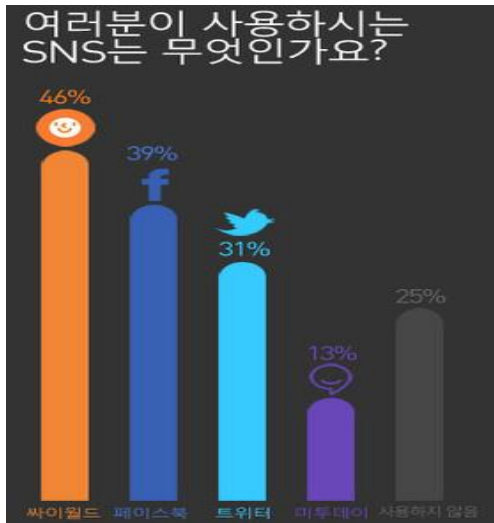
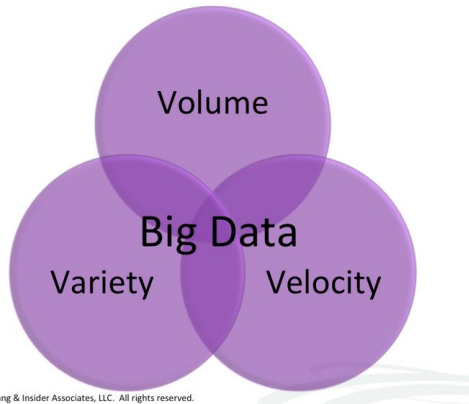




pyk

# 빅데이터

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



- 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 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 Hardware 에 구축할 수 있음
  - 기존 RDB와 동일하게 데이터 처리가 가능

## MongoDB란?

- 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
  - 장애가 발생하더라도 피해를 최소로 줄이기 위함.
- 관계형 데이터베이스의 주요 기능을 사용함. CRUD 위주의 다중 트랜잭션 처리.
  - 트랜잭션 위주의 데이터 처리 가능
  - NoSQL은 빅데이터의 빠른 저장과 추출 및 분석을 용이하기 함.

## 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

- 다운로드

- <https://www.mongodb.com/download-center/enterprise>

- 설치 환경 및 지원 드라이버

설치 가능한 플랫폼

- 윈도우 32비트 / 64비트
- 리눅스 32비트 / 64비트
- MAC OS X-32비트 / 64비트

Select the server you would like to run:

MongoDB Community Server

FEATURE RICH. DEVELOPER READY.

Version

4.2.3 (current release)

