

**MongoDB**

**1**

**MongoDB**

# 목차

---

- 01. MongoDB
- 02. 시작하기
- 03. 문서의 생성, 갱신, 삭제
- 04. 쿼리하기
- 05. 색인
- 06. 집계
- 07. 관리

*1-01*

# MongoDB

# 01 MongoDB

## ■ NoSQL != RDBMS

- § MongoDB를 포함한 모든 NoSQL Solution 들은 관계형 Database 가 아님.
- § 하나의 데이터를 하나의 문서로 표현함.
- § 즉, Row(Record) 라는 개념보다 Document 라는 개념이 맞음.
- § 데이터는 **JSON Type** 으로 저장되는데, 정형화된 **Scheme** 가 없음.
- § 때문에 Scheme 변경에 따른 대량 작업(Insert 혹은 Update) 또한 없음.

## ■ 유연하고 손쉬운 확장

- § 저장해야할 데이터가 폭발적으로 증가하면서 저장소 및 성능의 확장이 고려될 수 있음.
- § NoSQL 은 성능 확장 또는 분산 확장을 제공함.
  - 성능 확장: 데이터베이스 서버의 성능을 향상시킴. (확장의 폭이 매우 큼)
  - 분산 확장: 여러 대의 데이터베이스 서버를 설치해 분산처리하도록 구성함. (경제적으로 저렴)

# 01 MongoDB

## ■ 다양한 기능

기능	설명
색인	다양한 쿼리의 속도를 빠르게 할 수 있는 일반적인 보조 색인, 고유 색인, 복합 색인, 공간 정보 색인 정보를 제공함.
저장 자바스크립트	개발자는 저장 프로시저 대신에 자바스크립트 함수와 같은 값을 서버 단에 저장해 쓸 수 있다.
집계	맵리듀스를 비롯한 다양한 집계 기능을 제공함.
고정 크기 컬렉션	제한 컬렉션은 크기가 고정되어 있으며, 로그 같은 특정 유형의 데이터에 유용 함.
파일 저장소	큰 파일과 파일의 메타데이터를 편리하게 저장할 수 있는 프로토콜을 제공 함.

§ 단, Join 은 NoSQL에서 제공하지 않는다.

# 01 MongoDB

## ■ 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

**1-02**

시작하기



# 목차

---

- 01. 설치
- 02. MongoDB 명령어
- 03. 컬렉션
- 04. 문서
- 05. 데이터 형(Type)

설치

## 02 시작하기 - 설치

### ■ <https://www.mongodb.org/downloads>

Current Stable Release (3.0.3)

5/12/2015 [Release Notes](#) [Changelog](#)

Download Source: [tgz](#) | [zip](#)

Windows

Linux

Mac OS X

Solaris

VERSION:

Windows 64-bit legacy

The 64-bit legacy build does not include SSL encryption and lacks newer features of Windows that enhance performance. Use this build for Windows Server 2003, 2008, or Windows Vista.

INSTALLATION PACKAGE:

DOWNLOAD (MSI)

BINARY: [Installation Instructions](#) [View Build Archive](#)

DOWNLOAD (ZIP)

[https://fastdl.mongodb.org/win32/mongodb-win32-x86\\_64-3.0.3.zip](https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-3.0.3.zip)

COPY LINK

## 02 시작하기 - 설치

### ■ C:\W 아래에 압축 풀기

setup	2015-03-26 오전 9...	엑스브 문서	1KB
mongodb-win32-x86_64-3.0.3	2015-06-03 오전 1...	압축(ZIP) 파일	69,664KB
mongodb-win32-x86_64-3.0.3	2015-06-03 오전 1...	파일 폴더	

### ■ C:\W 아래에 Database 저장소 만들기

mongodb-win32-x86_64-3.0.3	2015-06-03 오전 1...	압축(ZIP) 파일	69,664KB
mongodb-win32-x86_64-3.0.3	2015-06-03 오전 1...	파일 폴더	
data	2015-06-03 오전 1...	파일 폴더	

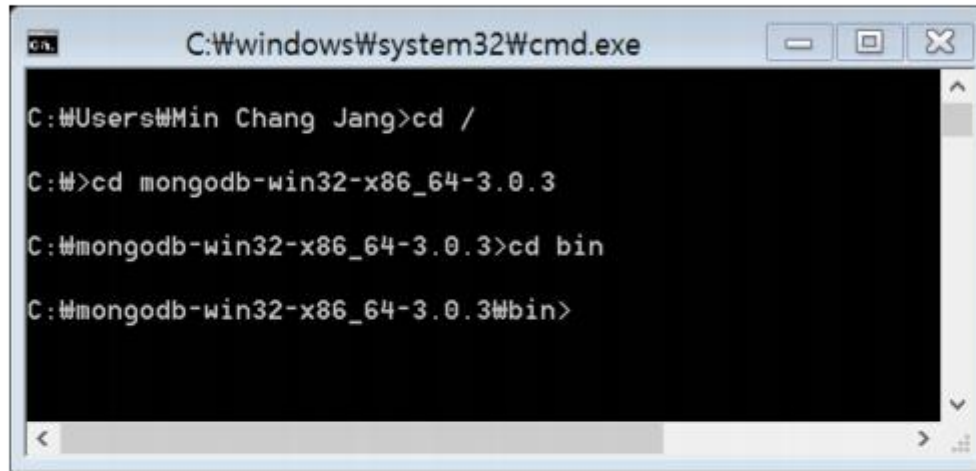
### ■ data 폴더 아래에 db폴더 만들기

↑  내 PC > 로컬 디스크 (C:) > data				
화면	↑	이름	수정한 날짜	유형
위치				크기
se		db	2015-06-03 오전 1...	파일 폴더

# MongoDB 명령어

## 02 시작하기 – MongoDB 명령어

### ■ MongoDB 시작



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

C:\Users\Min Chang Jang>cd /

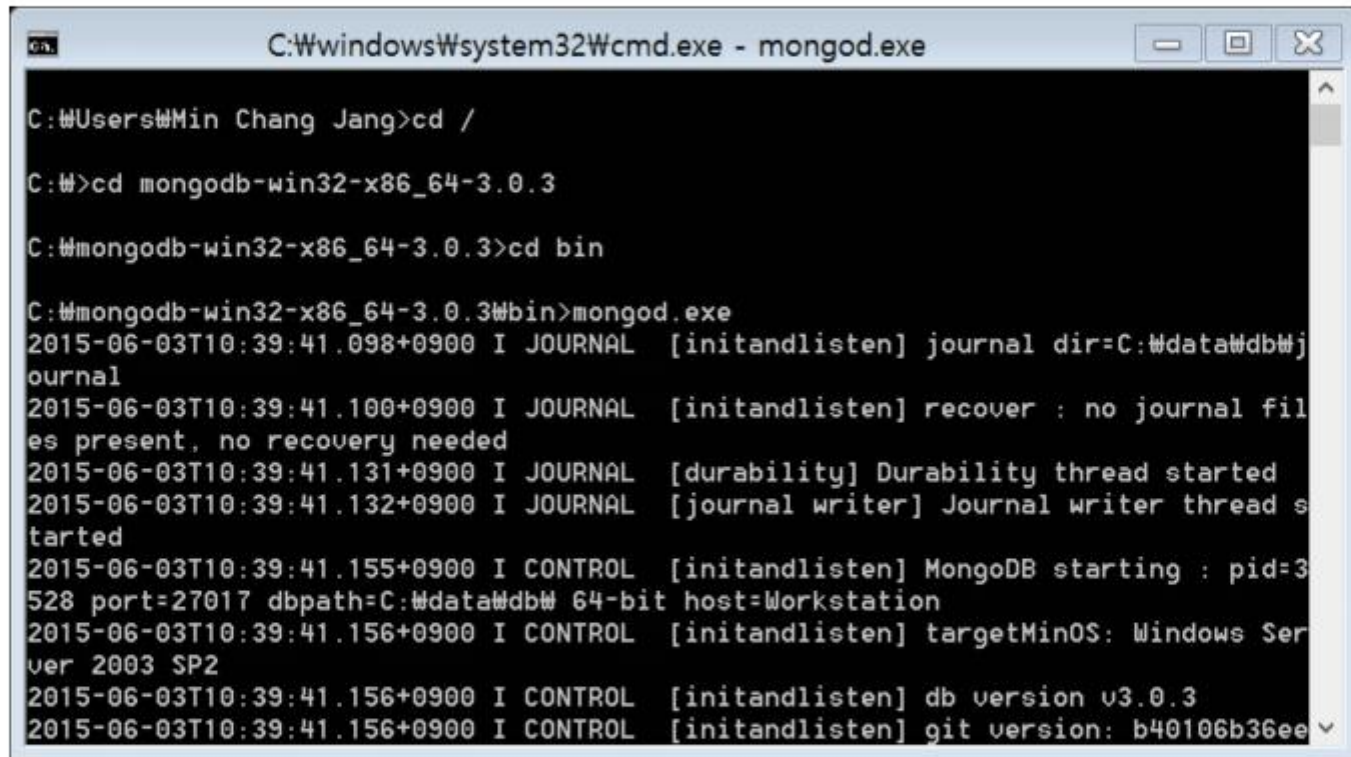
C:\>cd mongodb-win32-x86_64-3.0.3

C:\mongodb-win32-x86_64-3.0.3>cd bin

C:\mongodb-win32-x86_64-3.0.3\bin>
```

## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 시작



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

C:\Users\Min Chang Jang>cd /

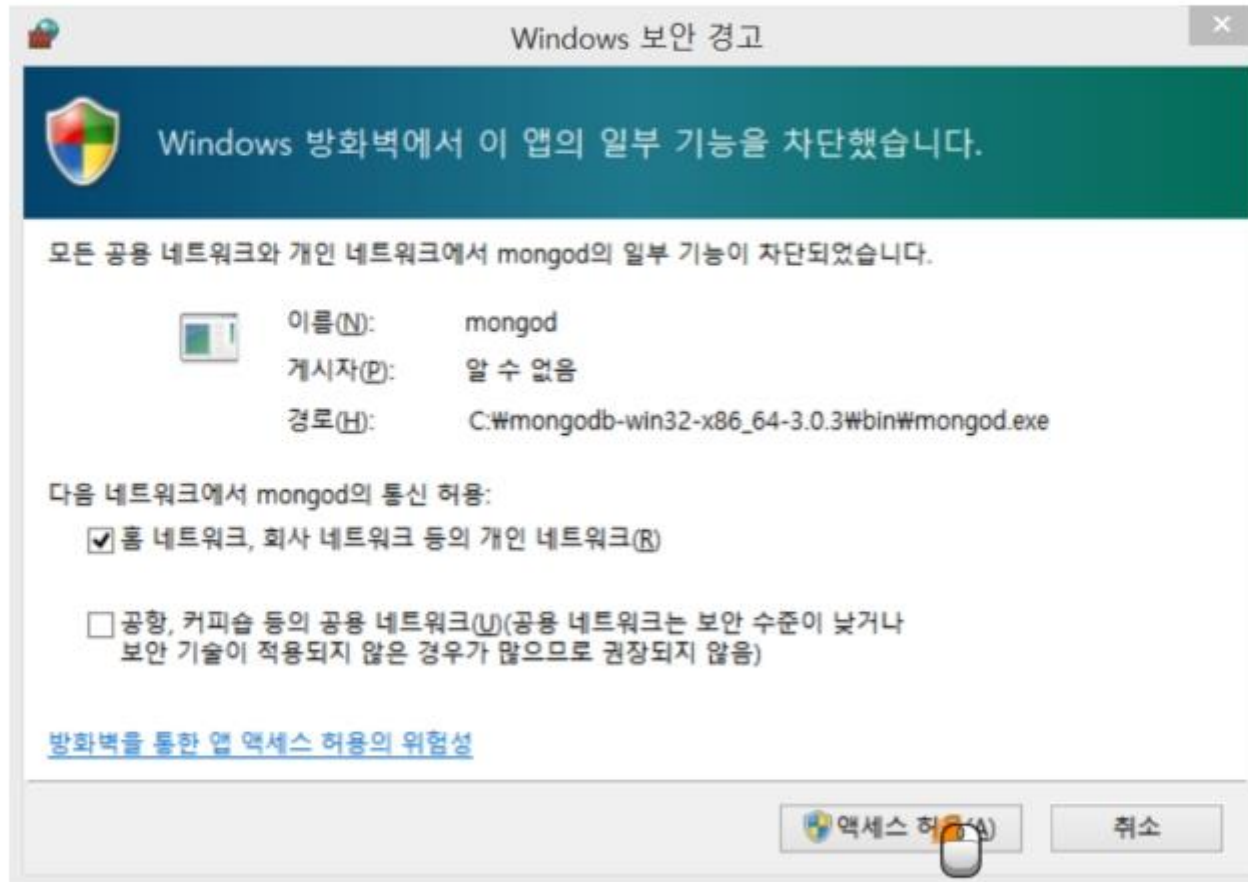
C:\>cd mongodb-win32-x86_64-3.0.3

C:\mongodb-win32-x86_64-3.0.3>cd bin

C:\mongodb-win32-x86_64-3.0.3\bin>mongod.exe
2015-06-03T10:39:41.098+0900 I JOURNAL [initandlisten] journal dir=C:\data\mdb\journal
2015-06-03T10:39:41.100+0900 I JOURNAL [initandlisten] recover : no journal files present, no recovery needed
2015-06-03T10:39:41.131+0900 I JOURNAL [durability] Durability thread started
2015-06-03T10:39:41.132+0900 I JOURNAL [journal writer] Journal writer thread started
2015-06-03T10:39:41.155+0900 I CONTROL [initandlisten] MongoDB starting : pid=3528 port=27017 dbpath=C:\data\mdb\ 64-bit host=Workstation
2015-06-03T10:39:41.156+0900 I CONTROL [initandlisten] targetMinOS: Windows Server 2003 SP2
2015-06-03T10:39:41.156+0900 I CONTROL [initandlisten] db version v3.0.3
2015-06-03T10:39:41.156+0900 I CONTROL [initandlisten] git version: b40106b36ee
```

## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 시작





## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 시작

#### § `mongod --dbpath [PATH]`

- 데이터 디렉토리로 사용할 경로를 지정함.
- 기본 값은 `/data/db` (Linux 기준)
- 여러 개의 MongoDB를 사용 할 경우 데이터 디렉토리를 개별로 정의해 주어야 함.

#### § `mongod --port [PORT_NUMBER]`

- 서버가 연결을 대기할 포트 번호를 지정함.
- 기본 값은 `27017` 포트를 사용함.
- 여러 개의 MongoDB를 사용할 경우 `port` 번호를 다르게 지정해 주어야함.

#### § `mongod --logpath [FILE_PATH]`

- `Log` 를 `Console`에 출력하지 않고 지정한 파일에 기록함.
- 동일한 파일의 경우 내용을 덮어 씌움.

#### § `mongod --logpath [FILE_PATH] --logappend`

- `--logappend` 옵션을 사용하면 파일을 덮어쓰지 않고, 이어 쓴다.

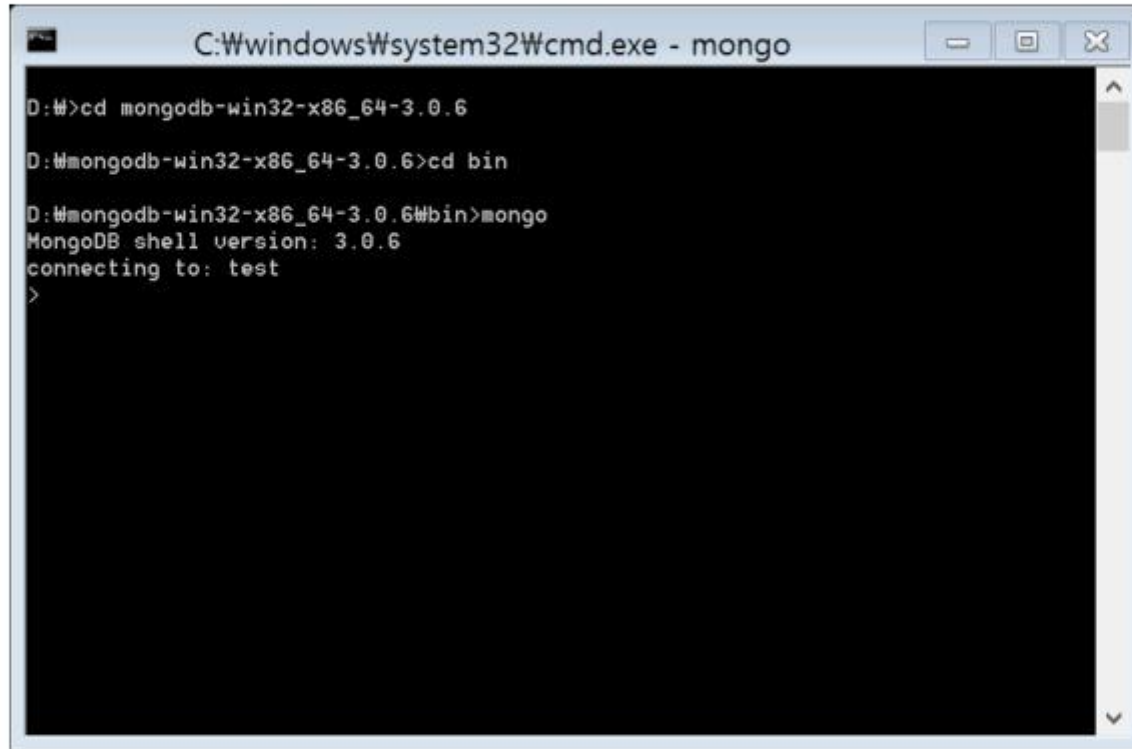
#### § `mongod --auth`

- 인가된 사용자만 접속할 수 있도록 함.

## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속

- MongoWbin 폴더에서 "mongo" 실행



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

D:\>cd mongodb-win32-x86_64-3.0.6

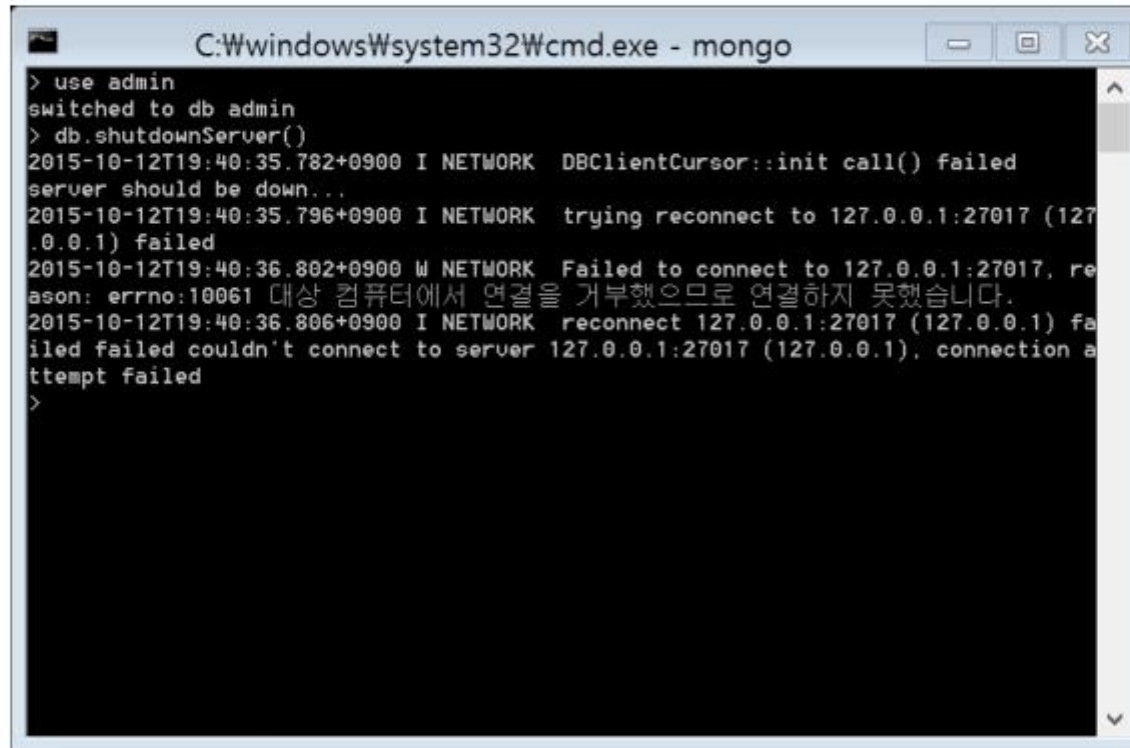
D:\mongodb-win32-x86_64-3.0.6>cd bin

D:\mongodb-win32-x86_64-3.0.6\bin>mongo
MongoDB shell version: 3.0.6
connecting to: test
>
```

## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 중지하기

- § > use admin
- § switched to db admin
- § > db.shutdownServer()

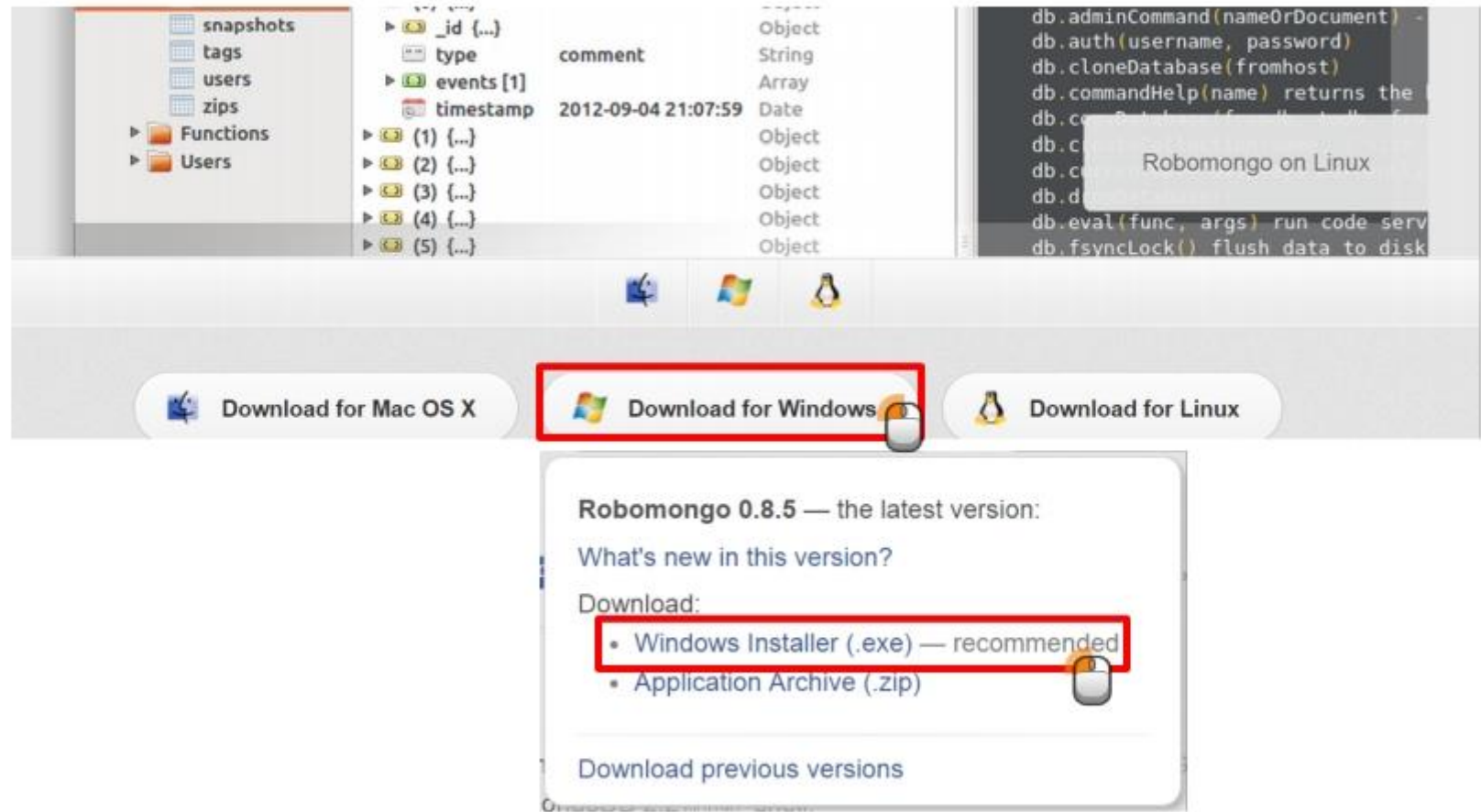


