 세 번째 Extent 정보
  "xnnext" : "null",
  "xprev" : "0:45000",
  "nsdiag" : "sales.s_emp.$_id_",
  "size" : 589824,
  "firstRecord" : "0:ea000",
  "lastRecord" : "0:122000"
},
],
"datasize" : 408800,
"nrecords" : 50,        // 인덱스의 key 수
"lastExtentSize" : 589824,
```

# MongoDB의 논리적 구조

- 논리적 구조
  - 현재 생성된 Collection 상태 분석 결과

```
> db.printCollectionStats()
blog
{
    "ns" : "sales.blog",
    "count" : 1,
    "size" : 96,
    "avgObjSize" : 96,
    "storageSize" : 8192,
    "numExtents" : 1,
    "nindexes" : 1,
    "lastExtentSize" : 8192,
    "paddingFactor" : 1,
    "systemFlags" : 1,
    "userFlags" : 0,
    "totalIndexSize" : 8176,
    "indexSizes" : {
        "_id_" : 8176
    },
    "ok" : 1
}
---
```

```
s_emp
{
    "ns" : "sales.s_emp",
    "count" : 12346,
    "size" : 691360,
    "avgObjSize" : 55.99870403369513,
    "storageSize" : 2793472,
    "numExtents" : 5,
    "nindexes" : 2,
    "lastExtentSize" : 2097152,
    "paddingFactor" : 1,
    "systemFlags" : 1,
```

# MongoDB의 논리적 구조

- 논리적 구조
  - 현재 생성된 Collection 상태 분석 결과

```
"s_emp
{
  "ns" : "sales.s_emp",                                //컬렉션 이름
  "count" : 12346,                                     //전체 도큐먼트 수
  "size" : 691360,                                     //전체 데이터 크기
  "avgObjSize" : 55.99870403369513,                  //오브젝트 평균 크기
  "storageSize" : 2793472,                             //저장공간 크기
  "numExtents" : 5,                                    //확장된 익스텐트 수
  "nindexes" : 2,                                     //생성되어 있는 인덱스 수
  "lastExtentSize" : 2097152,                          //활성화된 마지막 익스텐트의 크기
  "paddingFactor" : 1,
  "systemFlags" : 1,
  "userFlags" : 0,
  "totalIndexSize" : 727664,                           //인덱스의 전체 크기
  "indexSizes" : {
    "_id_" : 408800,
    "empno_1" : 318864 },
  "ok" : 1
}
```

# MongoDB의 논리적 구조

- 논리적 구조
  - 데이터베이스의 상태

```
> db.stats()
{
    "db" : "sales",
    "collections" : 4,
    "objects" : 12356,
    "avgObjSize" : 56.001294917449016,
    "dataSize" : 691952,
    "storageSize" : 2809856,
    "numExtents" : 8,
    "indexes" : 3,
    "indexSize" : 735840,
    "filesize" : 16777216,
    "nsSizeMB" : 16,
    "dataFileVersion" : {
        "major" : 4,
        "minor" : 5
    },
    "ok" : 1
}
```

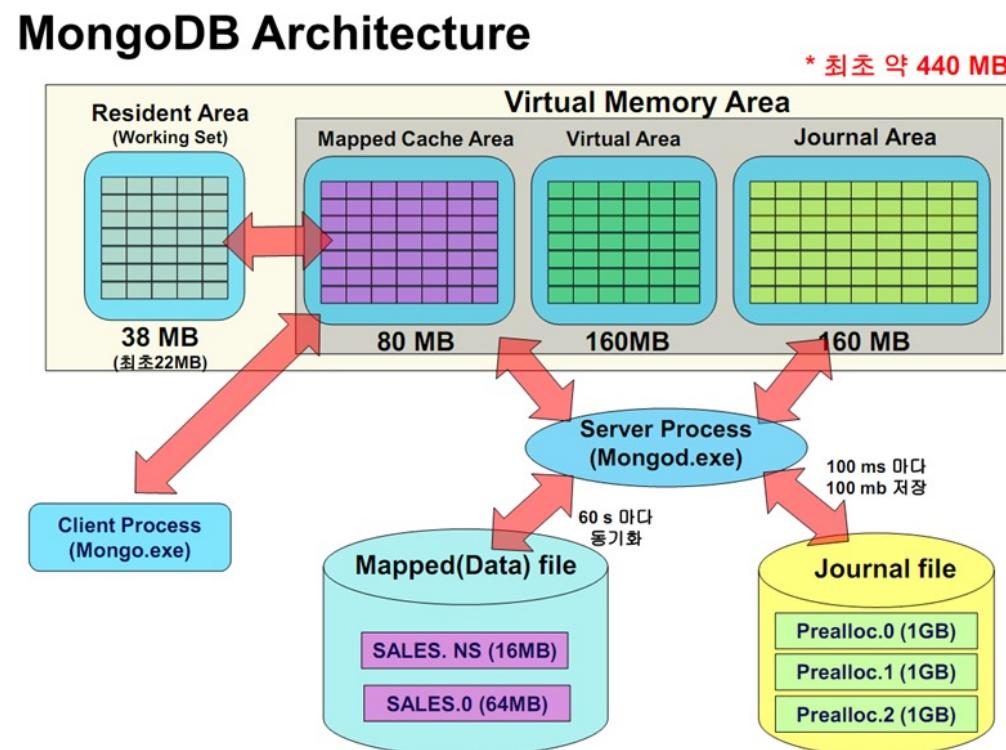
# MongoDB의 논리적 구조

- 논리적 구조
  - 데이터베이스의 상태

```
"db" : "sales",                                // 데이터베이스명
"collections" : 4,                             // 콜렉션 수
"objects" : 12356,                            // 현재 오브젝트 크기
"avgObjSize" : 56.001294917449016,           // 오브젝트의 평균 크기
"dataSize" : 691952,                           // 전체 데이터 크기
"storageSize" : 2809856,                        // 현재 할당된 공간 크기
"numExtents" : 8,                             // 할당된 총 익스텐트 개수
"indexes" : 3,                               // 인덱스 수
"indexSize" : 735840,                           // 인덱스 크기
"fileSize" : 16777216,                          // 현재 데이터 파일의 크기
"nsSizeMB" : 16,                             // 네임스페이스의 현재 크기
"dataFileVersion" : {
  "major" : 4,
  "minor" : 5}
```

# MongoDB의 논리적 구조

- 논리적 구조
  - 메모리 구조
    - 크게 두 가지 영역으로 나눌 수 있다.
      - Virtual Memory Area
      - Resident Area(Working Set)



# MongoDB의 논리적 구조

- **메모리 구조**
  - Virtual Memory Area
    - Mapped Cache Area : 데이터가 캐시될 메모리 영역
    - Virtual Area : 캐시 영역이 부족할 때 사용될 추가 캐시 영역
    - Journal Area : 사용자의 작업 내용을 실 시간으로 백업할 메모리 영역
  - Resident Area(Working Set)
    - 데이터를 처리하는 실제 메모리 영역

# MongoDB의 논리적 구조

- 메모리 구조
  - db.serverStatus().mem / db.serverStatus().extra\_info

```
> db.serverStatus().mem
{
    "bits" : 32,
    "resident" : 31,
    "virtual" : 169,
    "supported" : true,
    "mapped" : 32,
    "mappedWithJournal" : 64
}
>
> db.serverStatus().extra_info
{
    "note" : "fields vary by platform",
    "page_faults" : 9213,
    "usagePageFileMB" : 68,
    "totalPageFileMB" : 4961,
    "availPageFileMB" : 3695,
    "ramMB" : 3071
}
```

# MongoDB의 논리적 구조

- 메모리 구조
  - db.serverStatus().mem / db.serverStatus().extra\_info

```
db.serverStatus().mem
{
  "bits" : 32,                                // 시스템의 처리 사양
  "resident" : 31,                             // working set 저장을 위한 메모리 크기
  "virtual" : 169,                            // virtual 메모리 현재 크기
  "supported" : true,
  "mapped" : 32,                                // Mapped File 현재 크기
  "mappedWithJournal" : 64                      // Journal 메모리 현재 크기
}
db.serverStatus().extra_info
{
  "note" : "fields vary by platform",
  "page_faults" : 9213,                         // 현재 메모리 pageFault 수
  "usagePageFileMB" : 68,
  "totalPageFileMB" : 4961,
  "availPageFileMB" : 3695,                     // 현재 시스템 메모리 크기
  "ramMB" : 3071
}
```

# MongoDB의 논리적 구조

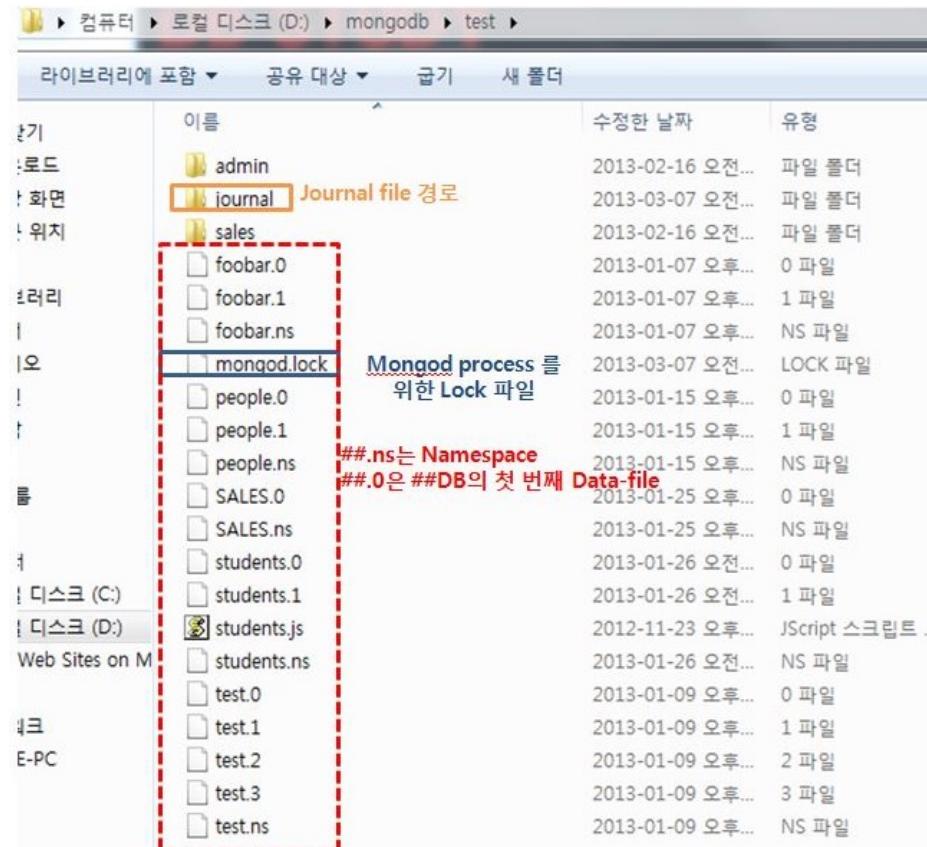
- **적정 메모리 요구 사항**
  - MongoDB는 Memory Mapped File 기법에 의해 데이터를 처리하기 때문에 최초 메모리 크기는 네임스페이스와 Data-file에 맞는 적정한 Mapped Area 크기가 요구된다.
  - 사용자의 데이터를 위한 Virtual Memory와 함께 MongoDB 서버를 원활하게 운영하기 위해 Resident Area(Working Set)가 요구된다. 이 공간은 서버에 의해 동적으로 할당되며, PageFault 가 발생하면 커진다.
  - Data File 크기가 2GB인 경우 요구되는 RAM 메모리는 약 10GB~12GB 정도가 요구된다. 시스템 메모리가 부족할 경우 성능 저하의 현상이 발생할 수도 있다.

# MongoDB의 논리적 구조

- **저널(Journal) 파일**
  - 사용자 데이터의 유실을 방지하기 위해 별도의 파일에 저장한다.
    - --dbpath로 정의된 경로에 저널 파일이 생성
    - 32 bit 시스템의 경우에는 MongoDB 인스턴트를 활성화 할 때 반드시 —journal 옵션을 부여해야 하며, 64bit에서는 부여하지 않아도 자동적으로 활성화된다.
    - 기본 크기는 1GB이며 여러 개의 파일로 구성된다.
  - 메모리 맵에 저장하기 전에 저널링 파일이 먼저 저장
  - 기본적으로 매 100ms 마다 100mb 를 저장함
    - mongoDB 구동할 때 —journing [CommitInterval]을 정의할 수 있으며, 2ms~300ms 범위에서 정의할 수 있다.
  - 저널 모드가 요구되는 경우
    - 싱글 모드 : Data의 무결성 보장을 위해 반드시 필요하다.
    - Replica Set : 최소 1 node에 정의해야 함

# MongoDB의 논리적 구조

- 저널(Journal) 파일
  - 각 데이터베이스의 데이터 파일과 분리된 물리적 디스크에 생성해야 좋은 성능 기대할 수 있다.
    - --directoryperdb 옵션절을 통해 각 데이터베이스의 데이터 파일과 분리 저장



# MongoDB의 논리적 구조

- **GridFS**
  - 대용량의 파일을 데이터베이스 내에 업데이트하거나 또는 데이터베이스 내에 업데이트된 파일을 OS상에 다운로드 해야 하는 경우 적용할 수 있음
  - Higher V 1.8에서 최대 16MB 처리 가능, 256kbyte의 chunk사이즈가 적용됨
  - 각 Chunk는 하나의 컬렉션에 여러 개의 분리된 도큐먼트로 저장됨
- **현재의 MongoDB 현재 상태를 확인**

```
> show dbs
admin    0.0625GB
local    0.03125GB
mdb      0.0625GB
sales    0.03125GB
test     0.0625GB
>
```

# MongoDB의 논리적 구조

- **GridFS 기능 실습**

- mongolist list : upload된 파일 정보를 확인

```
c:\Documents and Settings\MyHome>mongofiles list  
connected to: 127.0.0.1
```

- mongofiles get BIG\_EMPTXT : 데이터베이스 내에 저장된 파일을 OS 상에 다시 다운로드할 때 사용한다. 해당 정보가 없을 때에는 에러 발생함.

```
C:\Documents and Settings\MyHome>mongofiles get BIG_EMP.TXT  
connected to: 127.0.0.1  
ERROR: file not found
```

- mongofiles put BIG\_EMPTXT : PUT 옵션 절을 사용하여 OS 상의 파일을 DB 내로 upload 함. 파일이 꼭 있는 폴더로 경로를 옮겨야 함.

```
D:\mongodb-win32-i386-2.4.5\mongodb-win32-i386-2.4.5\bin>mongofiles put BIG_EMP.  
TXT  
connected to: 127.0.0.1  
added file: { _id: ObjectId('524abb2db53ba93ba54fc2be'), filename: "BIG_EMP.TXT"  
, chunksize: 262144, uploadDate: new Date(1380629294015), md5: "5847679427ed8528  
c2b0103d1923bff4", length: 2892607 }  
done!
```

# MongoDB의 논리적 구조

- **GridFS 기능 실습**

- 업로드된 파일 정보를 확인한다.

```
D:\mongodb-win32-i386-2.4.5\mongodb-win32-i386-2.4.5\bin>mongofiles list
connected to: 127.0.0.1
BIG_EMP.TXT      2892607
```

- DB내에는 어떤 변화가 있는지 확인한다.

```
> show collections
emp
empo
fs.chunks
fs.files
system.indexes
system.users
things
>
```

fs.chunks  
fs.files

// Binary Chunk 데이터를 저장하고 있는 컬렉션  
// Object Meta Data를 저장하고 있는 컬렉션

# MongoDB의 논리적 구조

- **GridFS 기능 실습**
  - 업로드 되어 있는 파일 상태 정보를 확인한다.