OS

Windows x64 x64

Package

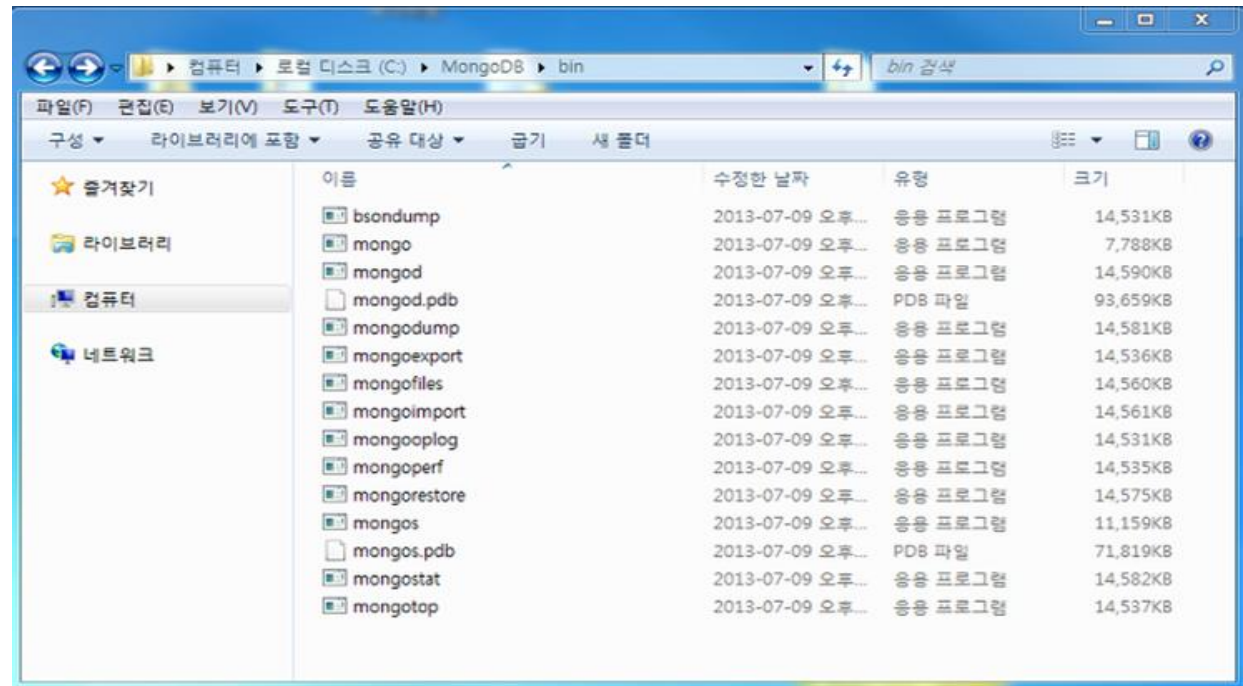
MSI

Download

## 설치 요령

- 방법1 :다운로드 된 MongoDB의 압축 파일 해제 (bin : 관련 실행 파일이 설치된 경로)
- 방법2 : 다운로드 된 msi 파일로 설치

- 하드 디스크 드라이브에 관련 sw가 설치될 물리적 경로 설정
- 예) c:\mongodb
- 확인방법 : bin 폴더 확인



## 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]

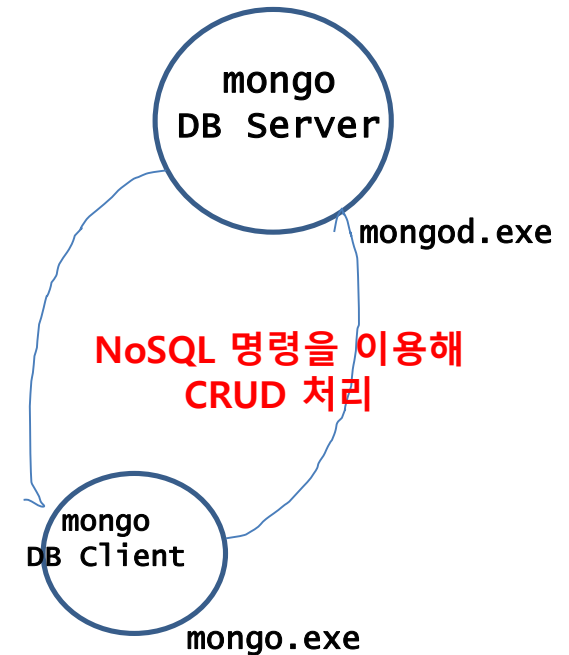
# MongoDB에서의 데이터 처리

- 시작과 종료
  - 메모리 영역, 파일 영역, 프로세서 영역 활성화
    - 데이터가 저장될 물리적인 저장 경로 생성
  - **c:\data\db**
  - mongoDB 버전 확인
    - **mongod --version 실행**

```
cmd 명령 프롬프트
Microsoft Windows [Version 10.0.18362.207]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\tykpark>mongod --version
db version v4.2.2
git version: a0bbbff6ada159e19298d37946ac8dc4b497eadf
allocator: tcmalloc
modules: enterprise
build environment:
  distmod: windows-64
  distarch: x86_64
  target_arch: x86_64

C:\Users\tykpark>
```