```
C:\Windows\system32\cmd.exe - mongo
> use admin
switched to db admin
> db.shutdownServer()
2015-10-12T19:40:35.782+0900 I NETWORK  DBClientCursor::init call() failed
server should be down...
2015-10-12T19:40:35.796+0900 I NETWORK  trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2015-10-12T19:40:36.802+0900 W NETWORK  Failed to connect to 127.0.0.1:27017, reason: errno:10061 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.
2015-10-12T19:40:36.806+0900 I NETWORK  reconnect 127.0.0.1:27017 (127.0.0.1) failed failed couldn't connect to server 127.0.0.1:27017 (127.0.0.1), connection attempt failed
>
```

## 02 시작하기 - MongoDB 명령어

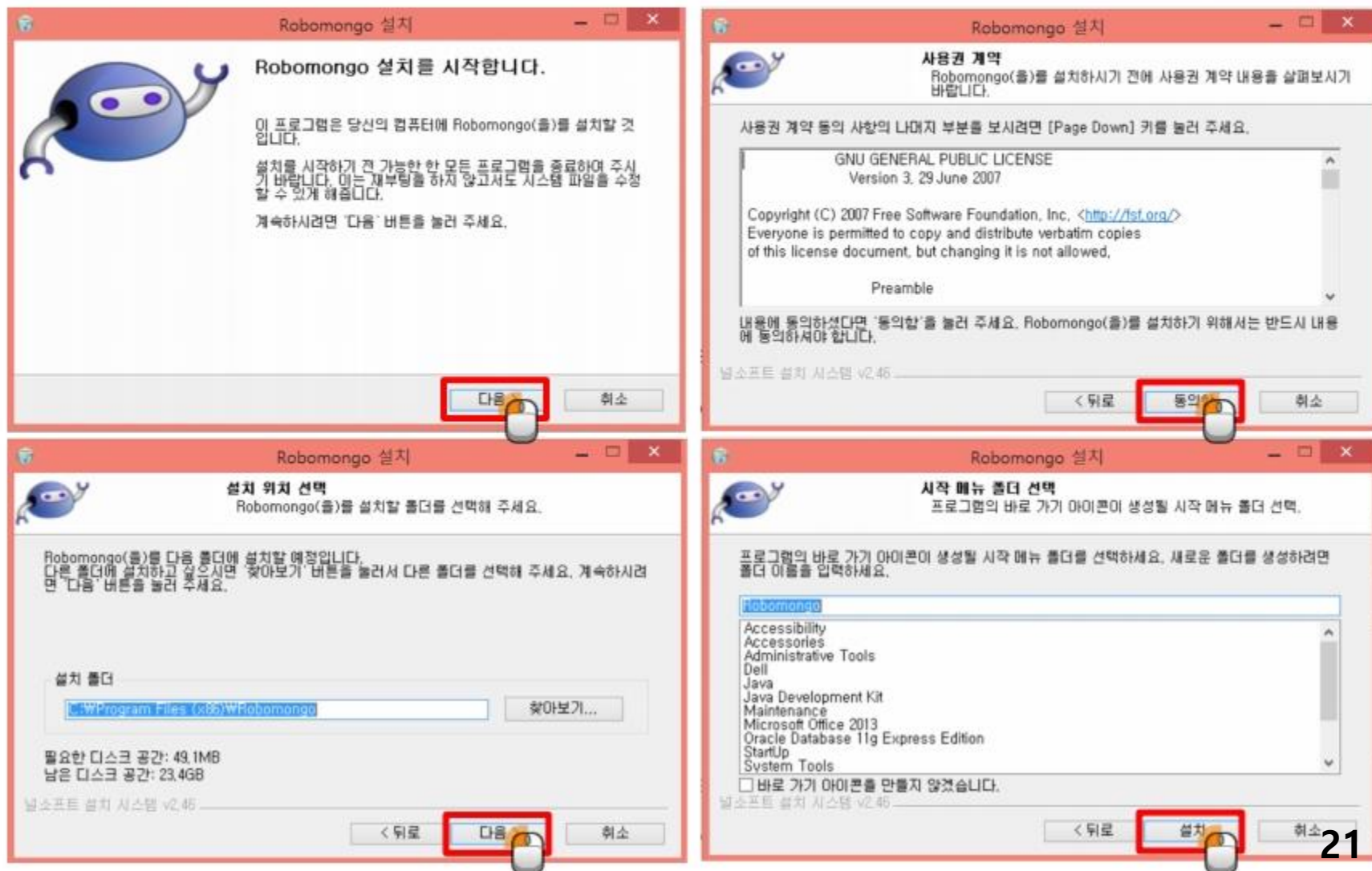
### ■ MongoDB 접속하기 – RoboMongo 사용하기

§ <http://robomongo.org>



## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속하기 - RoboMongo 사용하기



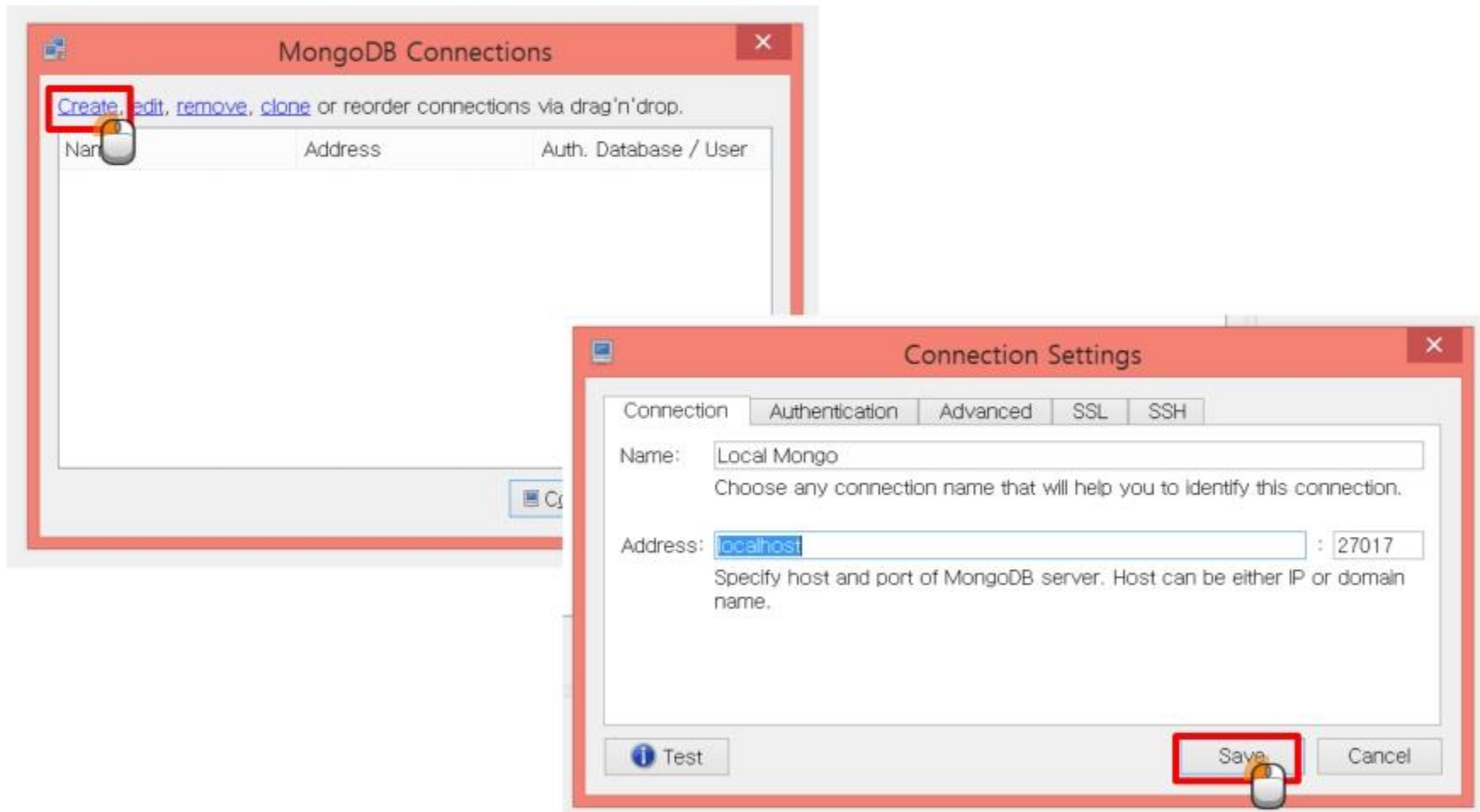
## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속하기 – RoboMongo 사용하기



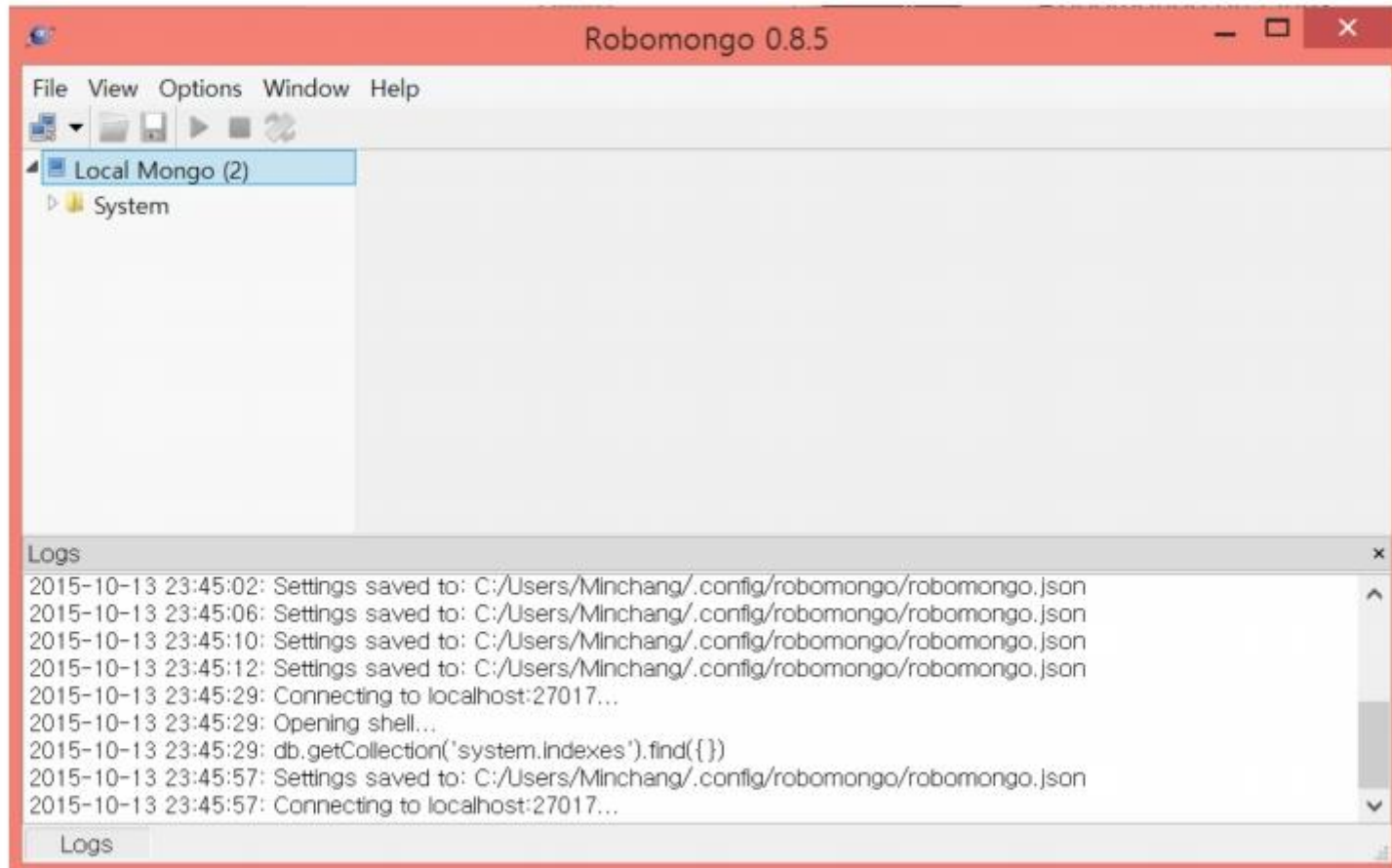
## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속하기 – RoboMongo 사용하기



## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속하기 – RoboMongo 사용하기





## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속하기 – Mongo Shell 이용하기

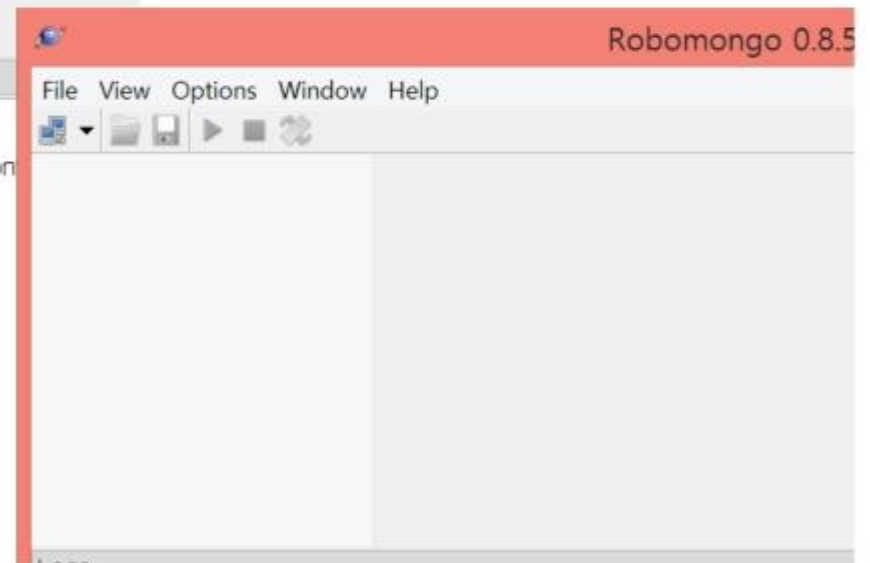
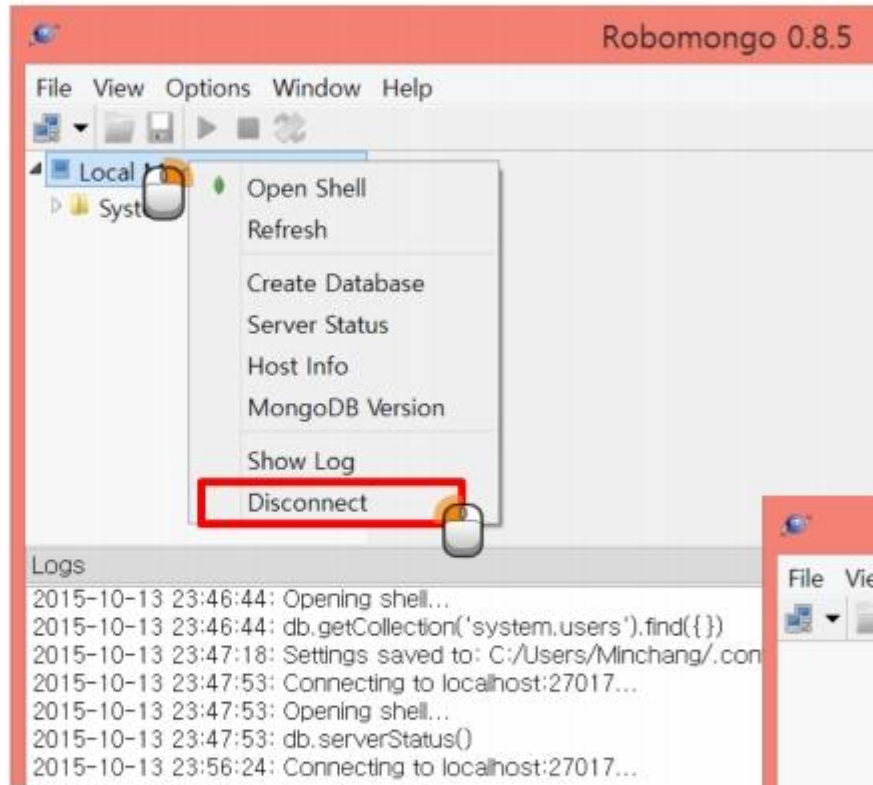
```
§ D:\mongodb-win32-x86_64-3.0.6\bin> mongo
§ MongoDB shell version: 3.0.6
§ connecting to: test
§ >
```