```
> db.fs.files.find()
{ "_id" : ObjectId("524abb2db53ba93ba54fc2be"),
  "filename" : "BIG_EMP.TXT",                                //upload된 파일명
  "chunkSize" : 262144,                                     // chunk 크기
  "uploadDate" : ISODate("2013-08-01T12:08:14.015Z"),      // upload 된 날짜 정보
  "md5" : "5847679427ed8528c2b0103d1923bff4",           // file의 chunk에 의해 upload
  "length" : 2892607 }                                       // upload된 파일 크기
```

- 원하는 데이터베이스에 파일 업로드
  - -d 데이터베이스명
- 파일을 삭제
  - delete 옵션

**10**

## **mongoDB의 명령어**

# MongoDB의 환경 명령어와 관리 명령어

- **환경 명령어**

- db.help
  - DB method 의 종류를 출력한다.
- db.<CollectionName>.help()
  - mongoDB에서 제공하는 함수의 종류를 출력한다.
- ls()
  - 현재 운영체계 경로에 존재하는 파일의 종류를 출력한다.
- getMemInfo()
  - 현재 할당된 메모리 상태를 출력한다.
- help keys
  - 현재 라인에 작성된 내용을 편집하는 방법을 출력한다.
- show dbs
  - 현재 생성되어 있는 데이터베이스명과 데이터 크기를 출력한다.
- show collections
  - 해당 DB 내에 생성되어 있는 Collection 종류를 출력한다.
- hostname()
  - 현재 서버의 호스트 명을 출력한다.

# MongoDB의 환경 명령어와 관리 명령어

- 관리 명령어

- db.adminCommand({getLog:"global"})
  - 현재 DB와 관련된 로그 상태 정보를 출력한다.
- db.adminCommand({setParameter:1, notablescan:true})
  - Collection SCAN시 에러 발생을 유도하도록 환경 설정한다.
- db.adminCommand({setParameter:1, notablescan:false})
  - Collection SCAN시 에러 발생을 유도하도록 환경 설정을 해제한다.
- db.adminCommand({fsync:1})
  - 모든 데이터 파일에 대한 flush 작업을 수행한다.
- db.foo.runCommand("compact")
  - 현재 컬렉션상에 대한 단편화 제거 및 장애를 repair합니다.
- db.fsyncLock()
  - 데이터파일의 백업을 위해 DB LOCK 설정합니다.
- db.fsyncUnlock()
  - 백업 작업 후 DB LOCK을 해제합니다.
- db.currentOp()
  - Operation Progress 상태 확인
- db.foo.validate({full:true})
  - 해당 컬렉션의 논리적, 물리적 구조 정보를 출력한다.

# 11

## **mongoDB의 데이터모델링**

# MongoDB를 위한 데이터 모델링

- **데이터 프로세스 모두가 설계의 중심이다.**
  - 관계형 DB는 Data 중심의 설계를 지향
  - 클라우드 컴퓨팅 환경의 noSQL은 Data와 프로세스, 모두를 설계의 중심으로 둔다.
- **Rich Document Structure를 제공한다.**
  - 일부 데이터가 중복이 발생하더라도 빠른 데이터 처리 및 효율적인 관리가 보장된다면 비 정규화된 설계 구조도 하나의 설계 방법이 될 수 있다는 것입니다.
- **조인하지 않습니다.**
  - 중첩 데이터 구조를 설계 할 수 있기 때문에 불필요한 조인을 최소화 시킬 수 있다.
- **N:M의 관계를 지원한다.**
  - 관계형 DB에서는 N:M 관계가 성립되지 않는다.
  - MongoDB에서는 허용
- **스키마 중심의 설계를 하지 않는다.**
  - 관계형 DB에서는 하나의 테이블을 생성하며 테이블명은 사용자 계정명과 함께 결합되어 사용된다. 즉, 관계형 DB에서의 사용자 계정은 DBMS에 접속할 수 있는 권한이 있는지 없는지 결정하는 인증의 의미와 객체 이름을 결정하는 중요한 의미도 있음
  - MongoDB에서는 사용자 계정은 오직 인증의 의미만 가지고 있기 때문에 스키마를 고려한 설계가 필요 없다.

# MongoDB를 위한 데이터 모델링

- 설계 기준
  - 데이터 조작은 어떻게 수행하는가?
  - Access Patterns은 어떤가?
  - 스키마 설계 시 고려사항은?
- 설계 패턴
  - embedded document
    - 내장형 도큐먼트 구조
    - 주문 정보를 저장하고 그 배열 안에는 주문 상세 document를 저장
  - 실습

```
db.ord.insert( {ord_id : "2013-09-03", custmor_name : "wonam" , emp_name : "magee", total : "601100", payment_type : "Credit", order_filled : "y", item_id : [{item_id : "1", product_name : "bunny", item_price : "135", qty : "500", price: "67000"}, { item_id : "2", product_name : "pro ski", item_price : "380", qty: "400", price : "152000"}] })
```

# MongoDB를 위한 데이터 모델링

- Extent Document
  - 확장형 도큐먼트 구조
  - 주문 정보를 입력하고, 추가로 주문 상세 정보를 update를 이용하여 추가 시키는 방법
- 실습

```
db.ord.insert( {ord_id : "2013-09-03", custmor_name : "wonam" , emp_name : "magee", total : "601100", payment_type : "Credit", order_filled : "y"})
```

```
db.ord.update({ord_id : "2013-09-03"}, {$set : { item_id : [{ item_id : "1", product_name : "bunny", item_price : "135", qty : "500", price: "67000"}, { item_id : "2", product_name : "pro ski", item_price : "380", qty: "400", price : "152000"} ] } })
```

- 이 두 가지 구조를 Rich Document 구조라고 한다.

# MongoDB를 위한 데이터 모델링

- **Rich Document** 장/단점
  - 장점
    - 쿼리가 단순
    - 조인문 필요 없음
    - 데이터 보안에 효과적
  - 단점
    - 최대 16MB 범위 내에서 가능
    - Embedded 되는 도큐먼트가 존재하지 않는 컬렉션에는 적합하지 않음

# MongoDB를 위한 데이터 모델링

- **Linking**

- 관계형 DB의 relationship 과 유사한 구조
- 단, MongoDB에서는 연결 고리가 ObjectId로 설정되어 논리적인 부분에서 유사하지만, 물리적으로 다른 데이터 구조를 가지고 있음
- 실습

```
db.ord.insert( {ord_id : "2013-09-03", custmor_name : "wonam" , emp_name : "magee", total : "601100", payment_type : "Credit", order_filled : "y",})
```

```
o=db.ord.findOne({ "ord_id" : "2013-09-03"}) // Object_id 정보 추출
```

```
db.ord_detail.insert({ord_id : "2013-09-03", item_id : [{ item_id : "1", product_name : "bunny", item_price : "135", qty : "500", price: "67000"}, { item_id : "2", product_name : "pro ski", item_price : "380", qty: "400", price : "152000"} ], ordid_id : ObjectId("522823c175fb365790e312a1")})
```

```
db.ord_detail.findOne({ordid_id: o._id})
```

```
db.ord.drop()
```

# MongoDB를 위한 데이터 모델링

- DBRef 함수를 이용한 Linking
  - 실습

```
x= {ord_id : "2013-09-03", custmor_name : "wonam" , emp_name : "magee",  
total : "601100", payment_type : "Credit", order_filled : "y"}
```

```
db.ord.save(x)
```

```
db.ord.find()
```

```
db.ord_detail.save({ord_id : "2013-09-03", item_id : [{ item_id : "1",  
product_name : "bunny", item_price : "135", qty : "500", price: "67000"}, {  
item_id : "2", product_name : "pro ski", item_price : "380", qty: "400", price :  
"152000"} ], ordid_id :[ new DBRef ("ord", x._id)] })
```

```
db.ord_detail.find()
```

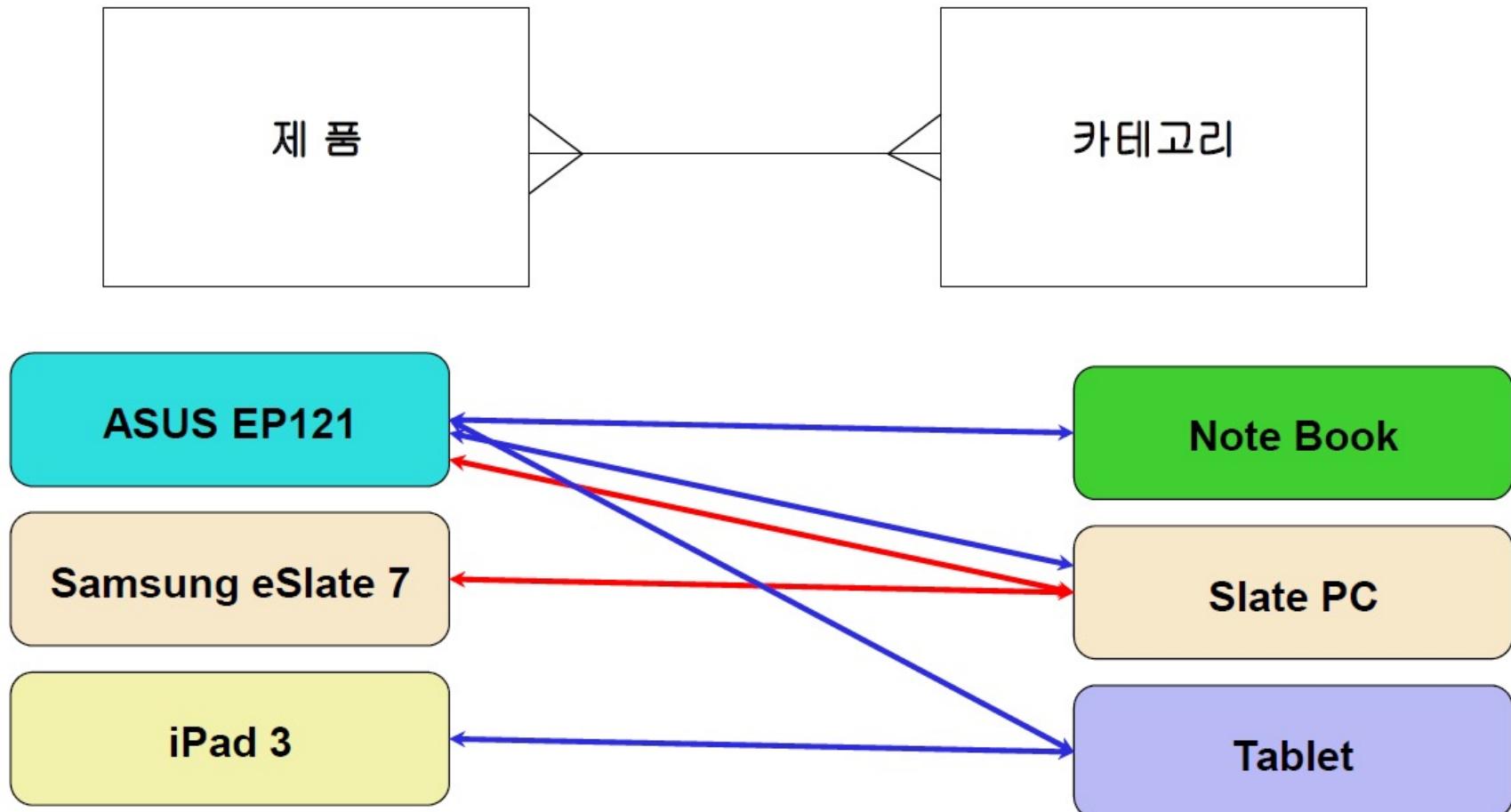
```
db.ord.drop()
```

# MongoDB를 위한 데이터 모델링

- **Linking** 장/단점
  - 장점
    - 별도의 논리적 구조로 저장되기 때문에 도큐먼트 크기 제한 없음
    - 비즈니스 툴 상 별도로 처리되는 데이터 구조에 적합
  - 단점
    - 매번 논리적 구조 간에 Linking 해야 하기 때문에 Embedded 보다 성능 떨어짐
    - 컬렉션 개수가 증가하며 관리 비용이 많이 듈다.

# MongoDB를 위한 데이터 모델링

- N:M 패턴



# MongoDB를 위한 데이터 모델링

- N:M 패턴
  - 실습

```
db.category.insert({ "cname" : "note Book ", "pname1" : "Asus EP121 M50 "})
```

```
db.category.insert({ "cname" : "Tablet ", "pname1" : "Asus EP121 M50 ", "pname2" : "ipad3"})
```

```
db.category.insert({ "cname" : "SlatePC ", "pname1" : "Asus EP121 M50 ", "pname2" : "Samsung eSlate7"})
```

```
db.category.find()
```

```
db.product.insert({ "pname" : "Asus EP121 ", "cname1" : "note Book ", "cname2" : "Tablet ", "cname3" : "SlatePC "})
```

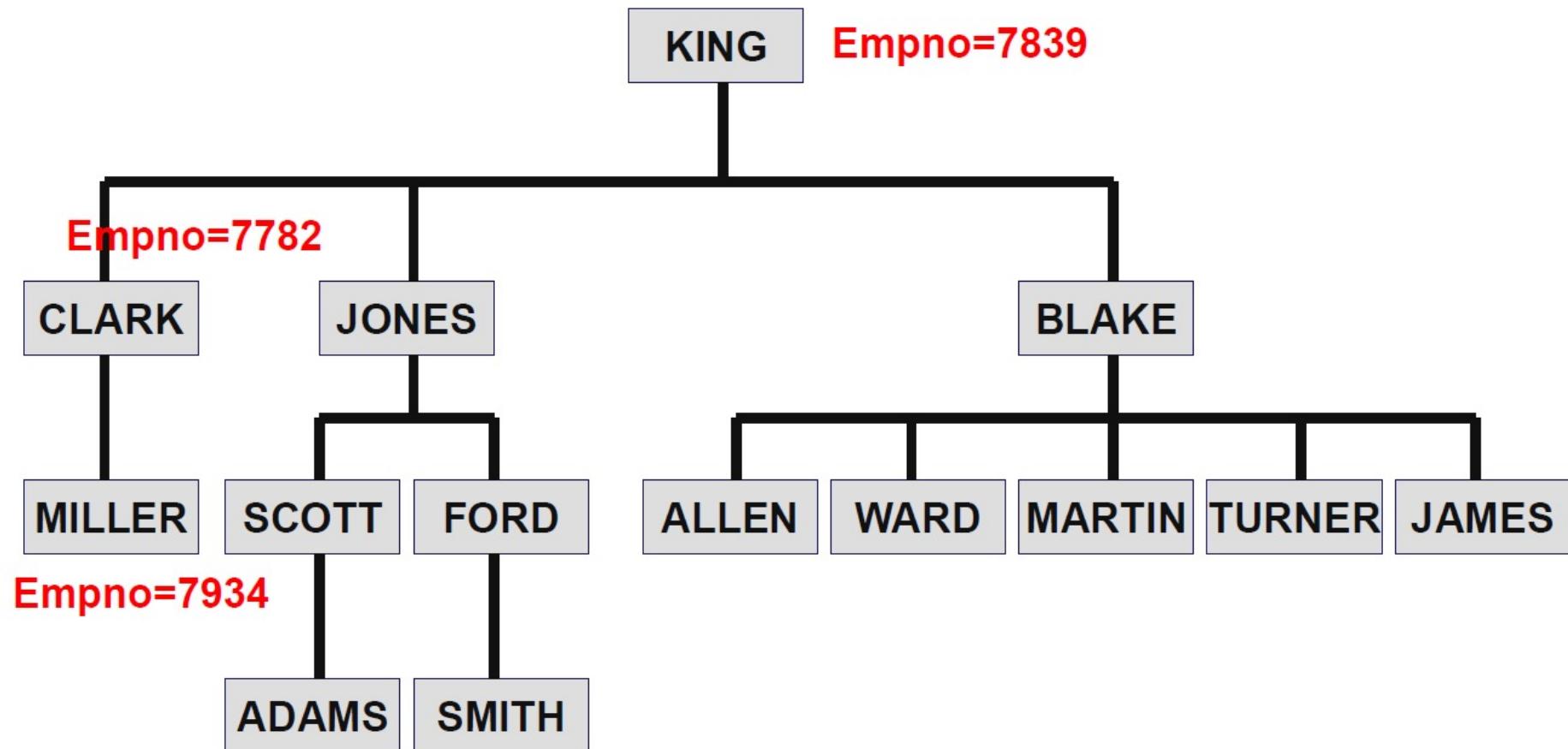
```
db.product.insert({ "pname" : "Samsung eSlate7 ", "cname1" : "SlatePC " })
```

```
db.product.insert({ "pname" : "ipad3 ", "cname1" : "Tablet " })
```

```
db.product.find()
```

# MongoDB를 위한 데이터 모델링

- 계층형 패턴



# MongoDB를 위한 데이터 모델링

- 계층형 패턴
  - 실습

```
db.employees.insert({ "_id" : "7839", "name" : "KING", "job" : "PRESIDENT" })
```