# MongoDB에서의 데이터 처리

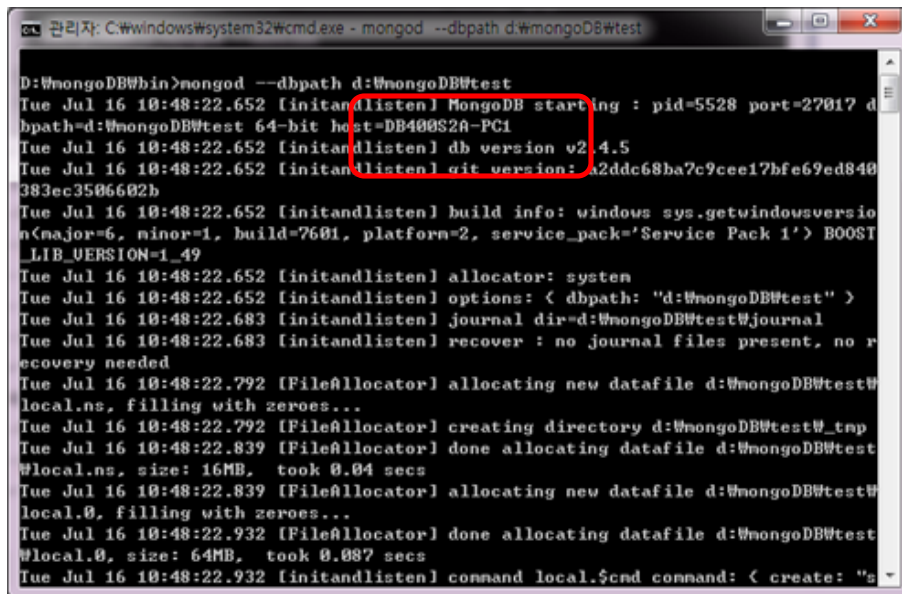
- 시작과 종료

- mongoDB 서버 실행

- MONGODB.EXE 파일 실행
    - mongodb/bin 폴더에서 실행
    - 예) **mongod --dbpath c:\mongodb\data\db**
    - 여기서 dbpath는 데이터가 저장될 물리적인 공간 지정함.
    - 32bit 운영체제에서는 반드시 --journal 을 옵션으로 구현

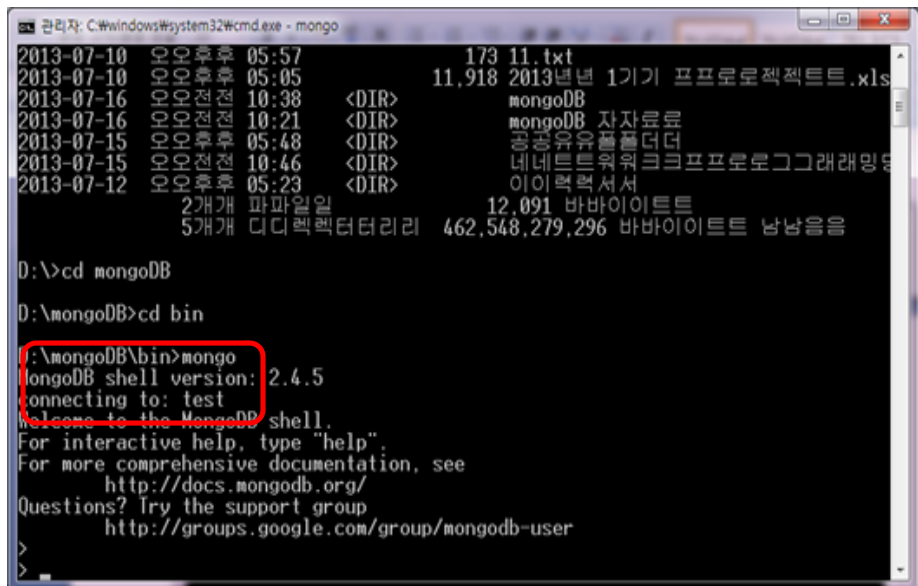
- 클라이언트 프로그램으로 mongoDB에 접속

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



```
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 db
bpah=d:\mongodb\test 64-bit host=DESKTOP-PC1
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.getwindowsversio
n(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 r
ecover 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\
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: "s
```



```
C:\Windows\system32\cmd.exe - mongo

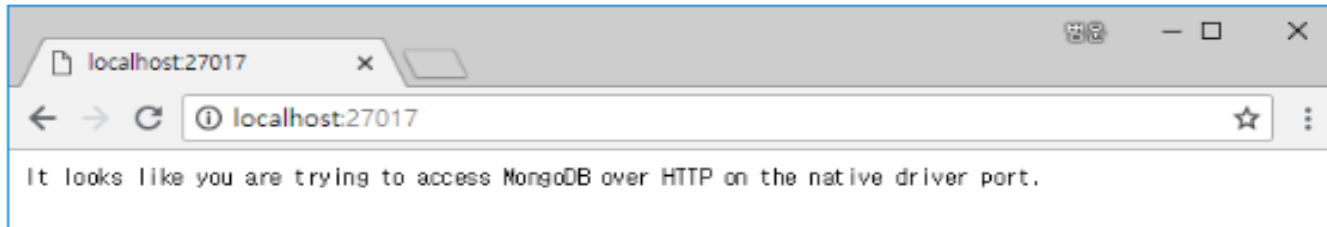
2013-07-10 오후 05:57 173 11.txt
2013-07-10 오후 05:05 11,918 2013년 1학기 프로젝트.xls
2013-07-16 오후 10:38 <DIR> mongoDB
2013-07-16 오후 10:21 <DIR> mongoDB 자료료
2013-07-15 오후 05:48 <DIR> 공유폴더
2013-07-15 오후 10:46 <DIR> 네트워킹프로그래밍
2013-07-12 오후 05:23 <DIR> 이력서
2개 파일 12,091 바이트
5개 디렉터리 462,548,279 바이트 남음

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

브라우저를 실행시켜 localhost:27017 를 입력하여 아래처럼 메시지가 출력되면 된다.



## • 시작과 종료

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