```
C:\Windows\system32\cmd.exe - mongo
D:\mongodb-win32-x86_64-3.0.6\bin>mongo
MongoDB shell version: 3.0.6
connecting to: test
>
```

## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속끊기 – RoboMongo 사용하기

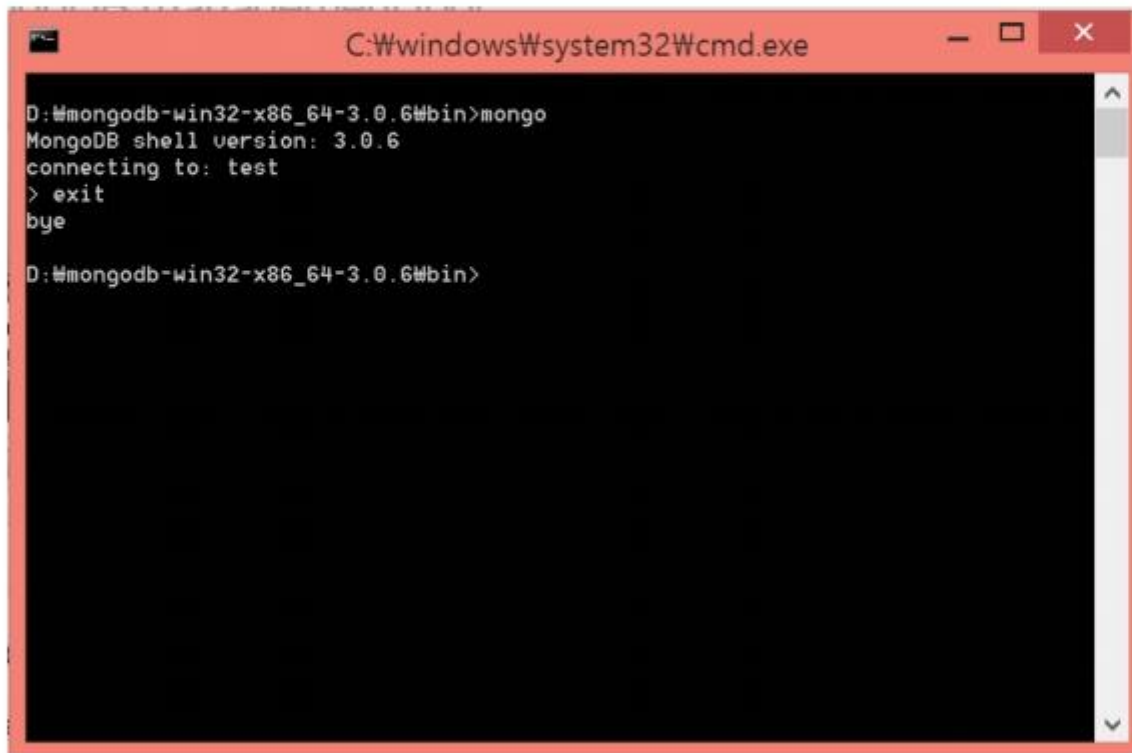


## 02 시작하기 - MongoDB 명령어

### ■ MongoDB 접속끊기 – Mongo Shell 이용하기

§ > **exit**

§ bye



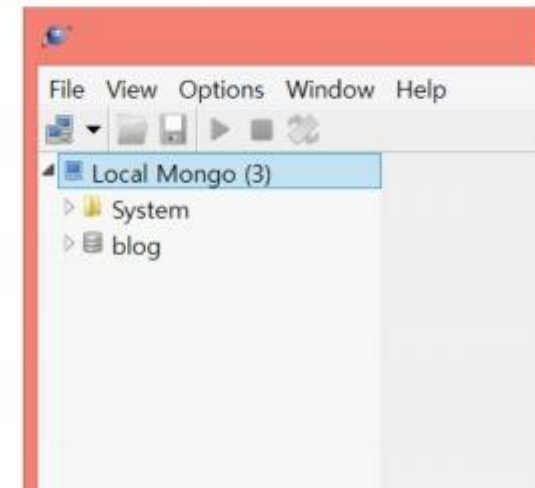
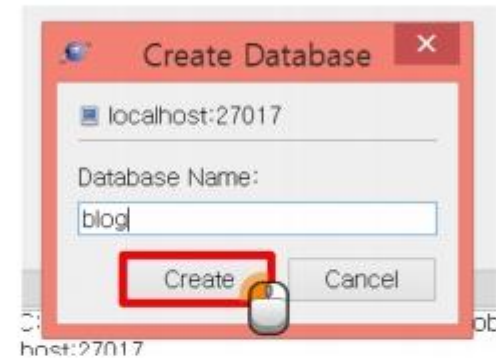
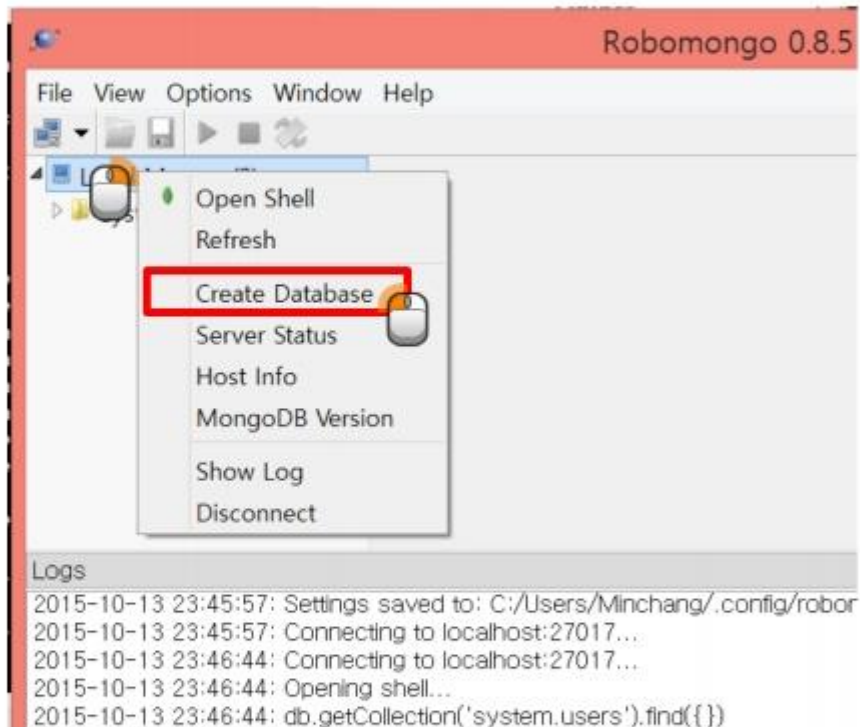
```
C:\Windows\system32\cmd.exe

D:\mongodb-win32-x86_64-3.0.6\bin>mongo
MongoDB shell version: 3.0.6
connecting to: test
> exit
bye

D:\mongodb-win32-x86_64-3.0.6\bin>
```

## 02 시작하기 - MongoDB 명령어

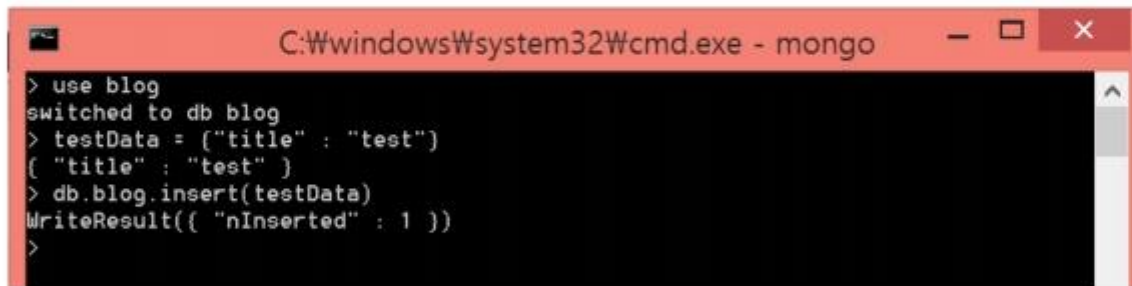
### ■ RoboMongo를 이용해 Database 만들기



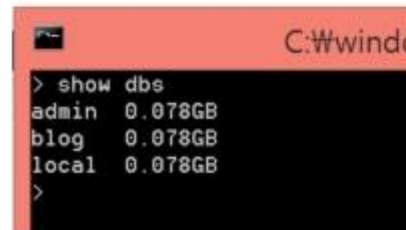
## 02 시작하기 - MongoDB 명령어

### ■ Mongo Shell을 이용해 Database 만들기