```
db.employees.insert({ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",  
"PARENT" : "7839" } )
```

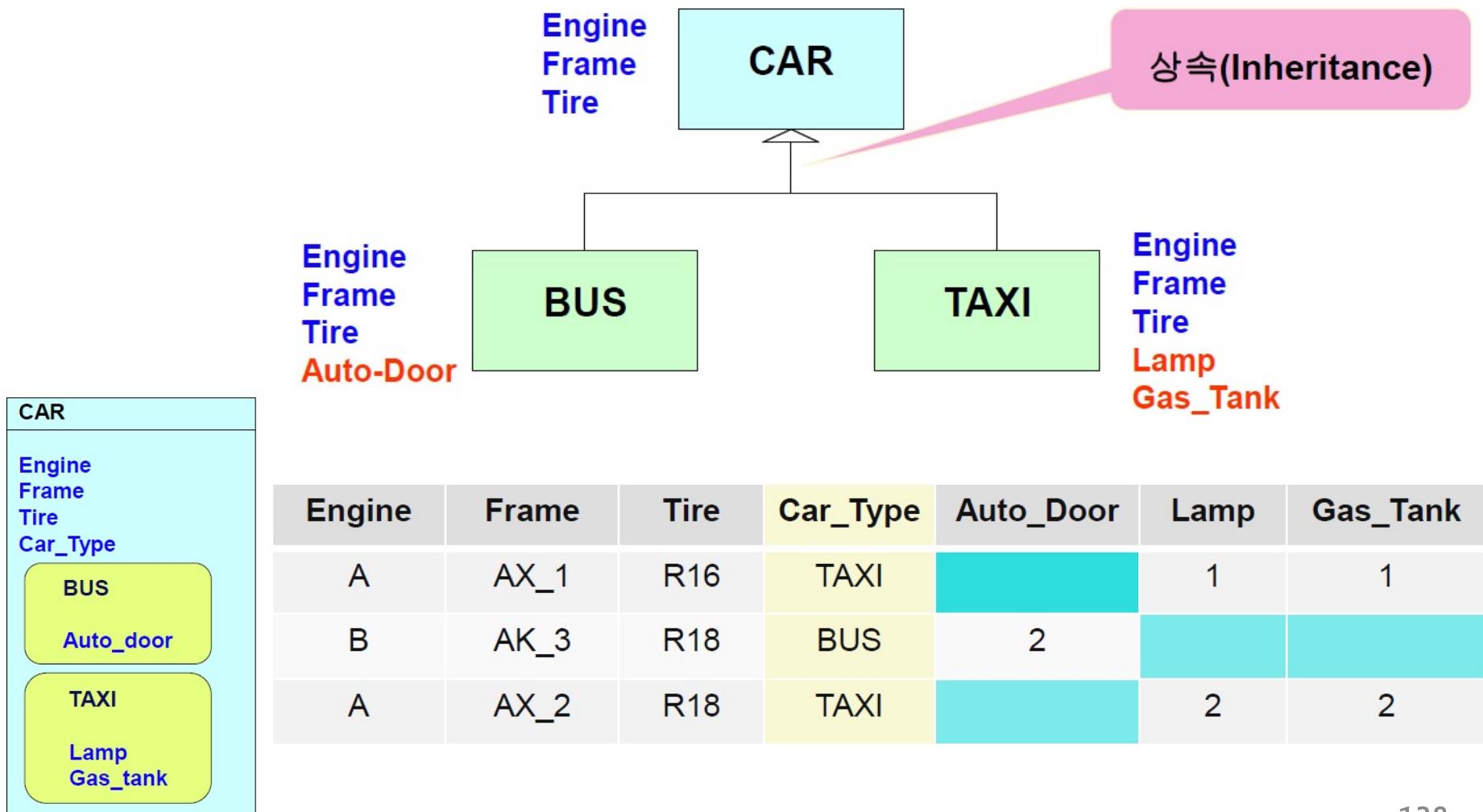
```
db.employees.insert({ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",  
"ANCESTORS" : "7839" , "PARENT" : "7782" } )
```

```
db.employees.find({"ANCESTORS" : "7839"})
```

```
db.employees.find({"PARENT" : "7839"})
```

# MongoDB를 위한 데이터 모델링

- 상속 패턴



# MongoDB를 위한 데이터 모델링

- 상속 패턴
  - 실습

```
db.createCollection ("car");
```

```
db.car.insert({ engine : "A", frame : "AX_1", tire : "R16", car_type : "TAXI", lamp : 1, gas_tank : 1 });
```

```
db.car.insert({ engine : "B", frame : "AK_3", tire : "R18", car_type : "BUS", auto_door: 2 });
```

```
db.car.insert({ engine : "A", frame : "AX_2", tire : "R18", car_type : "TAXI", lamp : 2, gas_tank : 2 });
```

Engine: A	Frame: AX_1	Tire: R16	Car_type: TAXI	Lamp: 1	Gas_tank: 1
Engine: B	Frame: AK_3	Tire: R18	Car_type: BUS	Auto_door: 1	
Engine: A	Frame: AX_1	Tire: R18	Car_type: TAXI	Lamp: 2	Gas_tank: 2

# 12

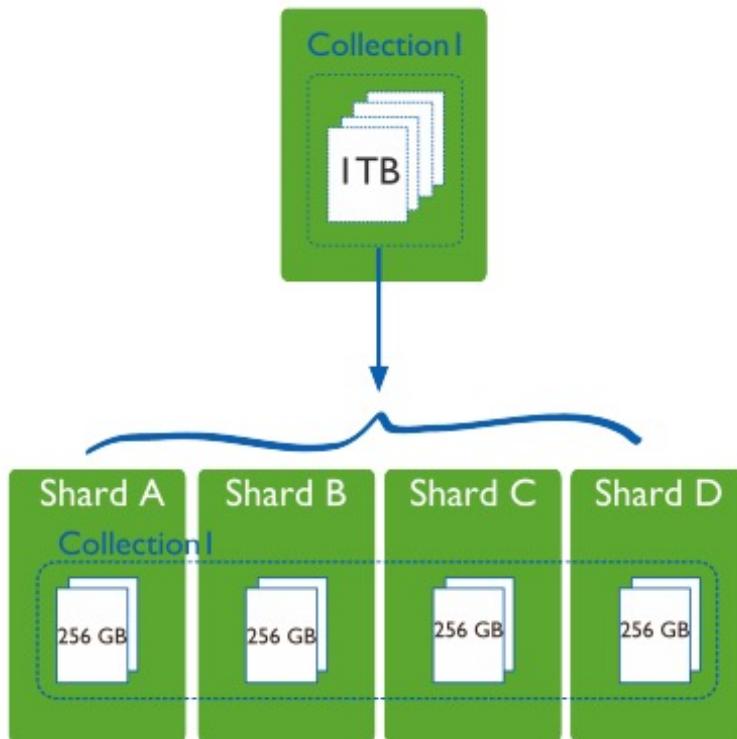
## **mongoDB의 샤딩시스템**

# MongoDB의 Sharding System

- **Sharding System**

- 빅 데이터 환경에서는 초당 몇 만 건 이상 되는 수많은 데이터를 빠른 시간 내에 처리해야 하며, 때로는 분산, 집계하여 사용자가 원하는 정보로 만들 수 있어야 한다.
- 이러한 시스템에서 효과적인 분산 저장 및 처리 기술이 필요한데 이것을 샤딩 시스템이라고 한다.

1개의 컬렉션과 4개의 샤드 서버로 구성



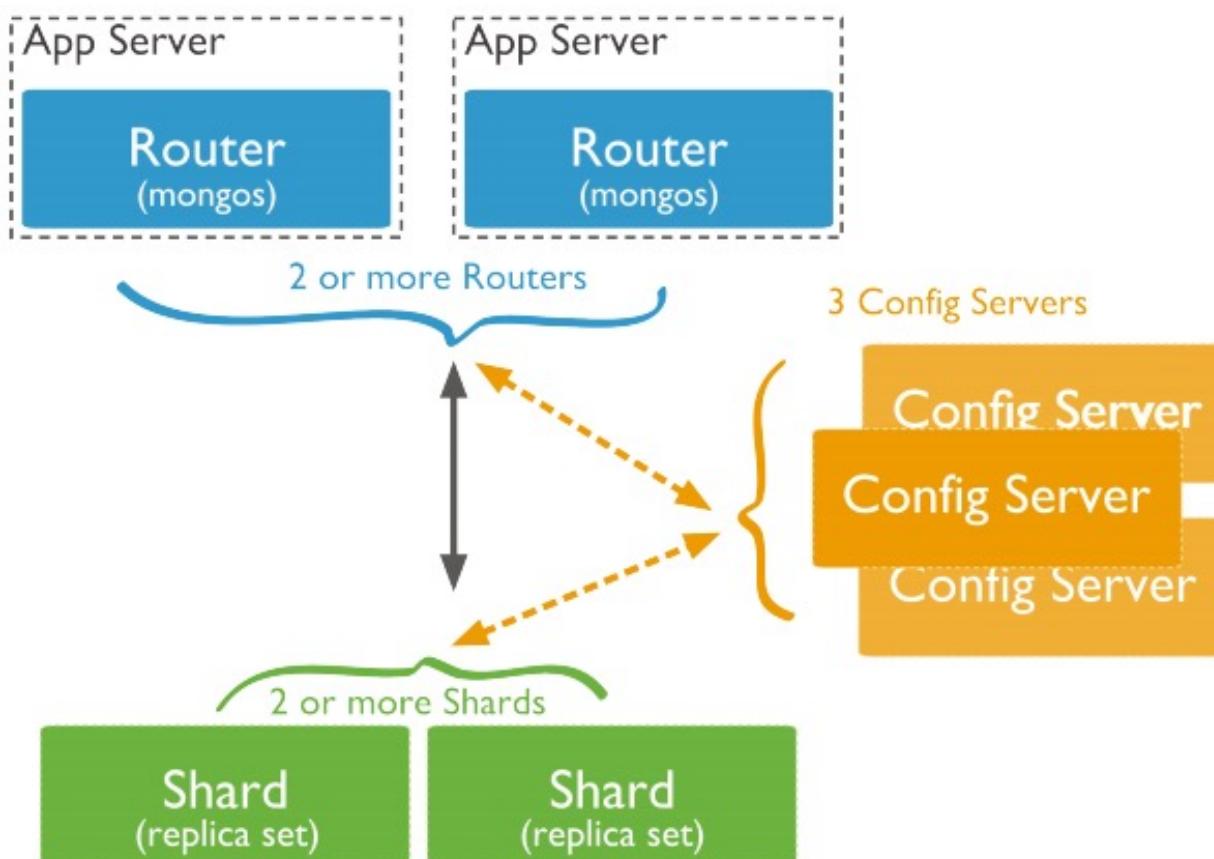
# MongoDB의 Sharding System

- **샤딩의 목적**
  - 데이터의 분산 저장
    - 빅 데이터의 효과적인 저장과 관리를 위해서는 여러 대의 서버가 필요
  - 빠른 성능
    - 여러 개의 CPU를 통해 동시 작업을 수행 했을 때 가장 이상적인 성능을 갖는다.
    - 샤딩 시스템은 분산처리 기술을 통한 빠른 성능을 보장 할 수 있는 기능이다.
  - 데이터의 백업과 복구
    - 하나의 서버에 빅 데이터를 저장하였다가 오류가 발생했을 경우 리스크는 상상의 초월할 것이다.
    - 이러한 것을 대비하기 위해 안전하게 데이터를 저장, 관리하기 위한 목적으로 샤딩 시스템을 사용한다.

# MongoDB의 Sharding System

- 샤딩 시스템의 구축

- 샤드 클러스터의 구성은 샤드 서버, Router, Config 서버로 구성된다.



# MongoDB의 Sharding System

- **샤딩 시스템의 구축**
  - 샤드 클러스터의 구성은 샤드 서버, Router, Config 서버로 구성된다.
    - 샤드 서버는 데이터를 저장한다. 샤드 클러스터의 고가용성 및 데이터의 일관성을 위해 각 샤드 서버는 리프리카셋으로 구성한다.
    - Router 또는 Mongos 인스턴스는 샤드 서버를 위해 클라이언트 응용 프로그램에서 직접 작업 할 수 있는 인터페이스를 제공한다.
    - 샤드 클러스터는 클라이언트 요청 로드를 분할하는 하나 또는 하나 이상의 Router를 가질 수 있다.
    - 클라이언트는 하나의 Router에 요청을 한다. 샤드 클러스터에는 여러 개의 쿼리 Router가 존재한다.
    - config 서버는 클러스터의 메타 데이터를 저장한다. 이 데이터는 샤드 서버를 위한 클러스터의 데이터 셋의 매핑을 포함하고 있다. 쿼리 router는 특정 샤드 작업을 대상으로 이 메타 데이터를 사용합니다.
    - 샤드 클러스터는 정확히 3개의 config 서버가 필요하다.

# MongoDB의 Sharding System

- **mongoDB에서의 샤딩 구축을 위한 적절한 시스템 환경**
  - 데이터 분산 효율성을 위해 2대 또는 이상의 샤드 서버로 구축한다.
  - 싱글 노드를 운영할 때 요구되는 메모리 영역보다 최소 20~30% 이상의 추가 메모리 영역이 요구된다. (mongos와 oplog 그리고, balancer 프로세스를 고려)
  - 샤드 시스템을 구축하는 때 요구되는 config 서버는 최소 3대 이상이 활성화할 것을 권장합니다. config 서버는 샤드 시스템 구축과 관련된 메타 데이터가 저장 관리하며 빅 데이터의 빠른 검색을 위한 인덱스의 정보를 저장, 관리하기 때문에 샤드 서버와는 별도의 서버로 구성해야 한다. (최대 1대 이상의 config 서버가 요구되며, 장애 발생으로 인해 정상 작동되지 않는 경우를 위해 추가 config 서버가 필요하다. 샤드 서버보다 저 사양으로 구성이 가능하다.)

# MongoDB의 Sharding System

- **샤딩 시스템 만들기**

- 물리적 서버(노드)를 최소 3대 이상이 필요하지만, 원활한 실습을 위해서 1개의 서버에서 환경 설정한다.
- 같은 서버(노드)에서 활성화하는 경우에는 반드시 다른 port 번호를 사용해야 한다.

Node 1의 Host명 : 자신의 컴퓨터명, DB 생성 홈 경로 : D:\MONGODB\SHARD1

Node 2의 Host명 : 자신의 컴퓨터명, DB 생성 홈 경로 : D:\MONGODB\SHARD2

Node 3의 Host명 : 자신의 컴퓨터명, DB 생성 홈 경로 : D:\MONGODB\SHARD3

SHARD 를 위한 CONFIG 서버 : 자신의 컴퓨터명, DB 생성 홈 경로 :  
D:\MONGODB\CONFIG1

<참고> 하나의 서버에서 실습할 때에는 "localhost"로 표시해도 무방하다.

# MongoDB의 Sharding System

- 샤딩 시스템 만들기
  - 각 호스트의 데이터베이스가 생성될 홈 경로를 설정한다.

```
D:\mongodb>mkdir shard1  
D:\mongodb>mkdir shard2  
D:\mongodb>mkdir shard3  
D:\mongodb>mkdir config1
```

```
D:\mongodb> mkdir shard1  
D:\mongodb> mkdir shard2  
D:\mongodb> mkdir shard3  
D:\mongodb> mkdir config1
```

# MongoDB의 Sharding System

- 샤딩 시스템 만들기

- 노드 1을 위한 샤드 서버를 활성화한다.

```
D:\mongodb>mongod --shardsvr --dbpath d:\mongodb\shard1 --port 40001
Sat Oct 12 20:12:27.968
Sat Oct 12 20:12:27.968 warning: 32-bit servers don't have journaling enabled by
default. Please use --journal if you want durability.
Sat Oct 12 20:12:27.968
Sat Oct 12 20:12:28.046 [initandlisten] MongoDB starting : pid=2780 port=40001 d
bpath=d:\mongodb\shard1 32-bit host=mainpc
Sat Oct 12 20:12:28.046 [initandlisten]
Sat Oct 12 20:12:28.046 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Oct 12 20:12:28.046 [initandlisten] **           32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Sat Oct 12 20:12:28.046 [initandlisten] **           Note that journaling defaults t
o off for 32 bit and is currently off.
Sat Oct 12 20:12:28.046 [initandlisten] **           See http://dochub.mongodb.org/c
```

```
mongod --shardsvr --dbpath d:\mongodb\shard1 --port 40001
```

# MongoDB의 Sharding System

- **샤딩 시스템 만들기**

- 노드 2을 위한 샤드 서버를 활성화한다. 별도의 콘솔창을 사용하여야 한다.