```
관리자: C:\windows\system32\cmd.exe - mongo
> 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

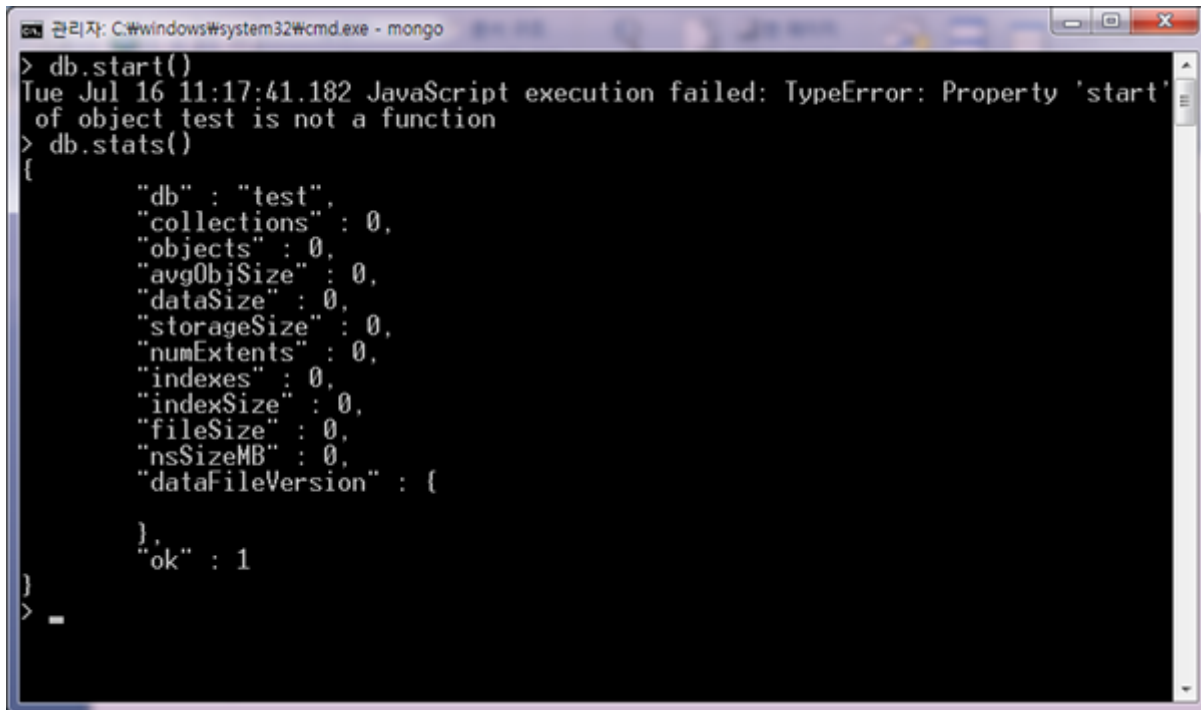
  show dbs                show database names
  show collections         show collections in current database
  show users              show users in current database
  show profile            show most recent system.profile entries with time >= 1ms
  show logs               show the accessible logger 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 further iterate
  DBQuery.shellBatchSize = x set default number of items to display on screen
```

# MongoDB에서의 데이터 처리

- 시작과 종료

- 데이터가 저장되어 있는 논리적인 구조 확인

- use db명
    - 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)

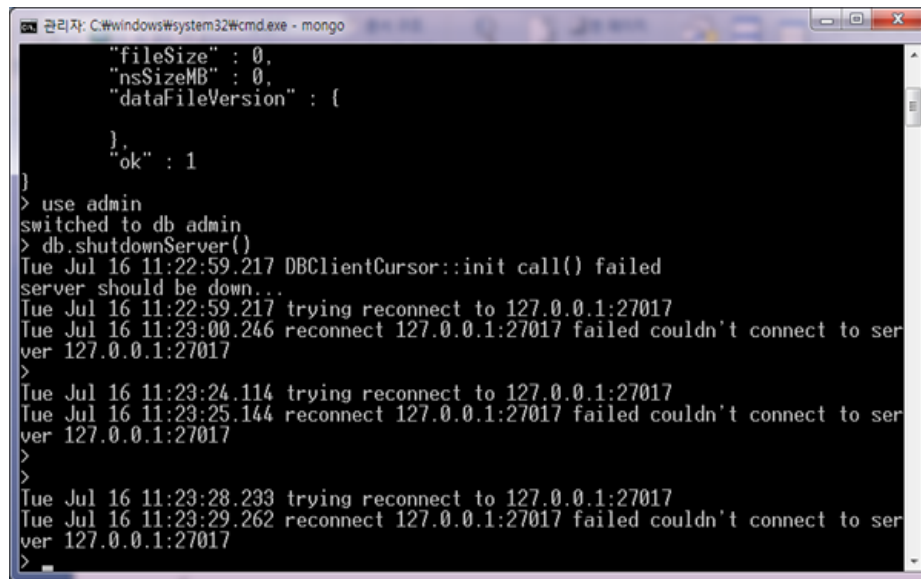