```
§ > use blog // blog db를 사용하도록 변경
§ switched to db blog
§ > testData = {"title" : "test"} // 테스트 데이터 생성
§ { "title" : "test" }
§ > db.blog.insert(testData) // blog db에 테스트 데이터 삽입 (이 때 DB가 만들어짐)
§ WriteResult({ "nInserted" : 1 })
§ >
```

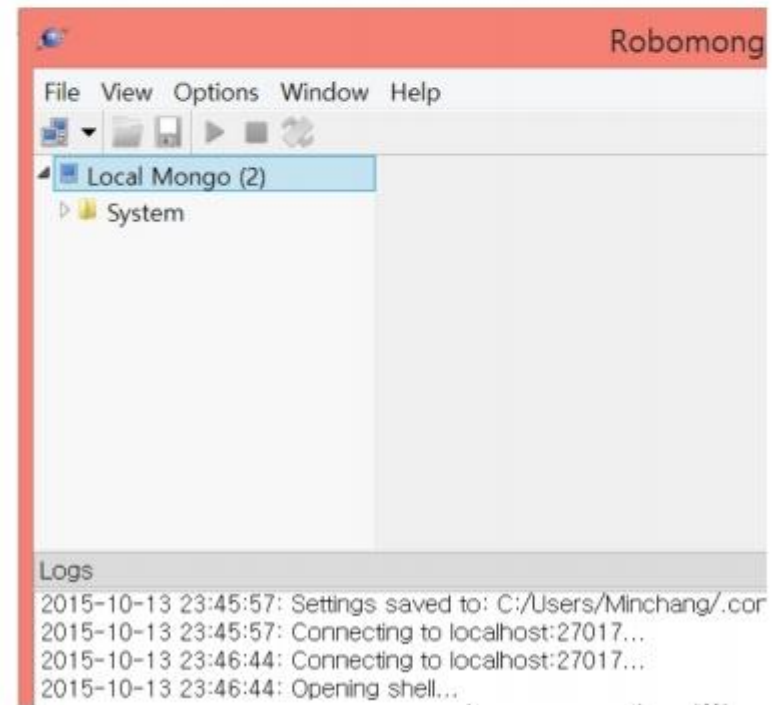
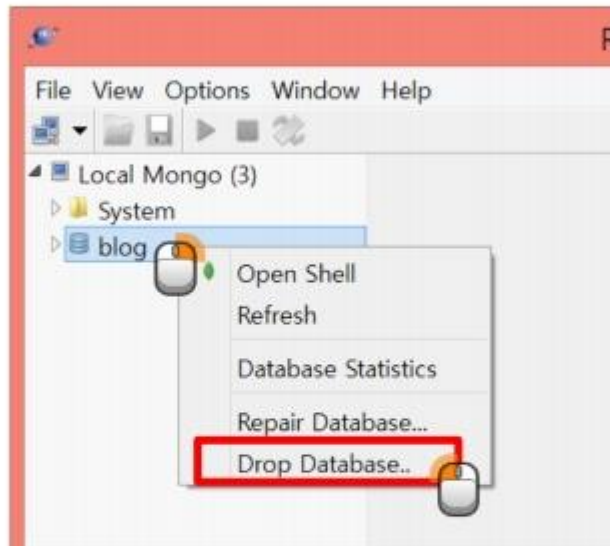


```
§ > show dbs // 만들어진 DB 목록을 출력
§ admin 0.078GB
§ blog 0.078GB
§ local 0.078GB
§ >
```



## 02 시작하기 - MongoDB 명령어

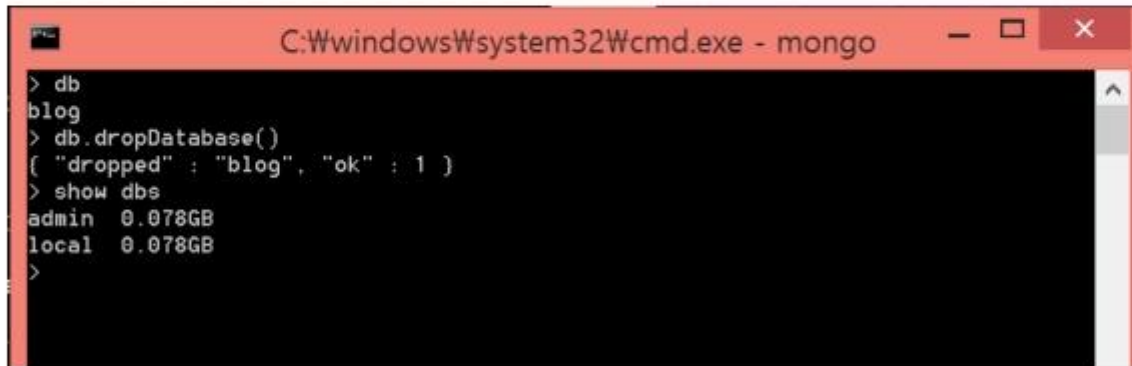
### ■ RoboMongo를 이용해 Database 삭제하기



## 02 시작하기 - MongoDB 명령어

### ■ Mongo Shell을 이용해 Database 삭제하기

```
> db
blog
> db.dropDatabase() // blog database를 삭제한다.
{ "dropped" : "blog", "ok" : 1 }
> show dbs
admin 0.078GB
local 0.078GB
>
```

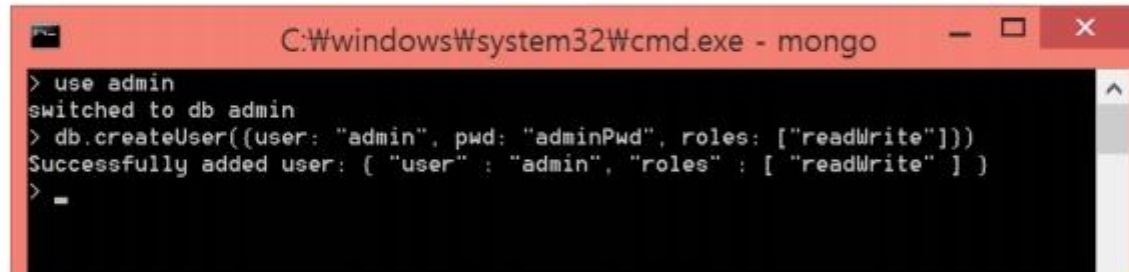


```
C:\Windows\System32\cmd.exe - mongo
> db
blog
> db.dropDatabase()
{ "dropped" : "blog", "ok" : 1 }
> show dbs
admin 0.078GB
local 0.078GB
>
```

## 02 시작하기 - MongoDB 명령어

### ■ Mongo Shell을 이용해 사용자 추가하기

- § > **use admin** // 계정을 만들 DB 선택
- § Switched to db admin
- § > **db.createUser({user: "admin", pwd: "adminPwd", roles: ["readWrite"]})**
- § Successfully added user: { "user " : "admin", "roles" : [ "readWrite" ] }
- § >



```
C:\Windows\system32\cmd.exe - mongo
> use admin
switched to db admin
> db.createUser({user: "admin", pwd: "adminPwd", roles: ["readWrite"]})
Successfully added user: { "user " : "admin", "roles" : [ "readWrite" ] }
>
```



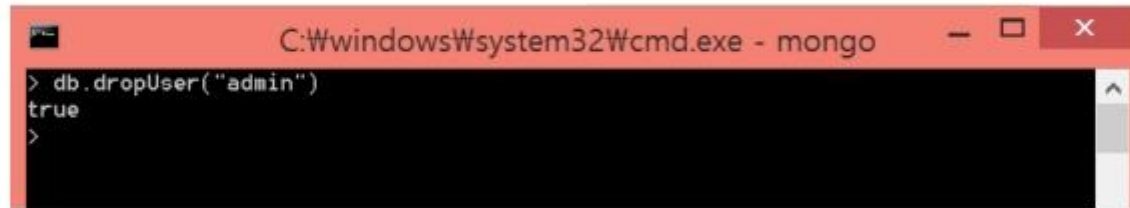
## 02 시작하기 - MongoDB 명령어

### ■ Mongo Shell을 이용해 사용자 삭제하기

§ > db.dropUser("admin")

§ true

§ >

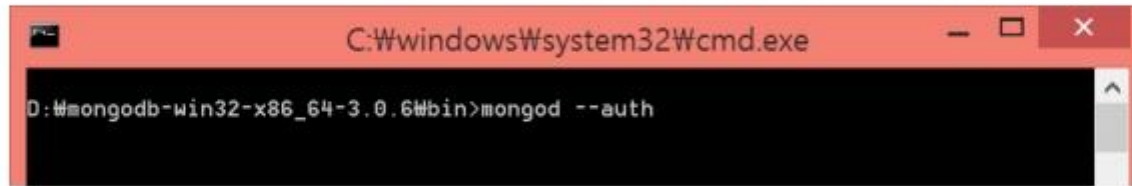


```
C:\Windows\system32\cmd.exe - mongo
> db.dropUser("admin")
true
>
```

## 02 시작하기 - MongoDB 명령어

### ■ Mongo Shell을 이용해 MongoDB 사용자 로그인하기

- § 1. blog db에 임의의 User 생성
- § 2. 기존에 동작중이던 MongoDB를 종료함.
- § 3. mongod -auth 옵션을 추가해 실행.



```
C:\Windows\system32\cmd.exe
D:\mongodb-win32-x86_64-3.0.6\bin>mongod --auth
```

- § 4. mongo 접속
  - D:\mongodb-win32-x86\_64-3.0.6\bin>mongo
  - MongoDB shell version: 3.0.6
  - connecting to: test
  - > use blog // 계정을 생성한 db로 이동.
  - Switched to db blog
  - > db.auth( " aaa1 " , " aaa1 " ) // 로그인 db.auth("id", "pwd")
  - 1 // 1은 성공, 0은 실패
  - >

## 02 시작하기 - MongoDB 명령어

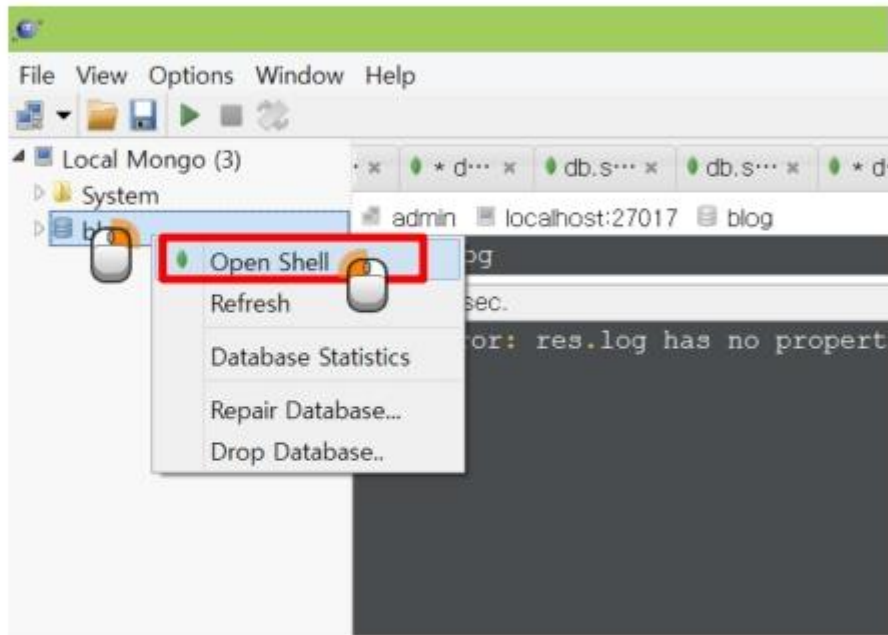
---

### ■ Robomongo를 이용해 MongoDB 사용자 로그인하기

- § Robomongo 0.8.x에서는 mongoDB 3.x 버전대의 사용자 로그인을 지원하지 않음.
- § 추후 배포될 Robomongo 0.9 버전에서 지원예정
- § 위 이유로 본 교재의 실습내용은 Spring-Data를 제외하고 사용자 인증을 하지 않을 채로 진행됨.

## 02 시작하기 - MongoDB 명령어

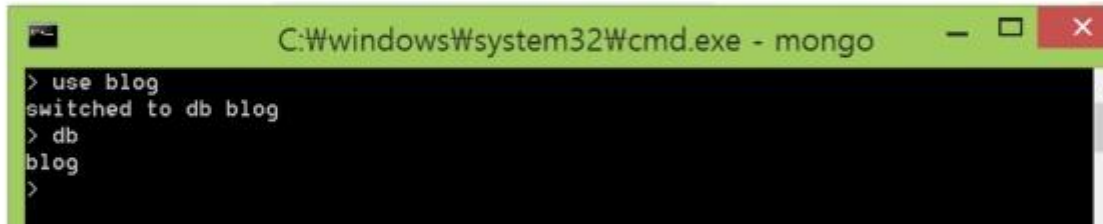
- Robomongo를 이용해 MongoDB Database 변경하기



## 02 시작하기 - MongoDB 명령어

### ■ Mongo Shell을 이용해 MongoDB Database 변경하기