```
D:\mongodb>mongod --shardsvr --dbpath d:\mongodb\shard2 --port 40002
Sat Oct 12 20:15:28.093
Sat Oct 12 20:15:28.093 warning: 32-bit servers don't have journaling enabled by
default. Please use --journal if you want durability.
Sat Oct 12 20:15:28.093
Sat Oct 12 20:15:28.125 [initandlisten] MongoDB starting : pid=2008 port=40002 c
opath=d:\mongodb\shard2 32-bit host=mainpc
Sat Oct 12 20:15:28.125 [initandlisten]
Sat Oct 12 20:15:28.125 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Oct 12 20:15:28.125 [initandlisten] **          32 bit builds are limited to 1e
```

```
mongod --shardsvr --dbpath d:\mongodb\shard2 --port 40002
```

# MongoDB의 Sharding System

- **mongoDB에서의 config 서버**
  - 3개의 mongod로 구성할 것을 권장한다. (최소 1개)
  - config1의 메타 데이터는 config2, config3에 동일하게 저장 관리된다.
  - 만약 하나의 config Server가 다운되면 나머지는 Read only가 된다.
  - 샤프에 대한 메타 데이터 정보를 가지고 있다. mongos가 데이터를 쓰고 읽고 할 때에는 config 서버를 통해 수집한다.

# MongoDB의 Sharding System

- config 서버 만들기

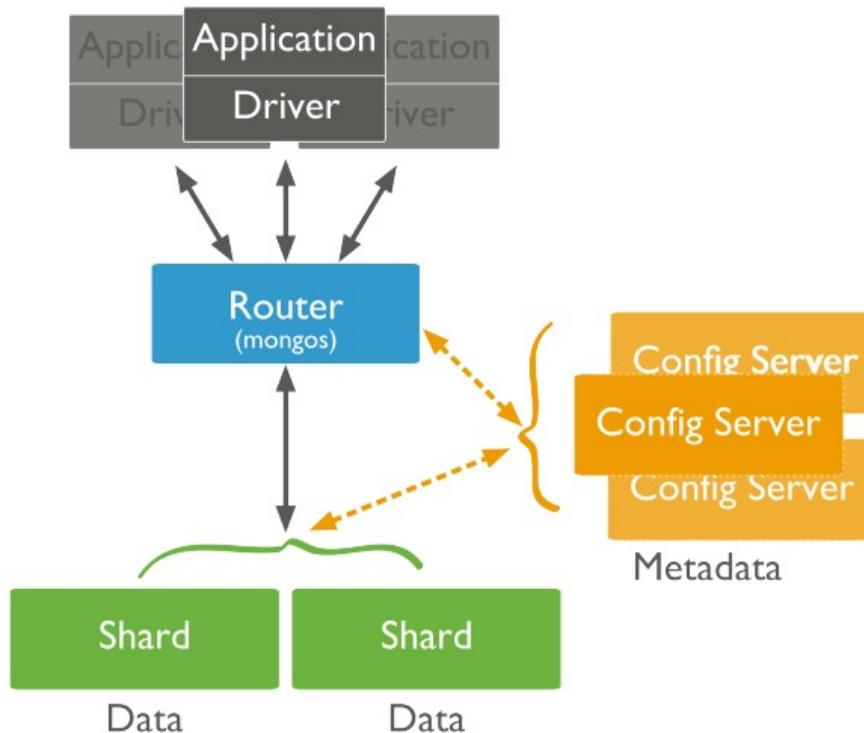
```
D:\mongodb>mongod --configsvr --dbpath d:\mongodb\config1 --port 50001
Sat Oct 12 20:35:30.296
Sat Oct 12 20:35:30.296 warning: 32-bit servers don't have journaling enabled by
default. Please use --journal if you want durability.
Sat Oct 12 20:35:30.296
Sat Oct 12 20:35:30.343 [initandlisten] MongoDB starting : pid=1396 port=50001 d
bpath=d:\mongodb\config1 master=1 32-bit host=mainpc
Sat Oct 12 20:35:30.343 [initandlisten]
Sat Oct 12 20:35:30.343 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Oct 12 20:35:30.343 [initandlisten] **          32 bit builds are limited to 1e
```

```
mongod --configsvr --dbpath d:\mongodb\config3 --port 50001
```

# MongoDB의 Sharding System

- **Router(mongos) 프로세서**

- 빅데이터를 샤드 서버로 분배해 주는 프로세서이다.
- 하나 이상의 프로세스가 활성화 된다.
- Application server에서 실행 가능하다.
- config 서버로부터 메타데이터를 캐시한다.



# MongoDB의 Sharding System

- Router(mongos) 프로세서 만들기
  - config 서버가 설치된 노드에서 각 노드가 상호 연결될 수 있도록 라우터를 등록해야 한다.

```
D:\mongodb>mongos --configdb localhost:50001 --port 50000 --chunksize 1
Sat Oct 12 21:00:43.468 warning: running with 1 config server should be done only for testing purposes and is not recommended for production
Sat Oct 12 21:00:43.531 [mongosMain] Mongos version 2.4.5 starting: pid=2748 port=50000 32-bit host=mainpc (--help for usage)
Sat Oct 12 21:00:43.531 [mongosMain] git version: a2ddc68ba7c9cee17bfe69ed840383ec3506602b
Sat Oct 12 21:00:43.531 [mongosMain] build info: windows sys.getwindowsversion(major=6, minor=0, build=6002, platform=2, service_pack='Service Pack 2') BOOST_LIB_VERSION=1_49
Sat Oct 12 21:00:43.531 [mongosMain] options: { chunksize: 1, configdb: "localhost:50001", port: 50000 }
Sat Oct 12 21:00:43.796 [LockPinger] creating distributed lock ping thread for 1
```

**mongos --configdb 192.168.0.10:50001(또는 localhost:50001) --port 50000 --chunkSize 1**

# MongoDB의 Sharding System

- MongoDB에서 Client 사용
  - 라우터 MongoS 프로세서로 접속하여 샤드 서버 등록 (mongos 프로세서의 기본 port는 27017)

```
D:\mongodb>mongo localhost:50000/admin
MongoDB shell version: 2.4.5
connecting to: localhost:50000/admin
mongos> -
```

```
mongo localhost:50000/admin
```

# MongoDB의 Sharding System

- MongoDB에서 Client 사용
  - 샤드 서버 등록

```
mongos> db.runCommand({addshard : "localhost:40001"})
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos> db.runCommand({addshard : "localhost:40002"})
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos> db.runCommand({addshard : "localhost:40003"})
{ "shardAdded" : "shard0002", "ok" : 1 }
```

```
db.runCommand({addshard : "localhost:40001"})
db.runCommand({addshard : "localhost:40002"})
db.runCommand({addshard : "localhost:40003"})
```

# MongoDB의 Sharding System

- MongoDB에서 Client 사용
  - 해당 DB의 shard 기능 추가 및 활성화

```
mongos>
mongos> db.runCommand({enablesharding : "test"})
{ "ok" : 1 }
mongos>
```

**db.runCommand({enablesharding : "test"})**

<참고> MongoDB의 샤딩은 컬렉션 단위로 수행되며, 해당 컬렉션의 샤드키(특정 필드) 값을 기준으로 분산된다. 샤딩을 위해 해당 샤드키에는 반드시 인덱스의 생성이 요구된다.

# MongoDB의 Sharding System

- MongoDB에서 Client 사용
  - ex) test DB에서 Things 컬렉션의 empno 색드키를 기준으로 shard 기능이 활성화

```
mongos> use test
switched to db test
mongos> db.things.ensureIndex({empno:1})
mongos>
mongos> use admin
switched to db admin
mongos>
mongos> db.runCommand({shardcollection : "test.things", key : {empno:1}})
{ "collectionsharded" : "test.things", "ok" : 1 }
mongos>
```

```
use test
db.things.ensureIndex({empno:1})
use admin
db.runCommand({shardcollection : "test.things", key : {empno:1}})
```

# MongoDB의 Sharding System

- **sharding system 테스트**
  - 환경 설정 확인
  - 현재 설정된 샤드 서버를 확인

```
mongos> db.runCommand({listshards:1})
{
  "shards" : [
    {
      "_id" : "shard0000",
      "host" : "localhost:40001"
    },
    {
      "_id" : "shard0001",
      "host" : "localhost:40002"
    },
    {
      "_id" : "shard0002",
      "host" : "localhost:40003"
    }
  ],
  "ok" : 1
}
mongos>
```

db.runCommand({listshards:1})

# MongoDB의 Sharding System

- **sharding system 테스트**
  - mongos 프로세스 상태 확인

```
mongos> use config
switched to db config
mongos> db.locks.find({_id : "balancer"})
{ "_id" : "balancer", "process" : "mainpc:50000:1381579243:41", "state" : 0, "ts"
" : ObjectId("52594516d588435739448bb7"), "when" : ISODate("2013-10-12T12:48:22.
140Z"), "who" : "mainpc:50000:1381579243:41:Balancer:41", "why" : "doing balance
round" }
mongos>
```

```
use config
db.locks.find({_id:"balancer"})
```

# MongoDB의 Sharding System

- **sharding system 테스트**

- version 컬렉션 수 확인

```
mongos> db.getCollection("version").findOne()
{
    "_id" : 1,
    "version" : 3,
    "minCompatibleVersion" : 3,
    "currentVersion" : 4,
    "clusterId" : ObjectId("525939ecd5884357394489d7")
}
mongos>
```

**db.getCollection("version").findOne()**

- 현재 설정된 chunk 크기 확인

```
mongos> db.settings.find()
{ "_id" : "chunksizes", "value" : 1 }
mongos>
```

**db.settings.find()**

# MongoDB의 Sharding System

- **sharding system 테스트**

- 샤드 서버 1 내용 확인

```
mongos> db.shards.findOne()
{ "_id" : "shard0000", "host" : "localhost:40001" }
mongos>
```

**db.shards.findOne()**

- mongos 상태 확인

```
mongos> db.collections.find()
{ "_id" : "test.things", "lastmod" : ISODate("1970-01-16T23:46:21.542Z"), "dropped" : false, "key" : { "empno" : 1 }, "unique" : false, "lastmodEpoch" : ObjectId("525942e6d588435739448b59") }
mongos>
mongos> db.mongos.findOne()
{
  "_id" : "mainpc:50000",
  "mongoVersion" : "2.4.5",
  "ping" : ISODate("2013-10-12T12:55:16.640Z"),
  "up" : 3272,
  "waiting" : true
}
mongos>
```

**db.collections.find()**  
**db.mongos.findOne()**

# MongoDB의 Sharding System

- **sharding system 테스트**
  - 실제 테스트
  - 이제 200,000건의 데이터를 입력하고 3개의 샤딩 시스템에 분산 입력되는 테스트를 한다.
  - router(mongos) 프로세서에 접속해서 데이터를 입력

```
D:\mongodb>mongo localhost:50000/test
MongoDB shell version: 2.4.5
connecting to: localhost:50000/test
mongos> for(var n= 100000; n <= 300000; n++) db.things.save({empno:n, ename:"test", sa1:1000})
mongos>
```

```
mongo localhost:50000/test
for(var n= 100000; n <= 300000; n++) db.things.save({empno:n, ename:"test", sa1:1000})
```

# MongoDB의 Sharding System

- **sharding system 테스트**

- 첫 번째 샤드 서버부터 세 번째 샤드 서버에 데이터가 어떻게 분산이 되었는지 확인 (첫 번째 shard 서버 : 총 200,000 건 중 74590개 저장)

```
D:\mongodb>mongo localhost:40001/test
MongoDB shell version: 2.4.5
connecting to: localhost:40001/test
Server has startup warnings:
Sat Oct 12 20:12:28.046 [initandlisten]
Sat Oct 12 20:12:28.046 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Oct 12 20:12:28.046 [initandlisten] **      32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Sat Oct 12 20:12:28.046 [initandlisten] **      Note that journaling defaults t
o off for 32 bit and is currently off.
Sat Oct 12 20:12:28.046 [initandlisten] **      See http://dochub.mongodb.org/c
ore/32bit
Sat Oct 12 20:12:28.046 [initandlisten]
Sat Oct 12 20:12:28.046 [initandlisten] ** NOTE: your operating system version d
oes not support the method that MongoDB
Sat Oct 12 20:12:28.046 [initandlisten] **      uses to detect impending page f
aults.
Sat Oct 12 20:12:28.046 [initandlisten] **      This may result in slower perfo
rmance for certain use cases
Sat Oct 12 20:12:28.046 [initandlisten]
> db.things.count()
74590
```

# MongoDB의 Sharding System

- **sharding system 테스트**
  - 첫 번째 샤드 서버부터 세 번째 샤드 서버에 데이터가 어떻게 분산이 되었는지 확인 (두 번째 shard 서버 : 총 200,000 건 중 74,646개 저장)

```
MongoDB shell version: 2.4.5
connecting to: localhost:40002/test
Server has startup warnings:
Thu Mar 13 15:46:05.234 [initandlisten]
Thu Mar 13 15:46:05.234 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Thu Mar 13 15:46:05.234 [initandlisten] **      32 bit builds are limited to less than 2GB of data (or less with --journal).
Thu Mar 13 15:46:05.234 [initandlisten] **      Note that journaling defaults to off for 32 bit and is currently off.
Thu Mar 13 15:46:05.234 [initandlisten] **      See http://dochub.mongodb.org/core/32bit
Thu Mar 13 15:46:05.234 [initandlisten]
Thu Mar 13 15:46:05.234 [initandlisten] ** NOTE: your operating system version does not support the method that MongoDB uses to detect impending page faults.
Thu Mar 13 15:46:05.234 [initandlisten] **      This may result in slower performance for certain use cases
Thu Mar 13 15:46:05.234 [initandlisten]
> db.things.count()
74646
>
```

# MongoDB의 Sharding System

- **sharding system 테스트**

- 첫 번째 샤드 서버부터 세 번째 샤드 서버에 데이터가 어떻게 분산이 되었는지 확인 (세 번째 shard 서버 : 총 200,000건 중에 50765개가 저장)

```
D:\mongodb>mongo localhost:40003/test
MongoDB shell version: 2.4.5
connecting to: localhost:40003/test
Server has startup warnings:
Sat Oct 12 20:18:29.156 [initandlisten]
Sat Oct 12 20:18:29.156 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Oct 12 20:18:29.156 [initandlisten] **      32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Sat Oct 12 20:18:29.156 [initandlisten] **      Note that journaling defaults t
o off for 32 bit and is currently off.
Sat Oct 12 20:18:29.156 [initandlisten] **      See http://dochub.mongodb.org/c
ore/32bit
Sat Oct 12 20:18:29.156 [initandlisten]
Sat Oct 12 20:18:29.156 [initandlisten] ** NOTE: your operating system version d
oes not support the method that MongoDB
Sat Oct 12 20:18:29.156 [initandlisten] **      uses to detect impending page f
aults.
Sat Oct 12 20:18:29.156 [initandlisten] **      This may result in slower perfo
rmance for certain use cases
Sat Oct 12 20:18:29.156 [initandlisten]
> db.things.count()
50765
```

# MongoDB의 Sharding System

- **sharding system 테스트**
  - 분산 배치된 상태 확인