```
> db.start()
Tue Jul 16 11:17:41.182 JavaScript execution failed: TypeError: Property 'start'
of object test is not a function
> db.stats()
{
  "db" : "test",
  "collections" : 0,
  "objects" : 0,
  "avgObjSize" : 0,
  "dataSize" : 0,
  "storageSize" : 0,
  "numExtents" : 0,
  "indexes" : 0,
  "indexSize" : 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



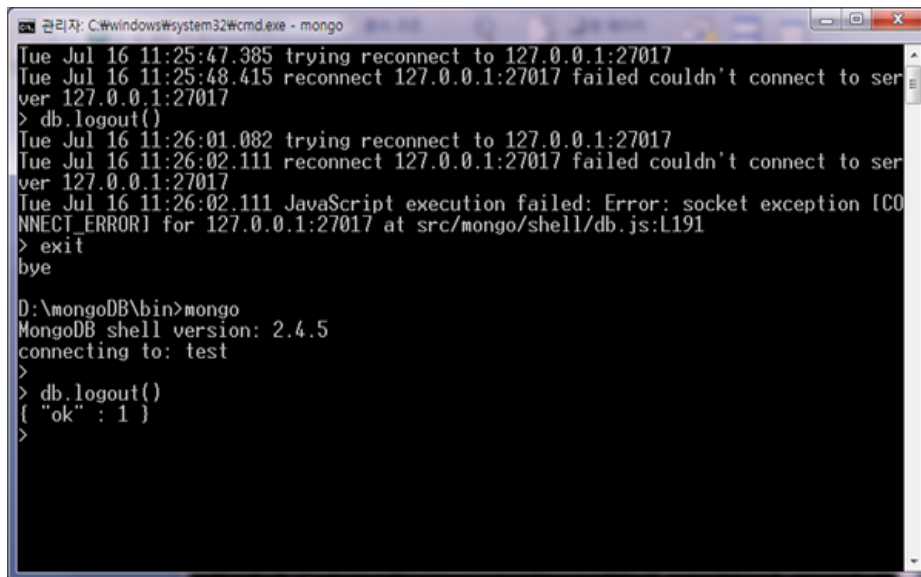
```

{
  "fileSize" : 0,
  "nsSizeMB" : 0,
  "dataFileVersion" : {
    },
    "ok" : 1
  }
}
> use admin
switched to db admin
> db.shutdownServer()
Tue Jul 16 11:22:59.217 DBClientCursor::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
>

```

- 시작과 종료

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



```

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 }
>