```
§ > use blog  
§ switched to db blog  
§ > db  
§ blog  
§ >
```



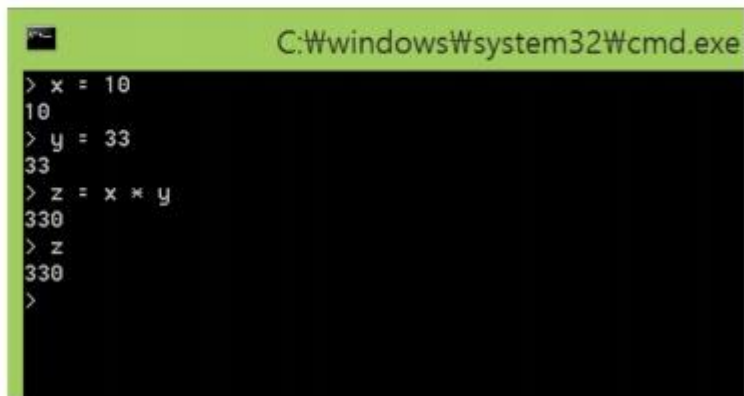
```
C:\Windows\system32\cmd.exe - mongo  
> use blog  
switched to db blog  
> db  
blog  
>
```

## 02 시작하기 - MongoDB 명령어

### ■ Javascript 수행하기

§ 단순 연산

- **> x = 10**
- 10
- **> y = 33**
- 33
- **> z=x\*y**
- 330
- **> z**
- 330
- >



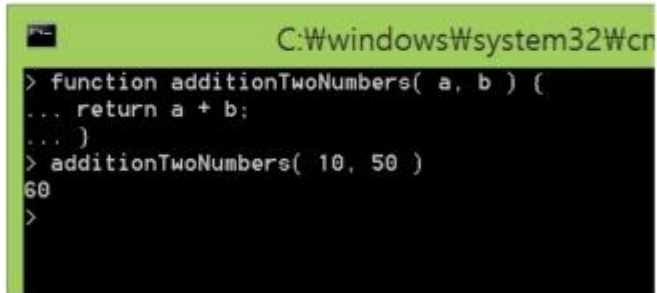
```
C:\Windows\system32\cmd.exe
> x = 10
10
> y = 33
33
> z = x * y
330
> z
330
>
```

## 02 시작하기 - MongoDB 명령어

### ■ Javascript 수행하기

§ function

- > **function** additionTwoNumbers( a, b ) {
- ... **return** a + b;
- ... }
- > **additionTwoNumbers( 10, 50 )**
- 60
- >



```
C:\Windows\system32\cmd.exe
> function additionTwoNumbers( a, b ) {
... return a + b;
... }
> additionTwoNumbers( 10, 50 )
60
>
```

컬렉션



## 03 시작하기 - 컬렉션

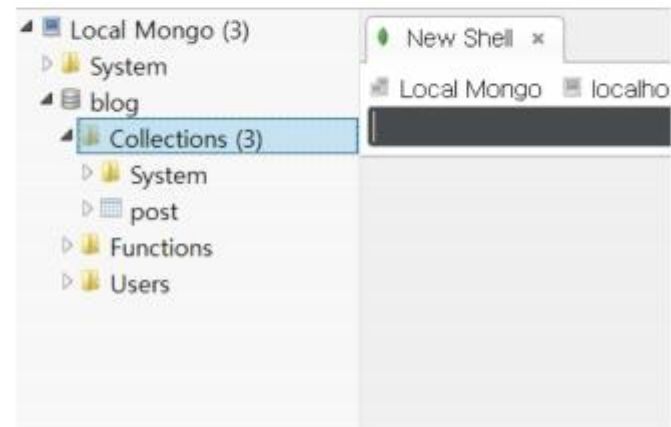
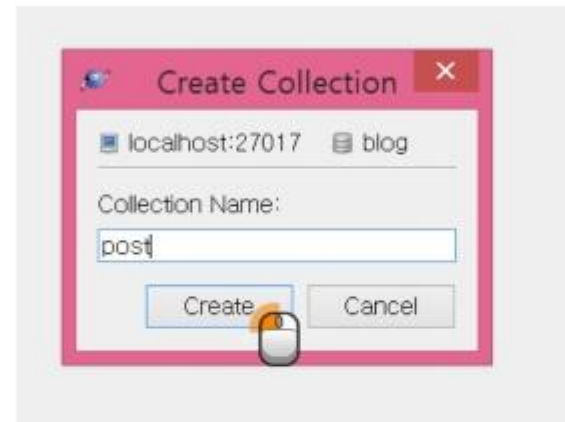
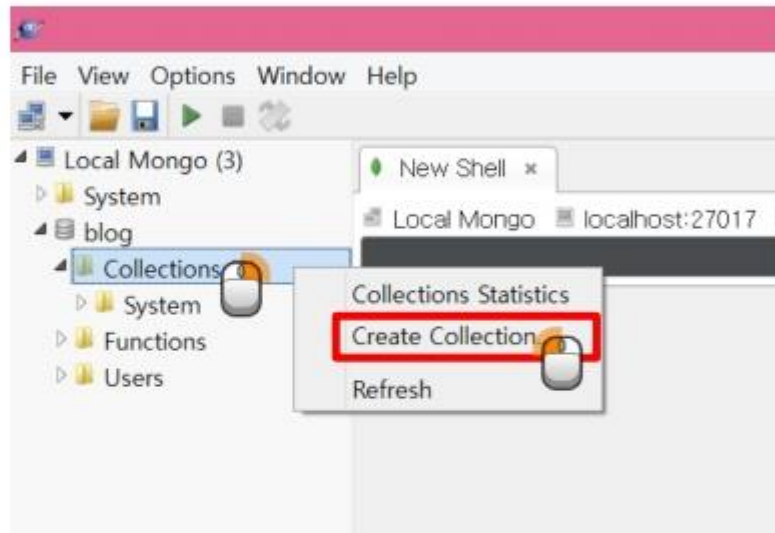
### ■ RDB에서 Table에 해당됨.

§ 여러 개의 문서(Document)로 구성되어 있음.

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

## 03 시작하기 - 컬렉션

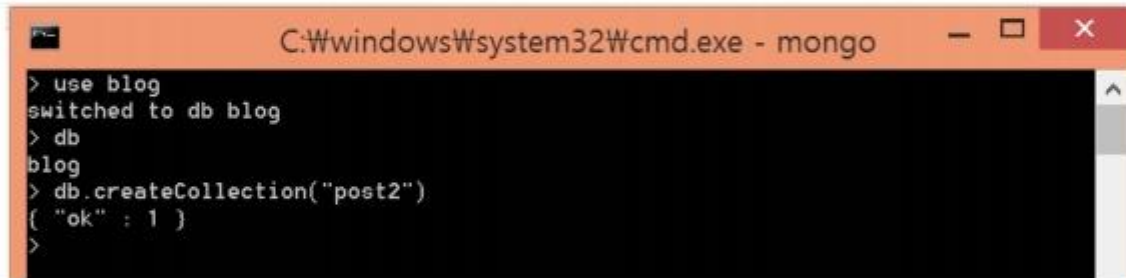
### ■ Robomongo를 이용해 컬렉션 만들기



## 03 시작하기 - 컬렉션

### ■ Mongo Shell을 이용해 컬렉션 만들기

```
> use blog
switched to db blog
> db
blog
> db.createCollection("post2")
{ "ok" : 1 }
>
```



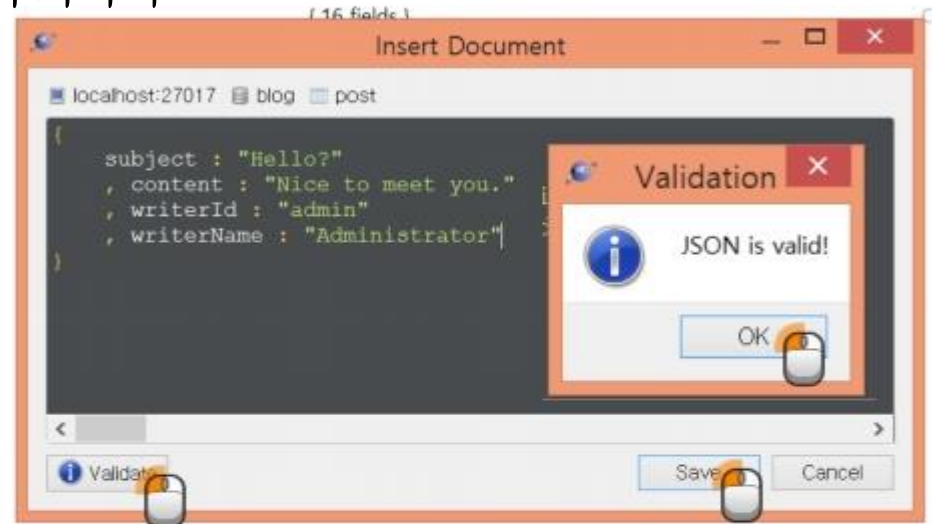
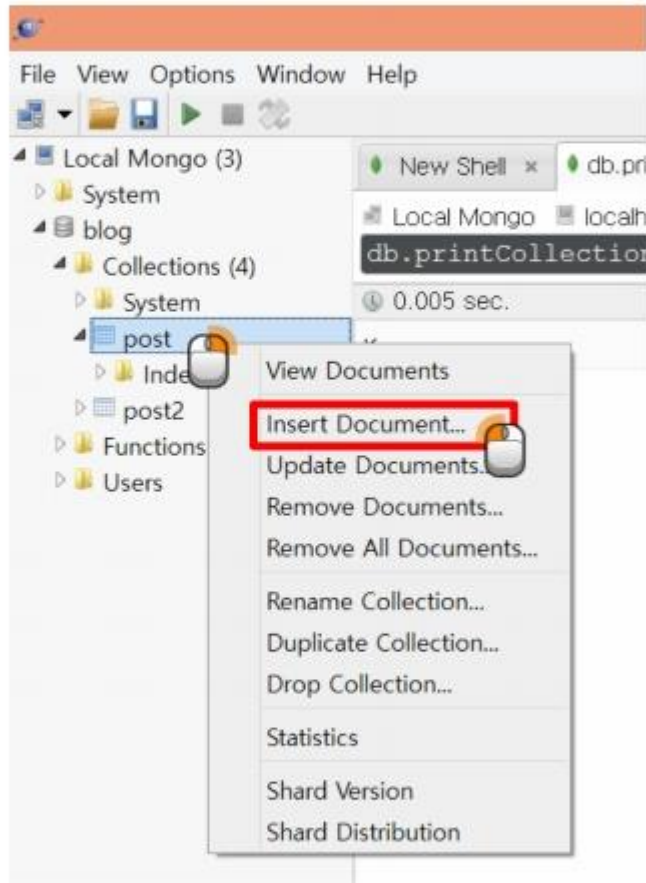
```
C:\windows\system32\cmd.exe - mongo
> use blog
switched to db blog
> db
blog
> db.createCollection("post2")
{ "ok" : 1 }
>
```

### ■ 컬렉션 네이밍

- § 컬렉션은 이름으로 식별함.
- § 빈 문자열은 사용할 수 없다.
- § **W0**(null 문자)은 쓸 수 없다.
- § **system.** 으로 시작하는 컬렉션이름은 사용할 수 없다. (예약어)
- § **\$** 를 쓸 수 없다.

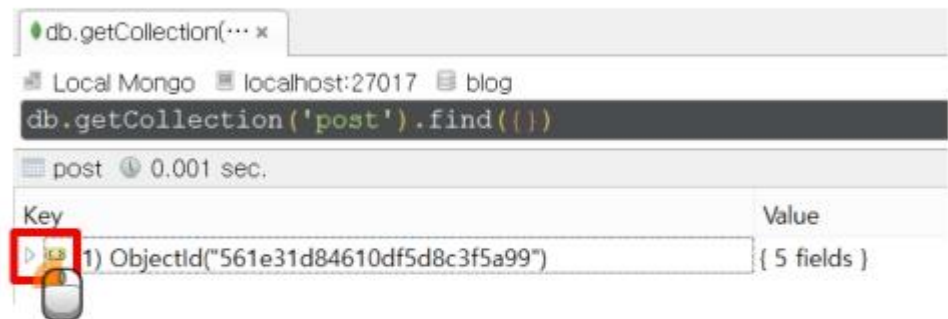
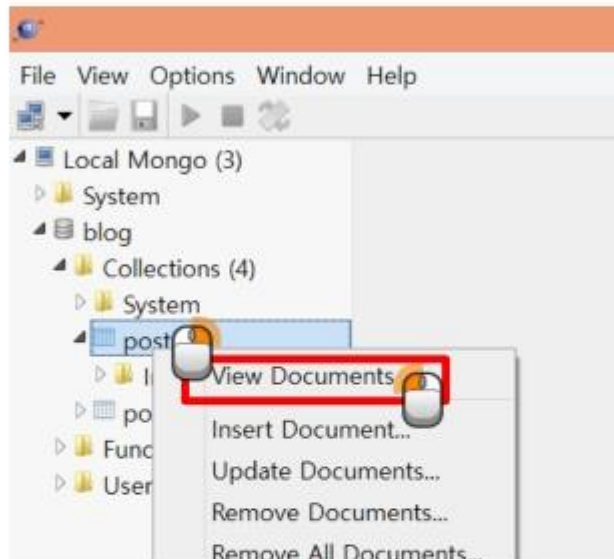
## 03 시작하기 - 컬렉션

### ■ Robomongo를 이용해 컬렉션에 문서 추가하기



## 03 시작하기 - 컬렉션

### ■ Robomongo를 이용해 컬렉션에 문서 추가하기

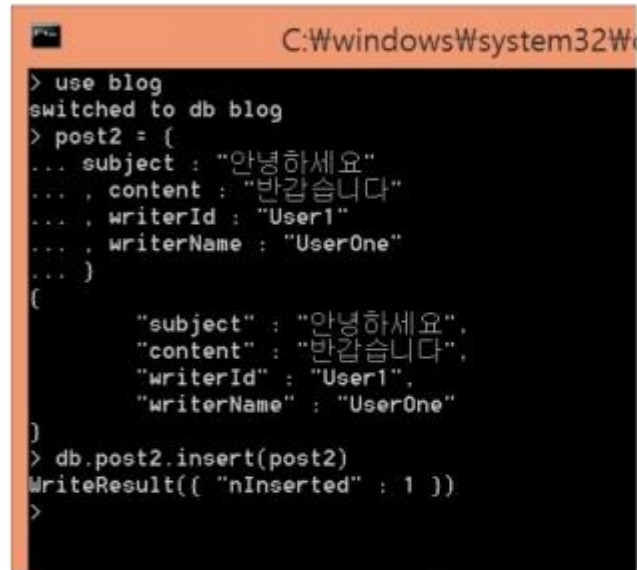


Key	Value	Type
(1) ObjectId("561e31d84610df5d8c3f5a99")	{ 5 fields }	Object
_id	ObjectId("561e31d84610df5d8c3f5a99")	ObjectId
subject	Hello?	String
content	Nice to meet you.	String
writerId	admin	String
writerName	Administrator	String

## 03 시작하기 - 컬렉션

### ■ Mongo Shell을 이용해 컬렉션에 문서 추가하기