```
D:\mongodb>mongo localhost:50000/test
MongoDB shell version: 2.4.5
connecting to: localhost:50000/test
mongos>
mongos> sh.status()
--- Sharding Status ---
sharding version: {
    "_id" : 1,
    "version" : 3,
    "minCompatibleVersion" : 3,
    "currentVersion" : 4,
    "clusterId" : ObjectId("525939ecd5884357394489d7")
}
shards:
[{"_id": "shard0000", "host": "localhost:40001"}, {"_id": "shard0001", "host": "localhost:40002"}, {"_id": "shard0002", "host": "localhost:40003"}]
databases:
[{"_id": "admin", "partitioned": false, "primary": "config"}, {"_id": "test", "partitioned": true, "primary": "shard0000"}]
    test.things
        shard key: { "empno" : 1 }
        chunks:
            shard0002      4
            shard0000      5
            shard0001      5
{ "empno" : { "$minKey" : 1 } } --> { "empno" : 100000
} on : shard0002 Timestamp(7, 0)
{ "empno" : 100000 } --> { "empno" : 107985 } on : shard0000 Timestamp(6, 1)
{ "empno" : 107985 } --> { "empno" : 124565 } on : shard0000 Timestamp(8, 0)
{ "empno" : 124565 } --> { "empno" : 141580 } on : shard0002 Timestamp(4, 1)
{ "empno" : 141580 } --> { "empno" : 158532 } on : shard0002 Timestamp(3, 4)
{ "empno" : 158532 } --> { "empno" : 175391 } on : shard0000 Timestamp(4, 2)
{ "empno" : 175391 } --> { "empno" : 192258 } on : shard0001 Timestamp(5, 1)
```

sh.status()

# MongoDB의 Sharding System

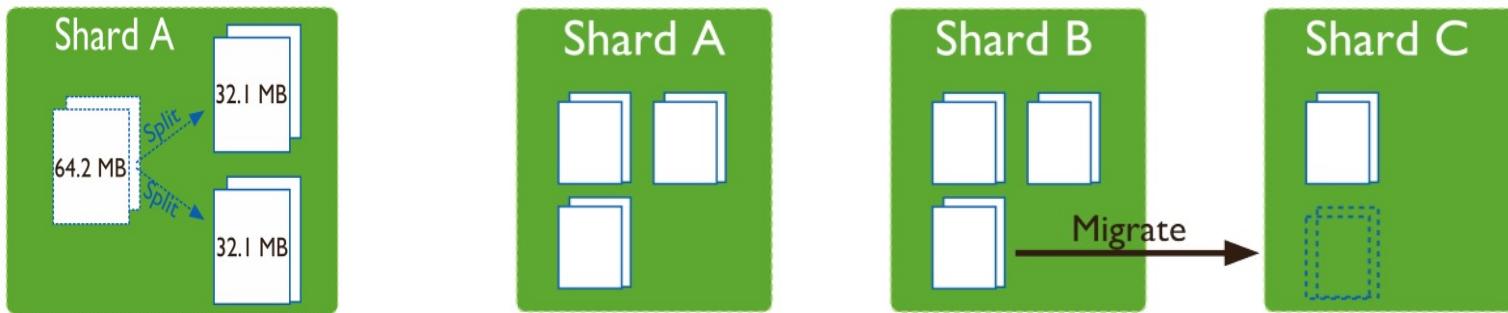
- **sharding 시스템 구축시 고려 사항**
  - shard key
    - 샤딩 시스템의 가장 중요한 요소 중에 하나가 shard key 이다.
    - 선택 필드 조건
      - 분할(partition)과 균등 분산(load balancing)을 위한 기준
      - 적절한 카디널리티(cardinality)를 가진 필드
      - 조건을 만족하는 데이터의 분산 정도가 넓으면 낮은 카디널리티라 하며, 분포가 좁으면 높은 카디널리티라고 하는데, 카디널리티가 너무 높거나 낮지 않아야 함.
      - 사용자에 의해 임의의 secondary key를 부여
    - ex) 사원번호는 고유의 값으로 구성되므로 높은 카디널리티를 가진 필드, 남/여 구분 필드는 2개의 값으로만 구성되므로 낮은 카디널리티를 가진 필드이므로,  
-> 해당 사원이 거주하는 지역구(광진구, 강남구, 강동구 등) 필드를 함께 샤드 키로 설정해서 적절한 분산이 되도록 선택한다.

# MongoDB의 Sharding System

- **sharding 시스템 구축시 고려 사항**
  - shard key index
    - 샤드 키는 반드시 하나의 인덱스를 생성해야 한다.
    - 여러 대의 서버에 데이터가 chunk 단위로 분산 저장되는 경우 데이터의 빠른 검색을 위해 필요하다.
    - 샤드 키 : 인덱스의 생성을 고려할 수 있는 필드, 빅데이터의 빠른 검색이 요구되는 필드
    - 좋은 카디널리티를 가진 필드를 샤드키와 샤드키 인덱스로 선정해야 한다.

# MongoDB의 Sharding System

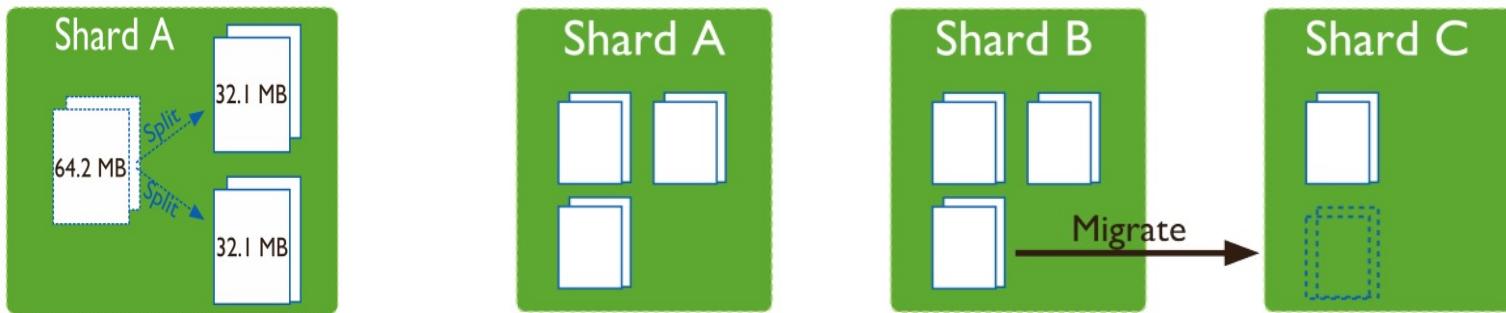
- sharding 시스템 구축시 고려 사항
  - chunk size & Migration 임계 값



- 청크(chunk)
  - 샤딩 시스템에서 데이터의 저장하는데 쓰이는 분할 단위
  - 기본 사이즈 : 64MB
  - 기본 사이즈는 shard key에 따라서 변화될 수 있다.

# MongoDB의 Sharding System

- sharding 시스템 구축시 고려 사항
  - chunk size & Migration 임계 값



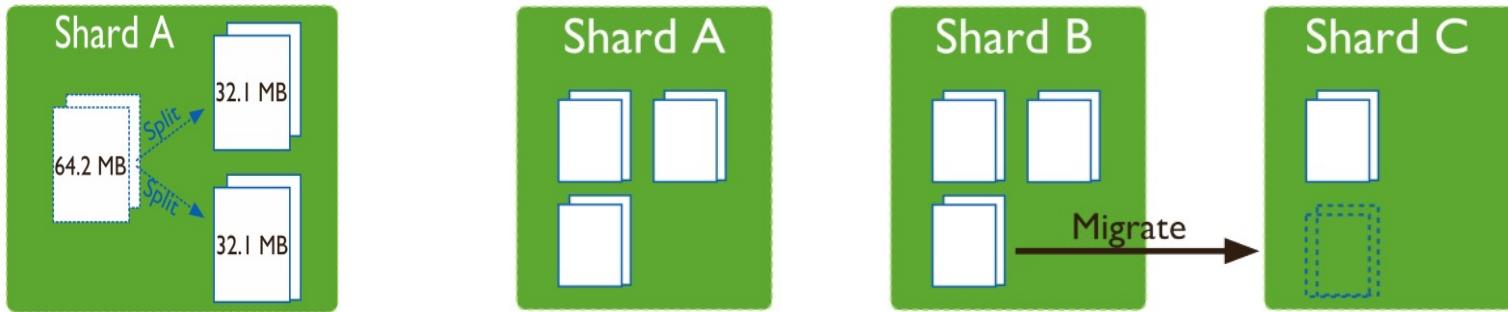
- chunk Migration 임계값

Number of Chunks	Migration Threshold
Less than 20	2
21-80	4
Greater than 80	8

- 나의 샤드 서버에 8개의 청크가 발생하면 다른 서버로 마이그레이션이 발생하는데 약 20개 미만의 청크가 발생하면 평균 2번 정도의 마이그레이션이 발생한다.
- 적절한 빈도의 마이그레이션의 발생시키면 샤딩 시스템의 효율성을 높일 수 있다.

# MongoDB의 Sharding System

- **sharding 시스템 구축시 고려 사항**
  - chunk size & Migration 임계 값



- 청크에 대한 이동은 다음 순서로 움직인다. (데이터 일관성을 유지, 청크의 가용성 최대화)
  - 밸런스 프로세스는 소스 샤드로 moveChunk 명령어를 전송
  - 소스 샤드는 moveChunk 명령을 수행
  - 목적 샤드는 청크에서 문서 요청을 수행하고, 데이터 복사본을 받음
  - 청크에서 마지막 문서를 받은 이후에 동기화를 수행
  - 동기화가 완전히 이루어지면 목적 샤드는 설정 데이터베이스에 연결하고, 클러스터의 메타데이터 갱신
  - 목적 샤드가 메타데이터 갱신을 완료한 이후에, 문서의 복사본은 삭제 처리 (복제 동작 일시 정지시킴)

# MongoDB의 Sharding System

- **sharding 시스템 구축시 고려 사항**
  - Shard 서버의 추가와 삭제
    - 샤딩 시스템의 큰 장점 중에 하나가 수평적인 확장이 용이하다는 점이다
    - 사용자의 필요에 따라 쉽게 샤드 서버를 추가하고 제거할 수 있다는 것이다.
  - 추가될 샤드 서버의 물리적 경로 설정 및 shard 서버(노드)의 활성화

```
D:\mongodb>mkdir shard4

D:\mongodb>mongod --shardsvr --dbpath d:\mongodb\shard4 --port 40004
Sat Oct 12 23:47:00.359
Sat Oct 12 23:47:00.359 warning: 32-bit servers don't have journaling enabled by
default. Please use --journal if you want durability.
Sat Oct 12 23:47:00.359
Sat Oct 12 23:47:00.406 [initandlisten] MongoDB starting : pid=5924 port=40004 d
bpath=d:\mongodb\shard4 32-bit host=mainpc
Sat Oct 12 23:47:00.406 [initandlisten]
Sat Oct 12 23:47:00.406 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Oct 12 23:47:00.406 [initandlisten] **      32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
```

```
D:\MongoDB>mkdir shard4
mongod --shardsvr --dbpath d:\mongodb\shard4 --port 40004
```

# MongoDB의 Sharding System

- sharding 시스템 구축시 고려 사항
  - 라우터(mongos)에 접속하여 4번째 샤드 서버를 추가

```
mongos> db.runCommand({addshard: "localhost:40004"})
{ "shardAdded" : "shard0003", "ok" : 1 }
mongos>
```

```
sh.addshard("<hostname>:<port>")
db.runCommand({addshard: <hostname>:<port>})
```

- 샤딩 시스템의 primary server 변경

```
mongos> db.runCommand({movePrimary:"test", to: "localhost:40002"})
{ "primary" : "shard0001:localhost:40002", "ok" : 1 }
mongos>
```

```
db.runCommand({movePrimary:"test", to: "localhost:40002"})
```

# MongoDB의 Sharding System

- sharding 시스템 구축시 고려 사항
  - 4번째 샤드 서버 제거

```
mongos> db.runCommand({removeshard:"localhost:40004"})
{
    "msg" : "draining started successfully",
    "state" : "started",
    "shard" : "shard0003",
    "ok" : 1
}
mongos>
```

db.runCommand({removeshard:<shard server명>})

# MongoDB의 Sharding System

- **Chunk 크기 관리**
  - 샤드 서버의 리스트를 확인
    - db.runCommand({listshards : 1})
  - 청크 사이즈 확인
    - db.setting.find()
  - 청크 사이즈 변경
    - db.setting.save({\_id : "cunuksiz", value : 128})
  - 변경된 청크 사이즈 확인
    - db.setting.find()
- **Chunk 크기 분리**
  - 하나의 청크 사이즈를 두 개로 분리하는 방법
  - Empno 필드의 값 20000을 기준으로 청크를 분리합니다.
    - sh.splitFind("test.tnings", {"empno" : 20000})
  - Empno 필드의 값 10005를 기준으로 기존의 청크 크기를 둘로 분리한다.
    - sh.splitAt("test.tnings", {"empno" : 10005})

# MongoDB의 Sharding System

- **Mongos(Balancer)** 프로세스가 샤드서버에 데이터를 분산하다 보면 작업의 일관성과 연속성을 위해 잠금 현상을 발생할 수 있다.
  - Use config
  - db.locks.find({ \_id : "balancer" }).pretty() // 밸런스의 잠금 상태 확인
  - sh.getBalancerState() // 작동여부 상태를 확인하는 방법
- **밸런스의 윈도우 설정**
  - 밸런스는 지속적으로 샤드 서버로 데이터를 분산 저장하게 된다.
  - 샤드키나 청크사이즈가 적절하지 않으면 마이그레이션을 불필요하게 일으켜 성능저하의 원인인 된다.
  - 오후 23:00 ~ 오전 06:00에서만 밸런스가 작동할 수 있도록 설정하는 법
    - db.setting.update({ \_id : "balancer" }, { \$set : { activeWindow : { start : "23:00" , stop : "06:00" } } }, true )
    - db.setting.update({ \_id : "balancer" }, { \$unset : { activeWindow : true } }) //밸런스 작동 설정
    - sh.stopBalancer() //밸런스 작동 중지

# MongoDB의 Sharding System

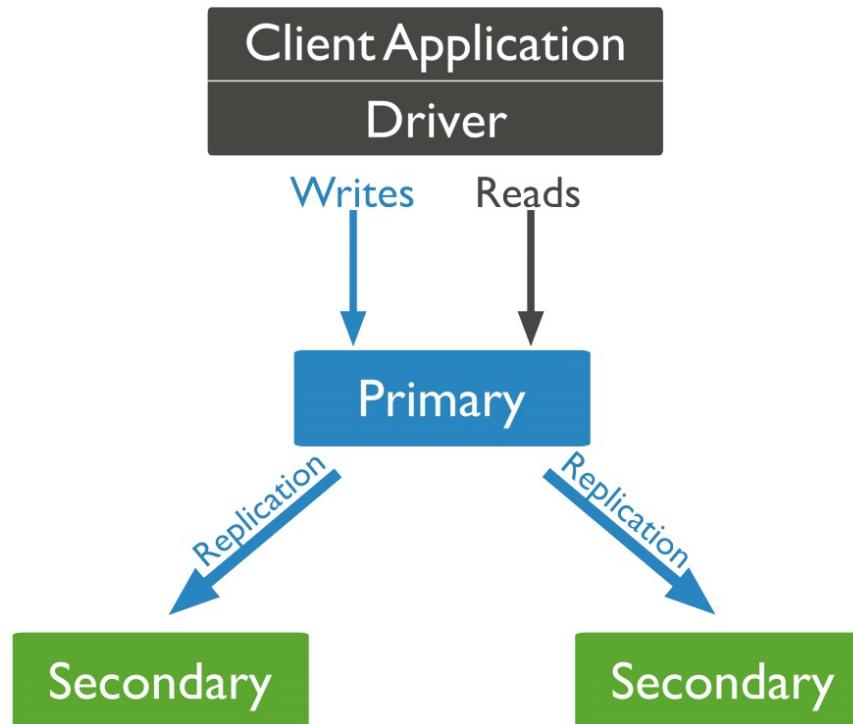
- **sharding 시스템 구축시 고려 사항**
  - 샤딩 시스템의 문제점
    - 하나의 샤드 서버에 데이터가 집중되고 균등 분산이 안 되는 경우
      - 샤드키의 잘못된 설정(카디널리티가 낮은 경우)
    - 특정 샤드 서버에 IO 트래픽이 증가하는 경우
      - 낮은 카디널리티가 설정되어 있는 경우
    - 하나의 클러스터의 밸런스가 균등하지 않을 때
      - 사용자의 특성에 따른 서버의 데이터의 삭제 또는 이동
      - 낮은 카디널리티 설정
      - 분산 처리되는 속도가 느린 경우
    - 과도한 chunk 마이그레이션이 클러스트 동작을 멈추어 있을 경우
    - 쓰기 성능이 지연되고 빠른 검색이 안 되는 경우
      - 부적절한 익스텐드의 크기 설정

**13**

## **mongoDB의 리프리카셋**

# Replica & ReplicaSets

- 빅 데이터 환경에서 예기치 못한 시스템 장애로 인한 데이터 유실은 기업 입장에서 치명적일 수 밖에 없는데 이러한 문제가 발생하더라도 빅데이터의 안전한 저장과 관리 그리고 복구가 수행되기 위한 적절한 백업 솔루션이 필요하다.
- 빅데이터의 백업을 통해 안정성을 보장하기 위한 솔루션 중 하나가 **Replica** 이다.
- Replication은 여러 서버를 통한 데이터의 동기화를 진행하는 솔루션이라고도 할 수 있다.



# Replica & ReplicaSets

- **Master & slave 서버**
  - Master 서버
    - 데이터를 저장하는 메인 서버 역할.
    - 클라이언트와의 Read/Write 가능
  - slave 서버
    - Master 서버와 동일한 구조를 가지고 있는 복제 서버
    - Master 서버에 장애가 발생하더라도 복제 서버를 이용하여 빠른 복구 가능
    - 예상치 못한 다양한 장애가 발생할 수 있으므로 최소 3대 정도의 slave server 설정을 권장.
  - 주의 사항
    - Replica 기능은 기본적은 최소 2개 이상의 Node로 구성되어야 한다.
    - Master 와 Slave 노드는 별도의 노드에 구축해야 한다.
    - 하나의 노드에 샤프 서버와 config 서버를 구축시에는 별도의 포트로 구분해야 한다.

# Replica & ReplicaSets

- Master & slave 서버
  - 시스템 환경
    - Master node : 포트번호 10000
    - slave1 node : 포트번호 10001
    - slave2 node : 포트번호 10002
  - Master & slave의 환경 설정
    - Master 데이터베이스가 설정될 흠 경로 설정

```
D:\mongodb>mongod --dbpath d:\mongodb\master --port 10000 --master
```

- slave1 과 slave2 데이터베이스에 대한 인스턴드를 만든다,

```
D:\mongodb>mongod --dbpath d:\mongodb\slave1 --port 10001 --slave --source localhost:10000
```

```
D:\mongodb>mongod --dbpath d:\mongodb\slave2 --port 10002 --slave --source localhost:10000
```

# Replica & ReplicaSets

- Master & slave 서버
  - replica 기능 테스트
    - master 서버로 접속

```
D:\#>mongo localhost:10000
```

- master 서버에 things collection 생성

```
> show dbs
> use test
switched to db test
> db.things.insert({empno:1101, ename:"tom", dept:"account"})
> db.things.find()
```

# Replica & ReplicaSets

- Master & slave 서버
  - replica 기능 테스트
    - slave1 서버의 DB에 접속