```

## Collection 데이터 관리

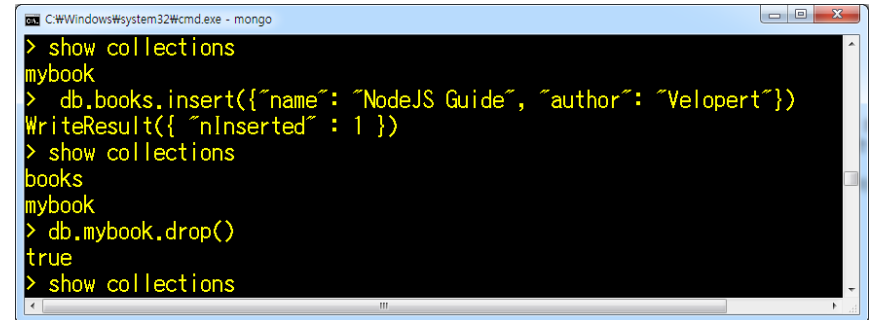
- 정형화된 데이터 구성요소(컬럼명, 데이터 타입과 길이, 제약 조건)가 결정되지 않아도 됨.
- Collection 생성 관리

- Capped Collection : 최초 제안된 크기로 생성된 공간(익스텐드) 내에서만 데이터 저장 가능하고, 만약 최초의 공간이 모두 사용되면 다시 처음으로 돌아가서 기존공간을 재사용하는 타입의 컬렉션. 기업에서는 로그 데이터 저장 등 일정한 기간 내에서만 저장, 관리가 필요한 데이터에 적용.
- Non Capped Collection : 관계형 데이터베이스의 테이블처럼 디스크 공간이 허용하는 범위 안에서 데이터를 저장 할 수 있는 타입

- collection 의 생성

- **db.createCollection( "컬렉션명"[,옵션])**

- db.createCollection( "mybooks"[,옵션])



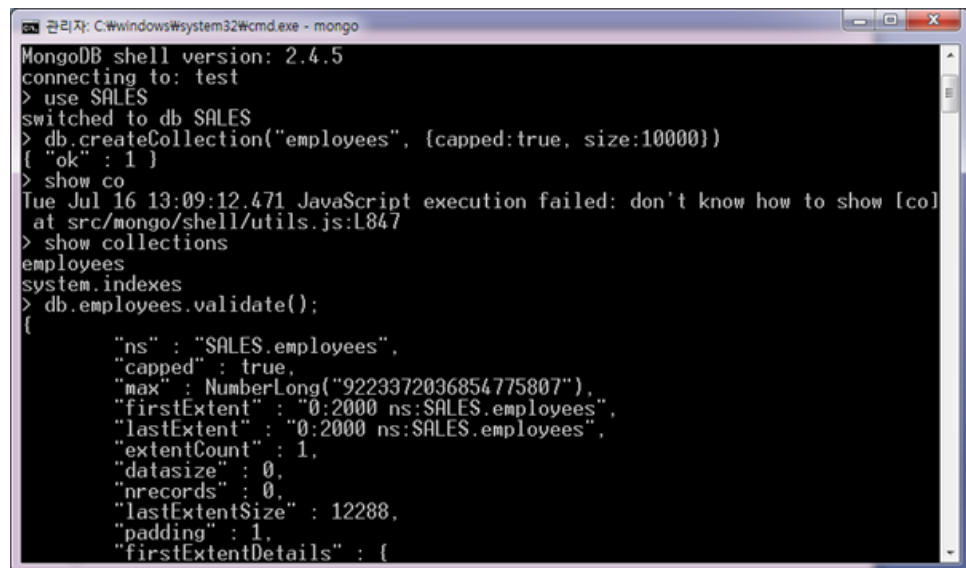
```
C:\Windows\system32\cmd.exe - mongo
> show collections
mybook
> db.books.insert({~name~: "NodeJS Guide", ~author~: "Velopert"})
WriteResult({ "~inserted~": 1 })
> show collections
books
mybook
> db.mybook.drop()
true
> show collections
```

- 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 [col
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 저장할 때에는 save 메소드 이용
```

```
db.things.save(n)
```

```
db.things.find( ) //저장된 데이터를 검색할 때는 find 메소드 실행
```

```
db.things.insert({ empno : 1102, ename : "king" }) // 데이터를 입력할 때 insert 메소드 사용  
가능함.
```