```
&gt; use blog
switched to db blog
> post2 = {
... subject : "안녕하세요"
... , content : "반갑습니다"
... , writerId : "User1"
... , writerName : "UserOne"
... }
{
  "subject" : "안녕하세요",
  "content" : "반갑습니다",
  "writerId" : "User1",
  "writerName" : "UserOne"
}
> db.post2.insert(post2)
WriteResult({ "nInserted" : 1 })
>
```

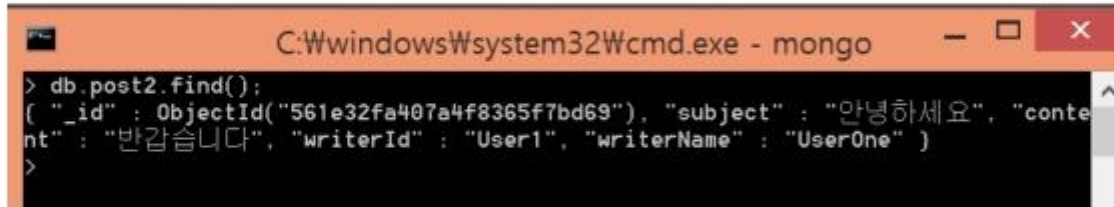


```
C:\Windows\system32\cmd.exe
> use blog
switched to db blog
> post2 = {
... subject : "안녕하세요"
... , content : "반갑습니다"
... , writerId : "User1"
... , writerName : "UserOne"
... }
{
  "subject" : "안녕하세요",
  "content" : "반갑습니다",
  "writerId" : "User1",
  "writerName" : "UserOne"
}
> db.post2.insert(post2)
WriteResult({ "nInserted" : 1 })
>
```

## 03 시작하기 - 컬렉션

### ■ Mongo Shell을 이용해 컬렉션에 문서 추가하기

```
> db.post2.find();  
{ "_id" : ObjectId("561e32fa407a4f8365f7bd69"), "subject" : "안녕하세요",  
  "content" : "반갑습니다", "writerId" : "User1", "writerName" : "UserOne" }  
>
```

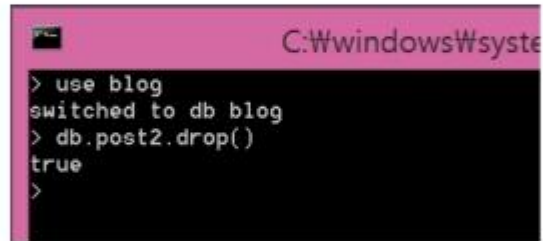


```
C:\Windows\system32\cmd.exe - mongo  
> db.post2.find();  
{ "_id" : ObjectId("561e32fa407a4f8365f7bd69"), "subject" : "안녕하세요", "content" : "반갑습니다", "writerId" : "User1", "writerName" : "UserOne" }  
>
```

## 03 시작하기 - 컬렉션

### ■ Mongo Shell을 이용해 컬렉션 삭제하기

```
§ > use blog  
§ switched to db blog  
§ > db.post2.drop()  
§ true  
§ >
```



A screenshot of a Windows command prompt window with a pink title bar. The title bar text is partially visible as 'C:\Windows\system...'. The command prompt shows the following text:

```
> use blog  
switched to db blog  
> db.post2.drop()  
true  
>
```



문서

## 04 시작하기 - 문서

- RDB에서 ROW에 해당됨.
  - § 테이블은 **Collection** 이라 부름.
  - § Row의 컬럼들은 **Field** 라고 부름.

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

## 04 시작하기 - 문서

### ■ RDB

§ **Table**은 저장되는 데이터의 형태가 고정되어 있음.

§ 예>

ARTICLE						
ARTICLE_ID	SUBJECT	CONTENT	WRITER	HIT	CRT_DT	MDFY_DT
AR-20150101-000001	안녕하세요?	처음 뵙겠습니다.	장민창	10	2015-10-14	2015-10-14
AR-20150101-000002	매력적인 MongoDB	Spring을 활용하면 더욱 멋지죠!	장민창	1	2015-10-14	2015-10-14

§ 심지어, 컬럼에 저장되는 데이터의 크기마저도 정해져 있음.

§ 테이블에 컬럼을 추가 할 경우 대량의 작업이 필요할 수도 있다.

§ **MySQL**의 경우 대용량 **Table**에 컬럼을 추가할 경우 엄청난 시간이 소요될 수 있다.

§ **Table Join**으로 발생하는 쿼리 지연이 발생할 수 있다.

### ■ MongoDB

§ **MongoDB**의 데이터는 **JSON** 형태로 저장돼 데이터의 형태가 자유롭다.

§ 문서에 저장되는 데이터의 크기가 정해져 있지 않다.

§ 문서에 필드를 추가해도 대량의 작업과 많은 시간이 소요되지 않는다. (즉시 반영)

## 04 시작하기 - 문서

### ■ MongoDB

- § RDB의 무결성 제약의 원칙이 무시됨.
- § RDB는 **Primary Key(PK)**를 "고유값" 을 가진 컬럼에 지정함.
  - → PK <-> FK로 중복을 최소화 한다.
- § MongoDB는 고유한 값을 자동으로 만들어 냄.
  - → 중복을 허용하고 적극적으로 활용한다.
    - RDB처럼 JOIN을 이용해 데이터를 가져올 필요가 없음.
    - 예 > 정규화 하지 않은 Table – Non Atomic Data
- § RDBMS 는 정형화된 테이블에 동일한 정보들을 넣고, 다른 정보가 필요할 때 JOIN을 한다.
  - 블로그 포스팅을 저장하는 테이블은
    - 글쓴이, 제목, 글쓴 날짜, 내용 으로 구성되고, 댓글이나 태그 등은 다른 테이블에 저장한다.
- § NoSQL 은 비정형화된 컬렉션에 관련된 정보들을 넣고, 다른 정보가 필요할 때 다른 문서를 찾는다.
  - 블로그 포스팅을 저장하는 컬렉션은
    - 글쓴이, 제목, 글쓴 날짜, 내용, 댓글, 태그 까지 모두 포함할 수 있다.

## 04 시작하기 - 문서

### ■ MongoDB

§ 문서가 JSON으로 저장되는 예



The screenshot shows the MongoDB Shell interface. At the top, it displays 'Local Mongo', 'localhost:27017', and 'admin'. The command entered is `db.getCollection('system.indexes').find({})`. Below the command, it shows the collection 'system.indexes' and the execution time '0.001 sec.'. The output consists of two JSON documents, each enclosed in a comment block `/* 1 */` and `/* 2 */`. Both documents have a version number of 1 and a key of `{ "_id" : 1 }`. The first document's name is `"_id_"` and its namespace is `"admin.system.version"`. The second document's name is `"_id_"` and its namespace is `"admin.system.users"`.

```
Local Mongo localhost:27017 admin
db.getCollection('system.indexes').find({})
system.indexes 0.001 sec.

/* 1 */
{
  "v" : 1,
  "key" : {
    "_id" : 1
  },
  "name" : "_id_",
  "ns" : "admin.system.version"
}

/* 2 */
{
  "v" : 1,
  "key" : {
    "_id" : 1
  },
  "name" : "_id_",
  "ns" : "admin.system.users"
}
```

## 04 시작하기 - 문서

### ■ MongoDB의 문서 형식

§ JSON 형태로 데이터가 저장됨.

§ 아래 문서 두 개는 **Field**의 개수가 다르므로 서로 다른 문서가 됨.

```
{"greeting" : "Hello, World"}
```

```
{"greeting" : "Hello, World", "foo" : 3}
```

§ 동일한 **Field**를 가지더라도 순서가 다르다면, 서로 다른 문서가 됨.

```
{"greeting" : "Hello, World", "foo" : 3}
```

```
{"foo" : 3, "greeting" : "Hello, World"}
```

§ 문서의 **Key**는 문자열로 하고, **Value**는 어느 값이든 쓸 수 있다.

§ 단, **Key**는 예약어를 제외하고 모두 사용할 수 있다.

- **Key**는 **W0**(null문자)을 포함하지 않는다.
- . 과 \$ 문자는 사용할 수 없다.
- \_로 시작하는 문자는 사용하지 않는다. (예약어일 가능성 높음)

## 04 시작하기 - 문서

### ■ MongoDB의 문서 형식

§ 대소문자 및 데이터형을 정확히 구분하며, 다를 경우 서로 다른 문서로 인식한다.

- 데이터형이 다른 경우

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

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

- 대/소문자가 다른 경우

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

```
{"Foo" : 3}
```

§ **Key** 가 중복될 수 없다.