```
D:₩>mongo localhost:10001
```

- slave1 서버에 Replication 기능으로 생성된 DB 확인

```
> show dbs
local 0.03125GB
test 0.0625GB
> use test
switched to db test
> db.things.find()
{ "_id" : ObjectId("52639095f8d21e43df15eae9"), "empno" : 1101, "ename" : "tom",
"dept" : "account" }
```

# Replica & ReplicaSets

- Master & slave 서버
  - replica 기능 테스트
    - slave2 서버의 DB에 접속

D:₩>mongo localhost:10002

- slave2 서버에 Replication 기능으로 생성된 DB 확인

```
> show dbs
local 0.03125GB
test 0.0625GB
> use test
switched to db test
> db.things.find()
{ "_id" : ObjectId("52639095f8d21e43df15eae9"), "empno" : 1101, "ename" : "tom",
"dept" : "account" }
```

# Replica & ReplicaSets

- **ReplicaSets**

- master / slave 서버의 개념은 단순 복제 개념만을 가지고 있는 관계를 가지고 있으며, 장애가 발생했을 복제 본 데이터베이스를 통한 복구 작업을 할 수 있다.
- 하지만, 실시간으로 복구 작업을 수행하는 것이 아니며, 슬레이브 역시 즉시 사용할 수 있는 것도 아니다.
- 이러한 Replica의 단점을 보완한 것이 ReplicaSet 이다.
  - heartbeat : 매 2초마다 Secondary 상태를 체크한다.
  - Secondary 가 Down 되더라도 복제만 중지될 뿐 Primary에 대한 작업은 정상적이다.
  - Primary가 다운되면 Secondary가 Primary가 된다.
  - OpLog는 복제가 실패하는 경우를 위해 로그 정보를 저장해 준다(기본 크기 1GB)

# Replica & ReplicaSets

- **ReplicaSets**

- Primary 서버

- 실시간으로 사용되는 메인 서버
    - 데이터를 입력, 수정, 삭제, 조회하는 기본적인 작업 수행한다.
    - 문제점 : 복제를 하려면 일정시간동안 서비스를 중지해야 한다. 실시간으로 복구 작업을 진행 할 수 없다.

- Secondary 서버

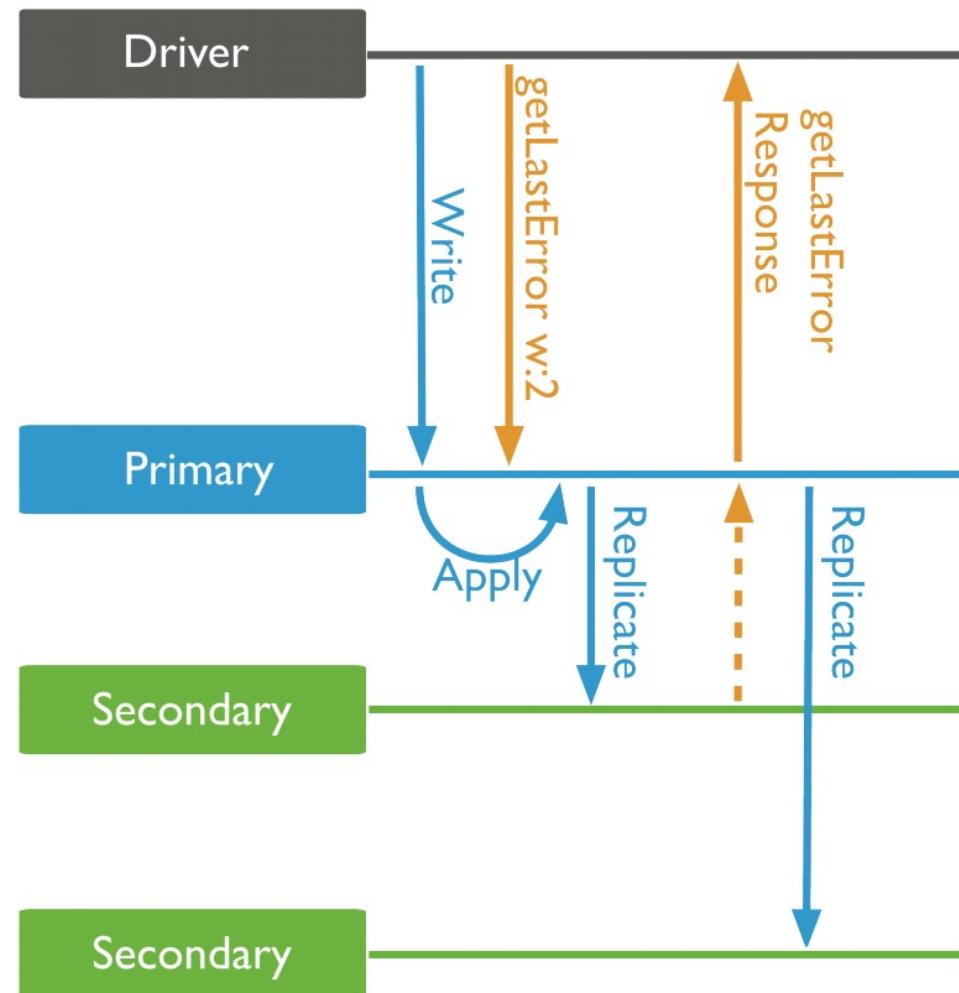
- Primary 서버가 장애 발생하여 서비스를 수행할 수 없는 경우에 즉시 프라이머리 서버의 마지막 수행 작업부터 연속적으로 작업을 수행하는 서버
    - 특징 : 장애 발생시 secondary 서버가 Primary 서버로 변동되고, 최초 설정된 Primary 서버는 복구 후 Secondary 서버가 된다.
    - secondary 서버도 만약을 대비해 Backup secondary를 설정해 놓는다.

# Replica & ReplicaSets

- ReplicaSets

- 작동원리

- ① Application 을 통해 Primary 서버로 Write 작업 시행
- ② Memory Mapped Cache Area에 데이터가 먼저 저장 완료
- ③ Primary 서버가 정상 작동이 안 되면 Get Last Error가 발생
- ④ Primary 서버의 Journal Area 로그 데이터를 Journal 파일에 백업
- ⑤ Primary server는 Application에게 장애가 발생했음을 알림
- ⑥ Primary server 는 Secondary server로 복제 작업을 수행



# Replica & ReplicaSets

- **ReplicaSets**

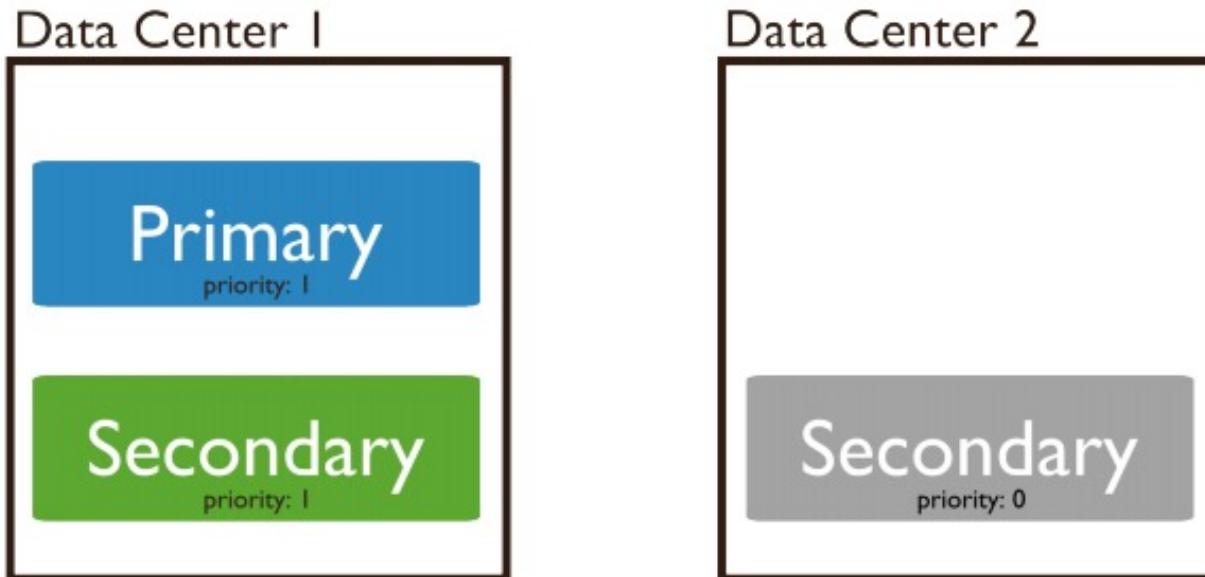
- Priority (우선 순위)

- Priority에 부여할 수 있는 값의 범위는 0 ~ 1,000이다.
    - Primary server에 장애가 발생하면 10초 내에 다음 Primary server가 되어야 하는 하나의 노드를 선택해야 한다.
    - 높은 값을 부여 받은 server가 Primary가 되기 위한 우선권을 부여받는다.

```
cfg=rs.conf()
cfg.members[0].priority = 0                         // 우선 순위 지정
cfg.members[1].priority = 0.5
cfg.members[2].priority = 1
cfg.members[3].priority = 2
rs.reconfig(cfg)
```

# Replica & ReplicaSets

- ReplicaSets 멤버의 유형
  - Secondary Only Member (Priority 0 Replica Set Members)
    - 사용자 데이터를 저장하고 있지만, 절대 프라이머리 서버가 될 수 없는 Only Secondary 서버를 의미한다.



# Replica & ReplicaSets

- **ReplicaSets 멤버의 유형**
  - Secondary Only Member (Priority 0 Replica Set Members)
    - 사용자 데이터를 저장하고 있지만, 절대 프라이머리 서버가 될 수 없는 Only Secondary 서버를 의미 한다.

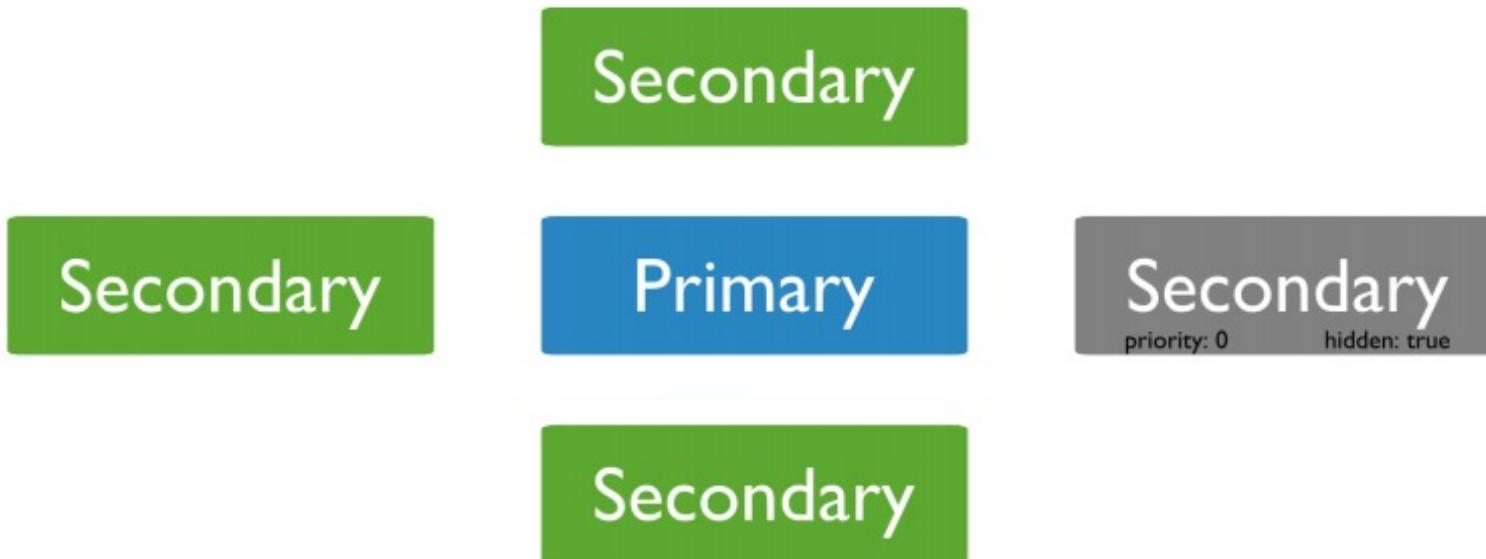
```
cfg=rs.conf()  
cfg.members[0].priority = 0 // Secondary Only Member 설정 방법  
cfg.members[1].priority = 0.5  
cfg.members[2].priority = 1  
cfg.members[3].priority = 2  
rs.reconfig(cfg)
```

# Replica & ReplicaSets

- **ReplicaSets 멤버의 유형**

- Hidden member

- Client Application을 위한 숨겨진 멤버. 아비터 서버가 프라이머리 서버를 선출하기 위해 투표시에는 사용되지 않는다.



# Replica & ReplicaSets

- **ReplicaSets 멤버의 유형**
  - Hidden member
    - Client Application을 위한 숨겨진 멤버. 아비터 서버가 프라이머리 서버를 선출하기 위해 투표시에는 사용되지 않는다.

```
cfg=rs.conf()  
cfg.members[0].priority = 0  
cfg.members[0].hidden = true  
rs.reconfig(cfg) // Hidden member 설정
```

# Replica & ReplicaSets

- ReplicaSets 멤버의 유형
  - Arbiter member
    - 사용자 데이터가 저장되지 않으며 오직 장애 발생시 Primary 서버를 선출하기 위해 투표시에만 사용된다.



# Replica & ReplicaSets

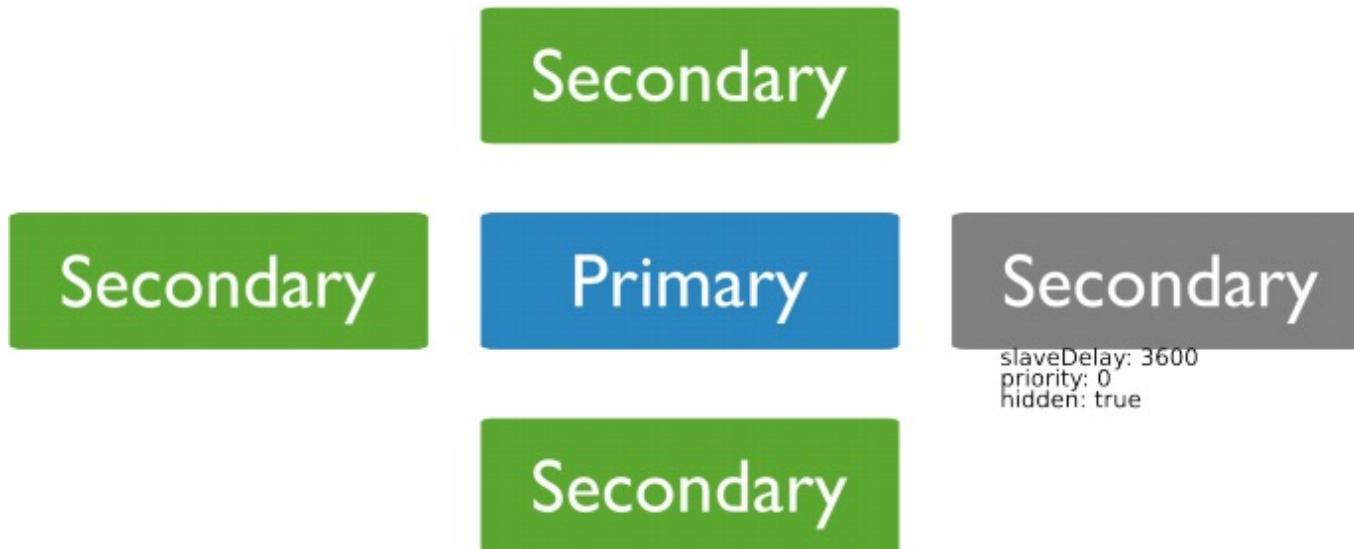
- **ReplicaSets 멤버의 유형**
  - Arbiter member
    - 사용자 데이터가 저장되지 않으며 오직 장애 발생시 Primary 서버를 선출하기 위해 투표시에만 사용된다.