```
db.things.find( ) //저장된 데이터를 검색할 때는 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
```

# MongoDB의 기본적인 셸

## • 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("51ea1c0a6cb27183320d6ae5"), "ename" : "smith" }
{ "_id" : ObjectId("51ea1c126cb27183320d6ae6"), "empno" : 1101 }
```

## 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 웹

- 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 tok
en }
> db.things.update( {empno : 1101}, n})
Sat Jul 20 15:40:24.609 JavaScript execution failed: SyntaxError: Unexpected tok
en }
> db.things.update( {empno : 1101}, n)
> db.things.find()
{ "_id" : ObjectId("51ea29086cb27183320d6ae7"), "empno" : 1102, "ename" : "king"
}
{ "_id" : ObjectId("51ea1c0a6cb27183320d6ae5"), "ename" : "smith", "ebith" : [
]
}
{ "_id" : ObjectId("51ea1c126cb27183320d6ae6"), "empno" : 1101, "ebith" : "1999.
10.1" }
>
```

## • 기본 데이터 표현

### – 데이터 갱신 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 이 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" }
>
```

### – 모든 문서 삭제

- db.things.remove({ })

### – 조회

- db.things.find( )

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



# MongoDB의 기본적인 웹

- 기본 데이터 표현

- <참고! 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

컬렉션에 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 시간내에 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

# 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

- 저장될 때 바이너리 형태의 데이터로 변환 저장
    - Binary Serial Object Notation

- 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)
```

## • 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) }
```

### – 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", "sal" : 1000 }
```

### – 배열 변수에 결과 저장

- varcursor = db.things.find( )

### – 배열 데이터를 반복문을 통해 출력

- while(cursor.hasNext( )) printjson(cursor.next( ))

## • Data type 종류

- 배열 변수에 결과 저장
  - `var cursor = db.things.find( )`
- 배열 데이터를 반복문을 통해 출력
  - `while(cursor.hasNext( )) printjson(cursor.next( ))`
- 배열에 데이터를 저장할 수 있음.
  - `var cursor = db.things.find()`
- 17번째 배열에 저장된 데이터만 출력
  - `printjson(cursor[17])`
- 17번째 배열에 저장된 데이터만 출력
  - `var arr = db.things.find( ).toArray( )`
  - `arr[17]`

```
> var cursor = db.things.find()
> while(cursor.hasNext()) printjson(cursor.next())

  "_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
```

```
> var cursor = db.things.find()
> printjson(cursor[17])
  "_id" : ObjectId("51ec389c69710dc7169a30df"),
  "empno" : 1120,
  "ename" : "test",
  "sal" : 1000

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

## • Date 타입

- Date 함수를 통해 현재 날짜 정보를 DATE 타입으로 출력.
  - `x = new Date( )`
- 수립된 DATE 값을 문자형으로 출력.
  - `x.things( )`
- ISODate 함수로 날짜정보를 DATE 타입으로 출력.
  - `d=ISODate( )`
- 수집된 DATE 값 중에 해당월을 출력.
  - `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 (KST)
>
> d=ISODate()
ISODate("2013-07-21T20:03:00.406Z")
>
> 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
>
```

- **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}, up
date:{$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

## 산술 연산자

- \$add : 두 개의 값을 합산한 결과를 리턴
- \$divide : 두 개의 값을 나눈 결과를 리턴
- \$mod : 첫 번째 값을 두 번째 값으로 나눈 후 나머지 값을 리턴
- \$multiply : 첫 번째 값과 두 번째 값을 곱한 결과를 리턴
- \$subtract : 첫 번째 값에서 두 번째 값을 뺀 결과를 리턴

## 문자 연산자

- \$toUpper : 해당 문자열의 값을 소문자로 변환
- \$toLower : 해당 문자열의 값을 대문자로 변환

## 비교연산자 & Boolean 연산자

- \$lt(미만) : 두 개의 값을 비교해 첫 번째 값이 두 번째 값보다 작으면 true, 크거나 같으면 false를 리턴
- \$lte(이하) : 두 개의 값을 비교해 첫 번째 값이 두 번째 값보다 작거나 같으면 true, 크면 false를 리턴
- \$gt(초과) : 두 개의 값을 비교해 첫 번째 값이 두 번째 값보다 크면 true, 작으면 false를 리턴
- \$gte(이상) : 두 개의 값을 비교해 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 비교해 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

끝 ~~~