```
db.runCommand({ "replSetInitiate" : {id" :"rptmongo", "members":  
[ {"_id" : 1, "host" : "localhost:10001"},  
{"_id" : 2, "host" : "localhost:10002"},  
{"_id" : 3, "host" : "localhost:10003", arbiterOnly : true}
```

//arbiter member 설정 방법

# Replica & ReplicaSets

- ReplicaSets 멤버의 유형
  - Delayed Member
    - 프라이머리 서버의 OpLog 정보를 정의된 시간 동안 Secondary 서버에 적용하지 않고 Delay 한 후 적용되는 멤버



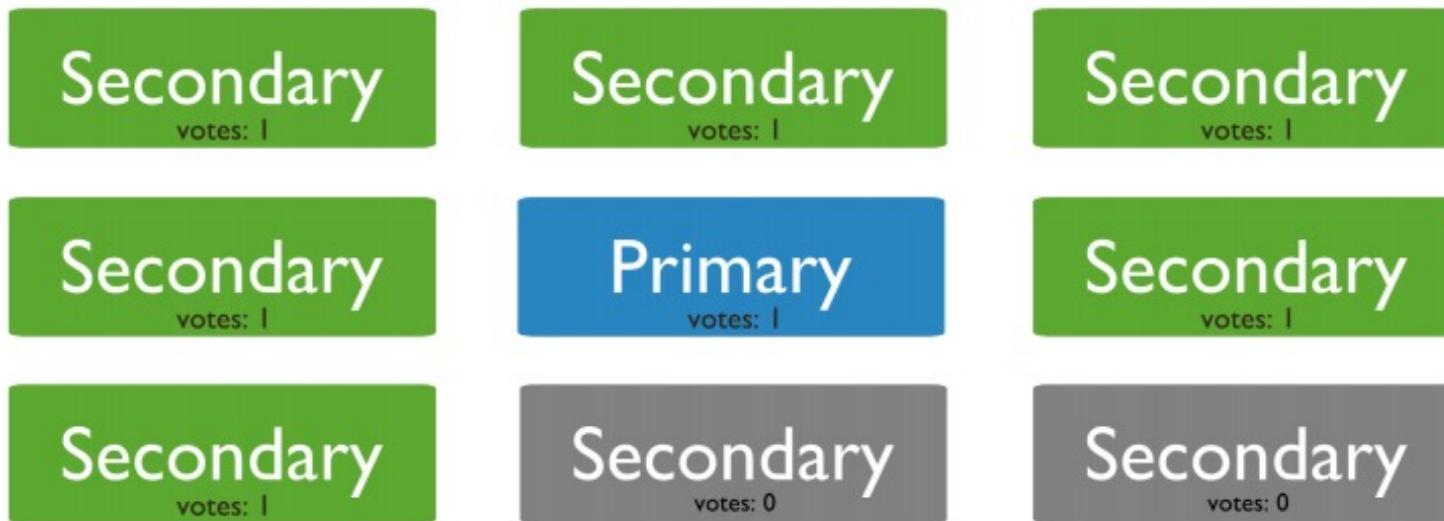
# Replica & ReplicaSets

- **ReplicaSets 멤버의 유형**
  - Delayed Member
    - 프라이머리 서버의 OpLog 정보를 정의된 시간 동안 Secondary 서버에 적용하지 않고 Delay 한 후 적용되는 멤버

```
cfg=rs.conf()
cfg.members[0].priority = 0
cfg.members[0].slaveDelay = 3600          // Delayed member 설정
rs.reconfig(cfg)
```

# Replica & ReplicaSets

- ReplicaSets 멤버의 유형
  - Non Voting member
    - 사용자 데이터를 저장하고 있지만 투표시에는 프라이머리 서버로 선택되지 않는 멤버이다.



# Replica & ReplicaSets

- **ReplicaSets 멤버의 유형**
  - Non Voting member
    - 사용자 데이터를 저장하고 있지만 투표시에는 프라이머리 서버로 선택되지 않는 멤버이다.

```
cfg = rs.conf()
cfg.members[3].votes = 0
cfg.members[4].votes = 0
cfg.members[5].votes = 0
rs.reconfig(cfg)
```

# Replica & ReplicaSets

- ReplicaSets 환경 설정
  - 하나의 리프리카셋으로 생성될 primary DB와 secondary DB, 아비티 DB의 홈 경로를 설정한다.

```
d:₩>mkdir d:₩mongDB₩DISK1  
d:₩>mkdir d:₩mongDB₩DISK2  
d:₩>mkdir d:₩mongDB₩ARBIT
```

- 리프리카셋을 활성화한다.

d:₩mongod --dbpath rptmongo/localhost:10002	d:₩mongdb₩DISK1	—port	10001	—repISet
d:₩mongod —dbpath rptmongo/localhost:10001	d:₩mongdb₩DISK1	—port	10002	—repISet
d:₩mongod —dbpath rptmongo/localhost:10001	d:₩mongdb₩ARBIT	—port	10003	—repISet

# Replica & ReplicaSets

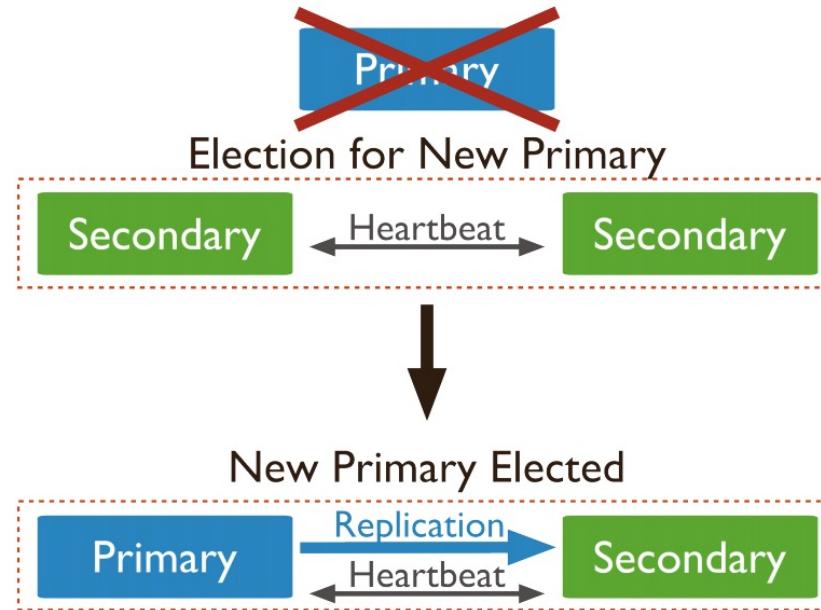
- ReplicaSets 환경 설정
  - 리프리카셋을 초기화한다.

```
D:\W>mongo localhost:10001/admin
```

```
db.runCommand({ "replSetInitiate" : {"_id" :"rptmongo", "members":  
[ {"_id" : 1, "host" : "localhost:10001"}, //primary 서버 등록  
{"_id" : 2, "host" : "localhost:10002"}, //secondary 서버 등록  
{"_id" : 3, "host" : "localhost:10003", arbiterOnly : true} ] }}) //아비터 서버 등록
```

# Replica & ReplicaSets

- ReplicaSets FailOver



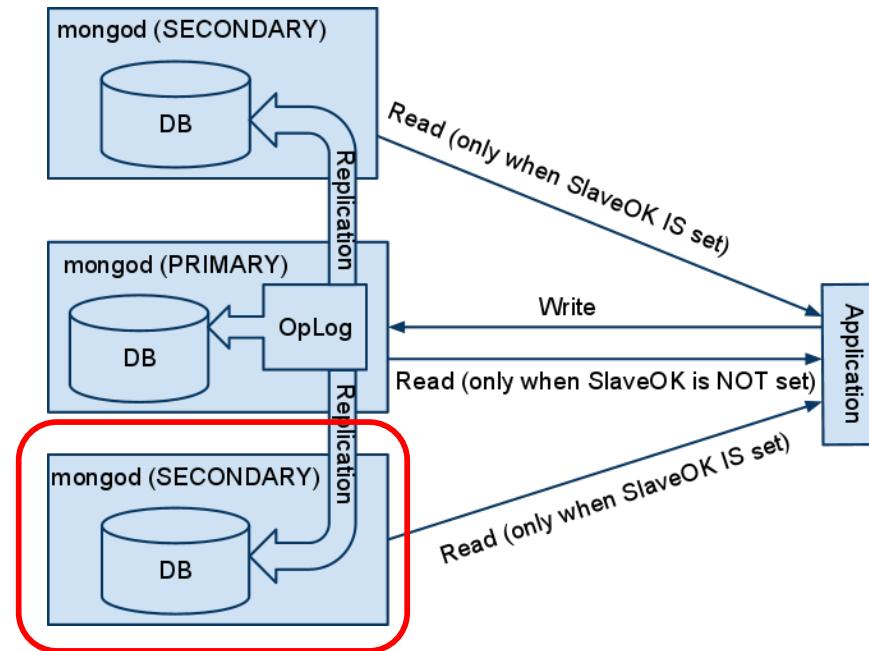
```
D:₩>mongo localhost:10001/admin
```

```
db.printReplicationInfo()  
db.printSlaveReplicationInfo()  
db.shutdownServer()
```

//admin으로 시행, 첫번째 primary 서버가 종료되어 secondary 서버가 primary가 된다.

# Replica & ReplicaSets

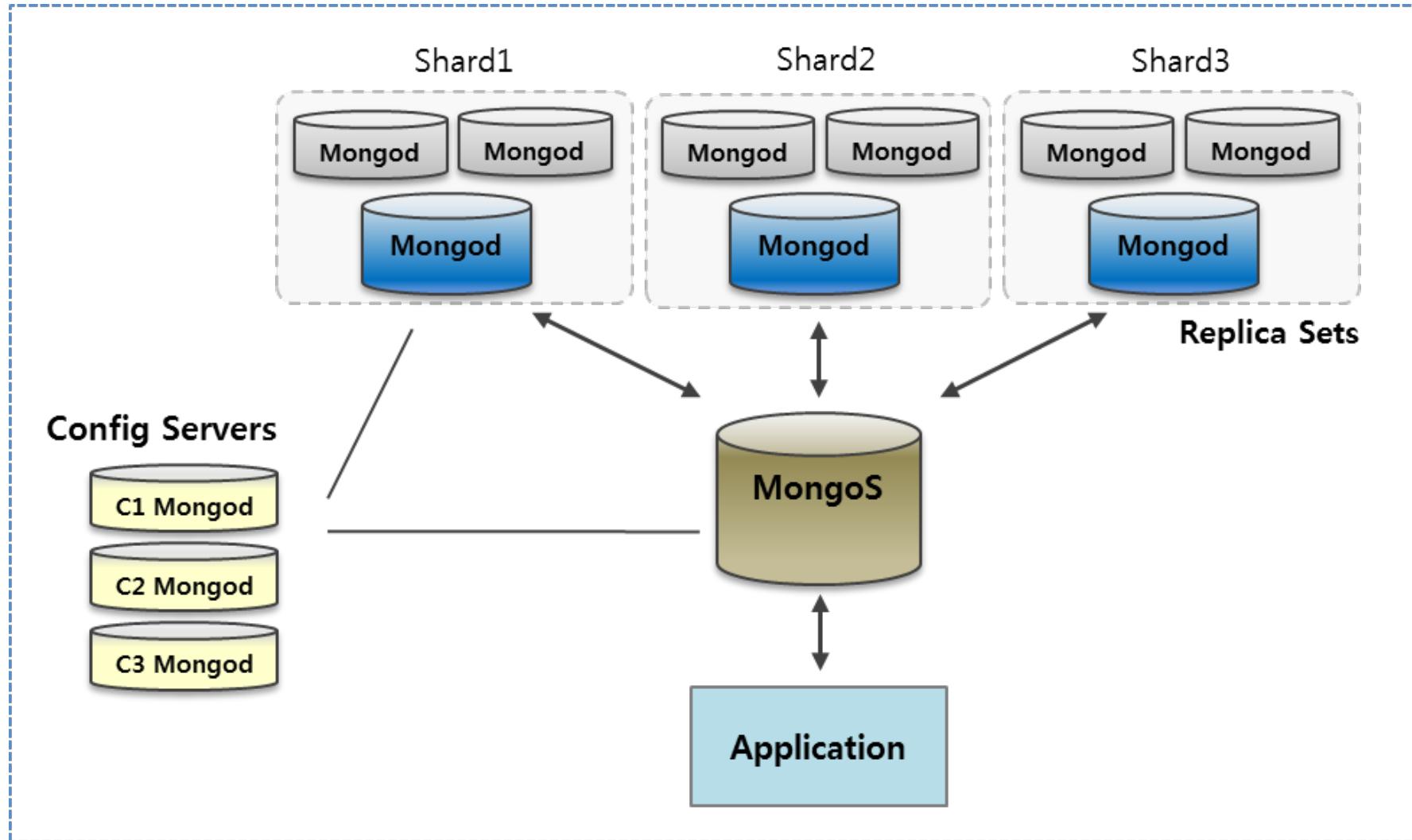
- ReplicaSets 복제 서버의 추가와 삭제



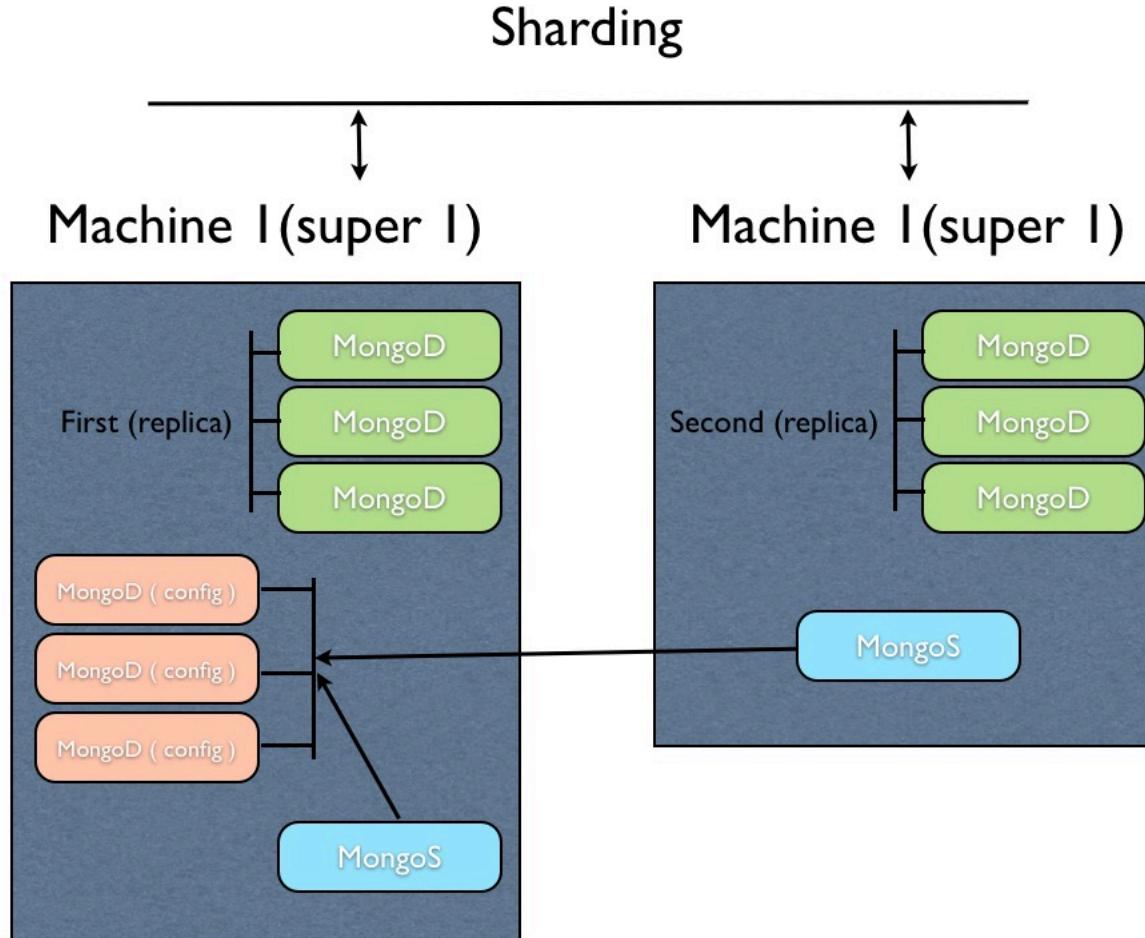
D:₩>mongo localhost:10002

```
rs.add("localhost:10004")      // 새로운 복제 서버 추가  
db.printSlaveReplicationInfo()  
rs.remove("localhost:10004")    // 해당 복제 서버 삭제
```

# Replica & ReplicaSets



# Replica & ReplicaSets 예제



- <http://blog.naver.com/PostView.nhn?blogId=parkjy76&logNo=30115082671>
- <http://cookbook.mongodb.org/operations/convert-replica-set-to-replicated-shard-cluster/>

# sharding & ReplicaSets 예제

## 구성 방법

- Config server 1대
- Mongos 2대
- Replicaset 1set
- Backupserver 1대

```
mongod --port 28000 --configsvr --dbpath d:\mongodb\config1
```

```
mongos --port 28101 --chunkSize 1 --configdb localhost:28000
```

```
mongos --port 28102 --chunkSize 1 --configdb localhost:28000
```

```
mkdir s1 (Backupserver 1대)
```

```
mkdir s2.r1 (ReplicaSet 첫번째 서버)
```

```
mkdir s2.r2 (ReplicaSet 두번째 서버)
```

```
mkdir s2.r3 (ReplicaSet 세번째 서버)
```

# sharding & ReplicaSets 예제

## 구성 방법

- Config server 1대
- Mongos 2대
- Replicaset 1set
- Backupserver 1대

```
mongod --port 28201 --dbpath d:\mongodb\ds1 --rest  
mongod --port 28211 --dbpath d:\mongodb\ds2.r1 --replSet s2 --rest  
mongod --port 28212 --dbpath d:\mongodb\ds2.r2 --replSet s2 --rest  
mongod --port 28213 --dbpath d:\mongodb\ds2.r3 --replSet s2 --rest
```

## Replicaset 첫번째 서버에 접속

```
mongo localhost:28211
```

# sharding & ReplicaSets 예제

## Replicaset 초기화

```
config = {  
  _id : 's2', members : [  
    {_id : 0, host: "localhost:28211"},  
    {_id : 1, host: "localhost:28212"},  
    {_id : 2, host: "localhost:28213"}  
  ]}  
  
rs.initiate(config);
```

## Replicaset 상태확인

```
rs.status( );
```

## 웹에서 관리화면 가능

```
http://localhost:29201/
```

# sharding & ReplicaSets 예제

## Replicaset 초기화

```
config = {  
  _id : 's2', members : [  
    {_id : 0, host: "localhost:28211"},  
    {_id : 1, host: "localhost:28212"},  
    {_id : 2, host: "localhost:28213"}  
  ]}  
  
rs.initiate(config);
```

## Replicaset 상태확인

```
rs.status( );
```

## 웹에서 관리화면 가능

```
http://localhost:29201/
```

# sharding & ReplicaSets 예제

## sharding 설정

```
mongo localhost:28101 // 라우팅 서버 중 하나에 접속
use admin
db.runCommand({addshard:"localhost:28201"});
db.runCommand({addshard:"s2/localhost:28211, localhost:28212, localhost:28213"});
db.printShardStatus();
```

## 테스트 데이터 입력

```
mongo localhost:28101
use admin
db.runCommand({enablesharding:"test"});
db.runCommand({shardcollection : "test.things", key : {empno:1},{unique:true}})
for(var n= 100000; n <= 300000; n++) db.things.save({empno:n, ename:"test", sa1:1000})
```

# 14

## **mongoDB의 유 틸리티**

# 백업 및 복구

- **MongoDump / MongoStore**
  - 컬렉션별 백업
    - mongodump --db test --collection employees --out (물리적 저장공간)
  - 데이터베이스 단위 백업
    - mongodump --db test --out (물리적 저장 공간)
  - 데이터베이스 전체 백업
    - mongodump --out (물리적 저장 공간)
  - 콜렉션 단위 복구
    - mongorestore --db test --collection things
    - d:\MongoDB\test\things.bson
  - 데이터베이스 단위 복구
    - mongorestore --db test d:\MongoDB\test
  - 데이터베이스 전체 복구
    - mongorestore --objcheck --drop .

# 백업 및 복구

- **copyDataBase / CloneDatabase**
  - copydatabase
    - db.copyDatabase("원본db", "대상db", "호스트명")
  - cloneDatabase
    - Remote 데이터베이스 복제
    - db.cloneDatabase("E\ host명")

# 백업 및 복구

- **MongoExport / MongoImport**
  - 운영체제 상의 파일 형태로 다운로드 / 업로드 하는 기능
  - mongoExport
    - mongoexport -d test -c things -o things.csv --csv  
-f empno, ename, job, hiredate, sal, deptno
  - mongoimport
    - mongoimport -d test --collection employees mongoimport.json
    - mongoimport -d test -c employees --type csv -f mongoimport.csv -headerline
    - mongoimport -d test -c employees --type tsv -f mongoimport.tsv -headerline

# 유ти리티

- **Collection의 read/write 상태 정보 제공**
  - Mongotop 5 // 5초마다 분석 결과 제공
- **다양한 쿼리 작업 및 메모리 상태 정보 출력**
  - Mongostat --rowcount 50 // 50라인의 분석 결과만 출력
- **Web monitoring**
  - 포트번호 + 1000번 웹으로 접속 (기본값 : 28017)

# 유 틸리티

- **Web monitoring**
  - 포트번호 + 1000번 웹으로 접속 (기본값 : 28017)

The screenshot shows a web browser window with the following details:

- Title Bar:** mongod APSEODESK058:10001
- Address Bar:** localhost:11001
- Toolbar:** Pig Toolbox Tip, AB 테스트, ACCOUNT, [Linux] vi 필수 명령어, SCREEN 사용법, [MSDN]SQL Server ...
- Content Area:**
  - Section:** mongod APSEODESK058:10001
  - Commands:** List all commands, **Replica set status** (highlighted with a red box).
  - Server Status:** Commands: buildInfo cursorInfo features isMaster listDatabases replSetGetStatus serverStatus top  
db version v2.0.6, pdfile version 4.5  
git hash: e1c0cbc25863f6356aa4e31375add7bb49fb05bc  
sys info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service\_pack='Service Pack 1') BOOST\_LIB  
uptime: 447 seconds
  - Low level:** requires read lock  
time to get readlock: 0ms  
# databases: 1
  - replication:** --repSet first
  - master:** 1  
**slave:** 0
- clients:** A table showing client connections.

Client	Opid	Active	LockType	Waiting	SecsRunning	Op	Namespace	Query
initandlisten	0		W			2004	local.system.namespaces	{ name: /^local.temp}
journal	0		0			0		

# 유 틸리티

- **Web monitoring**
  - 포트번호 + 1000번 웹으로 접속 (기본값 : 28017)

Replica Set Status APSEODI

localhost:11001/\_repSet

Pig Toolbox Tip AB 테스트 ACCOUNT [Linux] vi 필수 명령어 SCREEN 사용법 [MSDN]SQL Server ...

[Home](#) | [View Repset Config](#) | [repSetGetStatus](#) | [Docs](#)

Set name: first  
Majority up: yes

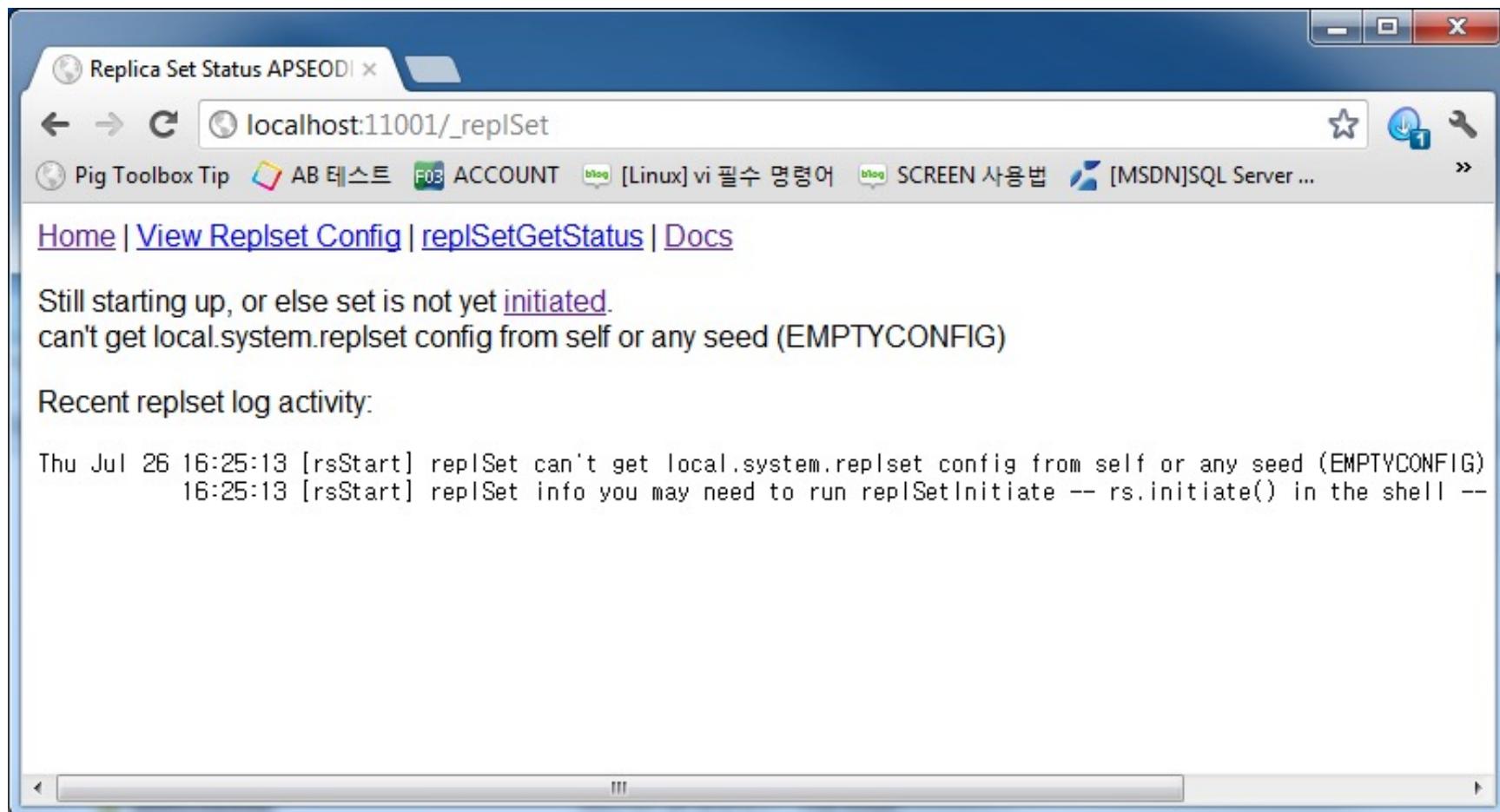
Member	id	Up	cctime	Last heartbeat	Votes	Priority	State	Messages	optime	skew
localhost:10001 (me)	1	1	79 secs		1	1	PRIMARY		<a href="#">5010f165:1</a>	
<a href="#">localhost:10002</a>	2	1	28 secs	0 secs ago	1	1	SECONDARY		<a href="#">5010f165:1</a>	
<a href="#">localhost:10003</a>	3	1	28 secs	0 secs ago	1	1	ARBITER		<a href="#">0:0</a>	

Recent repset log activity:

```
Thu Jul 26 16:26:49 [rsStart] repSet can't get local.system.replset config from self or any seed (EMPTYCONFIG)
16:26:49 [rsStart] repSet info you may need to run replSetInitiate -- rs.initiate() in the shell -- if that is
16:26:59 [rsStart] repSet can't get local.system.replset config from self or any seed (EMPTYCONFIG)
16:27:09 .
16:27:19 .
16:27:29 .
16:27:33 [conn3] repSet replSetInitiate admin command received from client
16:27:33 [conn3] repSet replSetInitiate config object parses ok, 3 members specified
16:27:33 [conn3] repSet replSetInitiate all members seem up
16:27:33 [conn3] repSet info saving a newer config version to local.system.replset
16:27:33 [conn3] repSet saveConfigLocally done
16:27:33 [conn3] repSet replSetInitiate config now saved locally. Should come online in about a minute.
16:27:39 [rsStart] repSet STARTUP2
16:27:39 [rsHealthPoll] repSet member localhost:10002 is up
16:27:39 [rsHealthPoll] repSet member localhost:10003 is up
```

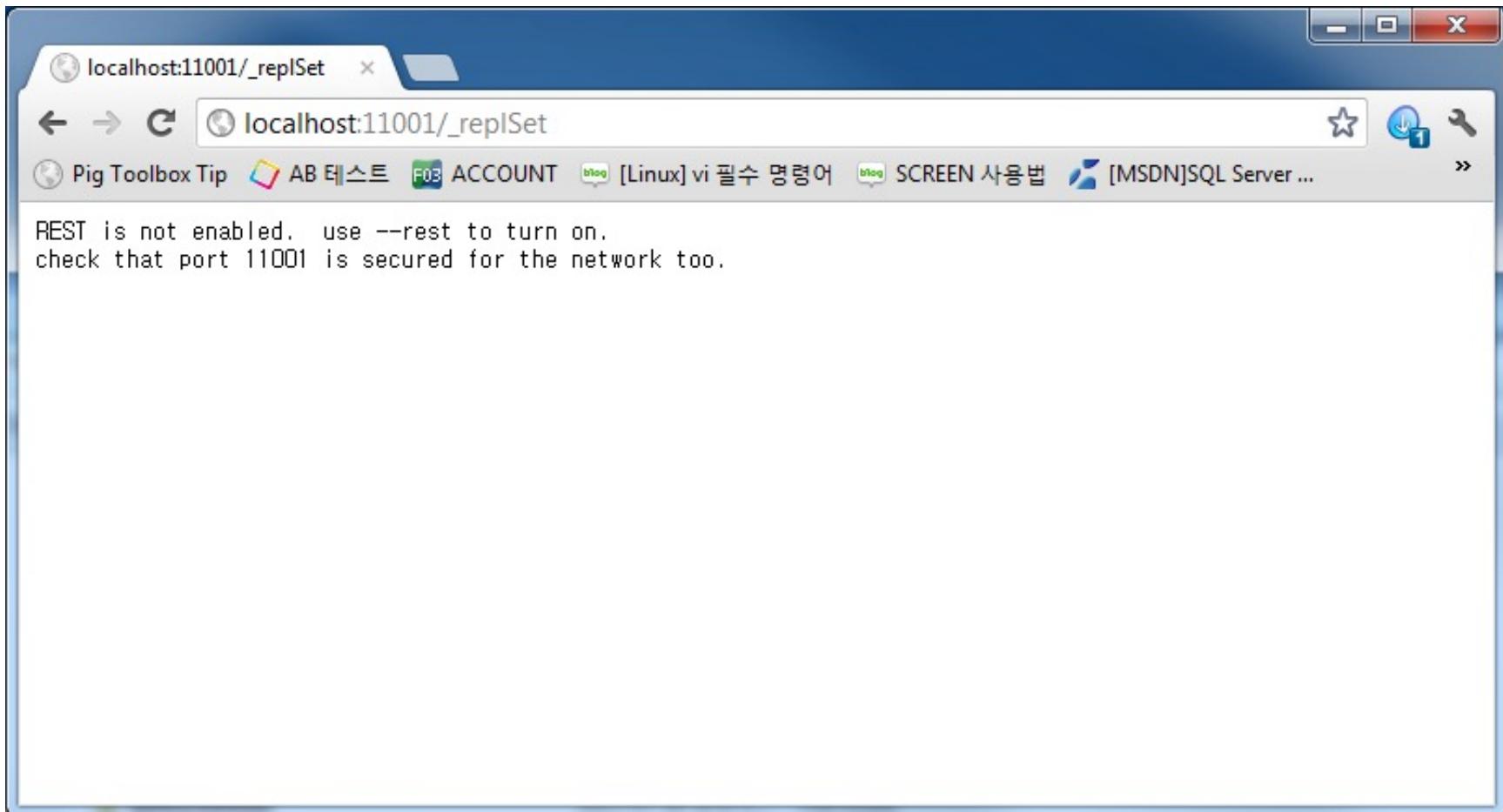
# 유 틸리티

- **Web monitoring**
  - 포트번호 + 1000번 웹으로 접속 (기본값 : 28017)



# 유 틸리티

- **Web monitoring**
  - 포트번호 + 1000번 웹으로 접속 (기본값 : 28017)



# MongoDB 연결 Sample

C# sample

<http://www.codeproject.com/Articles/273145/Using-MongDB-with-the-Official-Csharp-Driver>

The screenshot shows a CodeProject article page. At the top, there's a banner for 'Godomall' and a search bar. The main navigation menu includes 'home', 'articles', 'quick answers', 'discussions', 'features', 'community', and 'help'. On the left, there's a sidebar with links for 'Article' (Browse Code, Bugs / Suggestions, Stats, Revisions (2), Alternatives, Comments & Discussions (13)), 'Code Project' (Browse Code, Bugs / Suggestions, Stats, Revisions (2), Alternatives, Comments & Discussions (13)), and 'Workspaces' (Fork me on Workspaces). The main content area features the title 'Using MongoDB with the Official C# Driver' by Ercan Anlama, posted on 24 Oct 2011. It has a 4.94 rating from 19 votes. Below the title is a 'Download demo project - 2.49 MB' button. The 'Introduction' section explains that MongoDB is an appealing applicant in the NoSQL world, providing support for drivers, comprehensive documents, and a large community. It notes that while there are alternatives like LINQ support, the author prefers the official C# driver. The 'A few words about the MongoDB data structure' section is also visible. The right sidebar contains 'About Article' details: Type (Article), Licence (CPOL), First Posted (24 Oct 2011), Views (36,761), Downloads (1,980), Bookmarked (36 times), and categories (C# .NET Dev, Intermediate, mongodb). There are also 'Print' and 'Email' buttons, and social sharing links for LinkedIn, Google+, and Facebook.

# **THANK YOU**