```
{  
  "greeting" : "Hello, World",  
  "greeting" : "Hello, MongoDB"  
}
```

# 데이터 형 (Type)



## 05 시작하기 - 데이터 형(Type)

- 일반 JSON Type의 Data Type 을 그대로 유지하면서 추가적인 데이터형을 지원함.

요소명		Data Type	
기능		포맷	설명
	null	<code>{"x" : null}</code>	null값과 존재하지 않는 Field를 표현하는데 사용.
	불린	<code>{"x" : true}</code>	참과 거짓을 구분할 때 사용.
숫자	실수	<code>{"x" : 3.14}</code>	숫자는 8바이트 부동소수점이 기본 형
	정수	<code>{"x" : 3}</code>	일반적인 정수도 8바이트 부동소수점을 사용함.
		<code>{"x" : NumberInt("3")}</code>	4바이트 정수 표현
		<code>{"x" : NumberLong("3")}</code>	8바이트 정수 표현
	문자열	<code>{"x" : "foobar"}</code>	UTF-8 문자열을 표현할 때 사용.
	날짜	<code>{"x" : new Date()}</code>	1970년 1월 1일부터의 시간을 1/1000 초 단위로 저장
	정규표현식	<code>{"x" : /foobar/i}</code>	자바스크립트 정규표현식 문법사용이 가능함.
	배열	<code>{"x" : ["a", "b", "c"]}</code>	값의 셋트나 리스트를 배열로 표현
	내장 문서	<code>{"x" : {"foo" : "bar"}}</code>	문서는 다른 문서를 포함할 수도 있음.
	객체 ID	<code>{"x" : ObjectId()}</code>	문서용 12바이트 ID. RDB의 PK와 같은 개념
	코드	<code>{"x" : function() { /* ..... */ }}</code>	임의의 코드를 포함 할 수도 있음.

**1-03**

문서의 생성, 갱신, 삭제

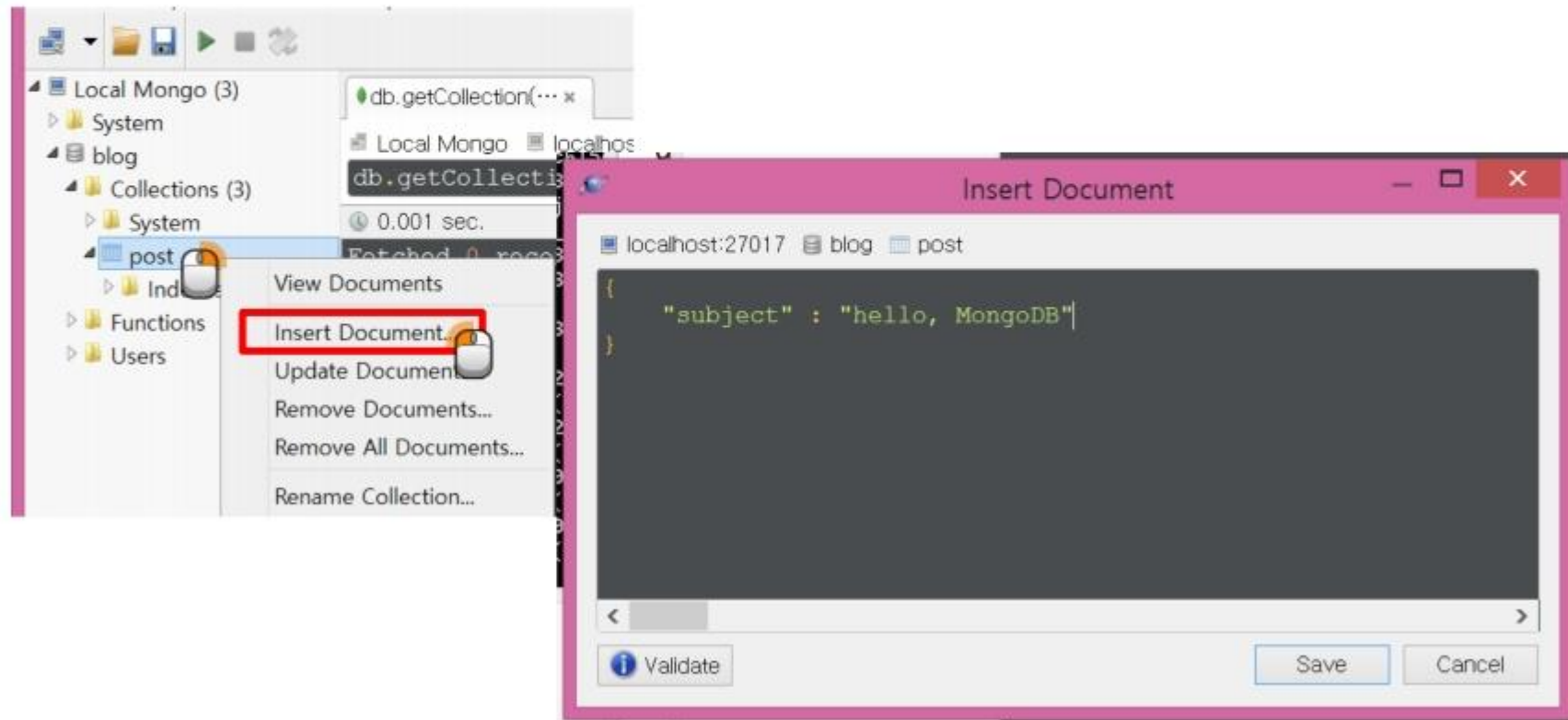
# 문서의 삽입과 저장

# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ INSERT

```
db.foo.insert({"bar" : "baz"})
```

§ MongoDB에 데이터를 추가하는 기본 적인 방법 - 1 RoboMongo

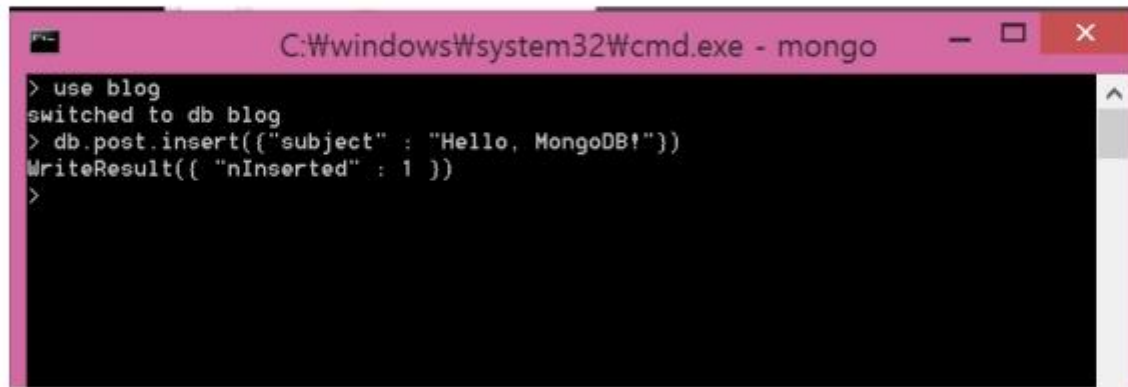


# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ INSERT

```
db.foo.insert({"bar" : "baz"})
```

§ MongoDB에 데이터를 추가하는 기본 적인 방법 - 2 Mongo Shell



```
C:\Windows\system32\cmd.exe - mongo
> use blog
switched to db blog
> db.post.insert({"subject" : "Hello, MongoDB!"})
WriteResult({ "nInserted" : 1 })
>
```

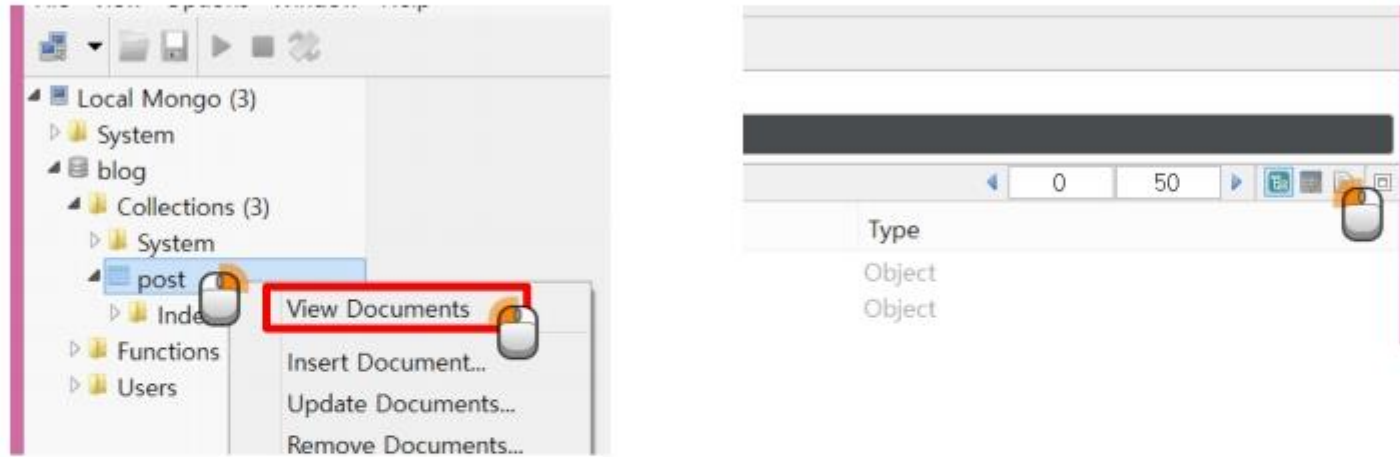
- > use blog
- switched to db blog
- > **db.post.insert({"subject" : "Hello, MongoDB!"})**
- WriteResult({ "nInserted" : 1 })
- >




# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ INSERT

§ MongoDB에 Data를 Insert 할 경우 문서의 고유한 Key인 Object ID 가 자동으로 생성됨.



Type
Object
Object

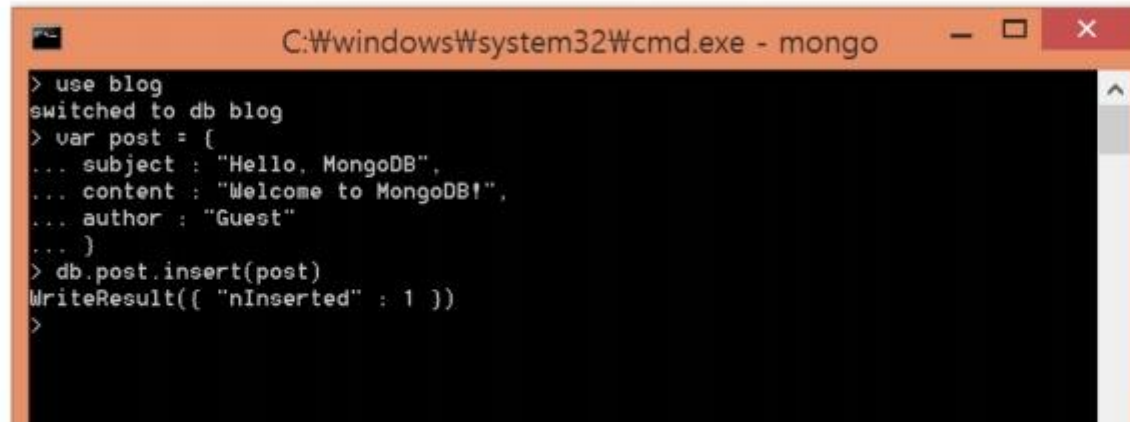
  


```
post 0.001 sec.
/* 1 */
{
  "_id" : ObjectId("5621dbad33725e07cc0ef7ad"),
  "subject" : "Hello, MongoDB!"
}
/* 2 */
{
  "_id" : ObjectId("5621dc1b4bfbaed505e6bcbf"),
  "subject" : "hello, MongoDB"
```

# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ INSERT

§ JavaScript 객체를 통해 Insert 하기



```
> use blog
switched to db blog
> var post = {
...  subject : "Hello, MongoDB",
...  content : "Welcome to MongoDB!",
...  author : "Guest"
... }
> db.post.insert(post)
WriteResult({ "nInserted" : 1 })
>
```

§ > use blog  
§ switched to db blog  
§ > var post = {  
§ ... subject : "Hello, MongoDB",  
§ ... content : "Welcome to MongoDB!",  
§ ... author : "Guest"  
§ ... }  
§ > db.post.insert(post)  
§ WriteResult({ "nInserted" : 1 })  
§ >

# 문서의 삭제

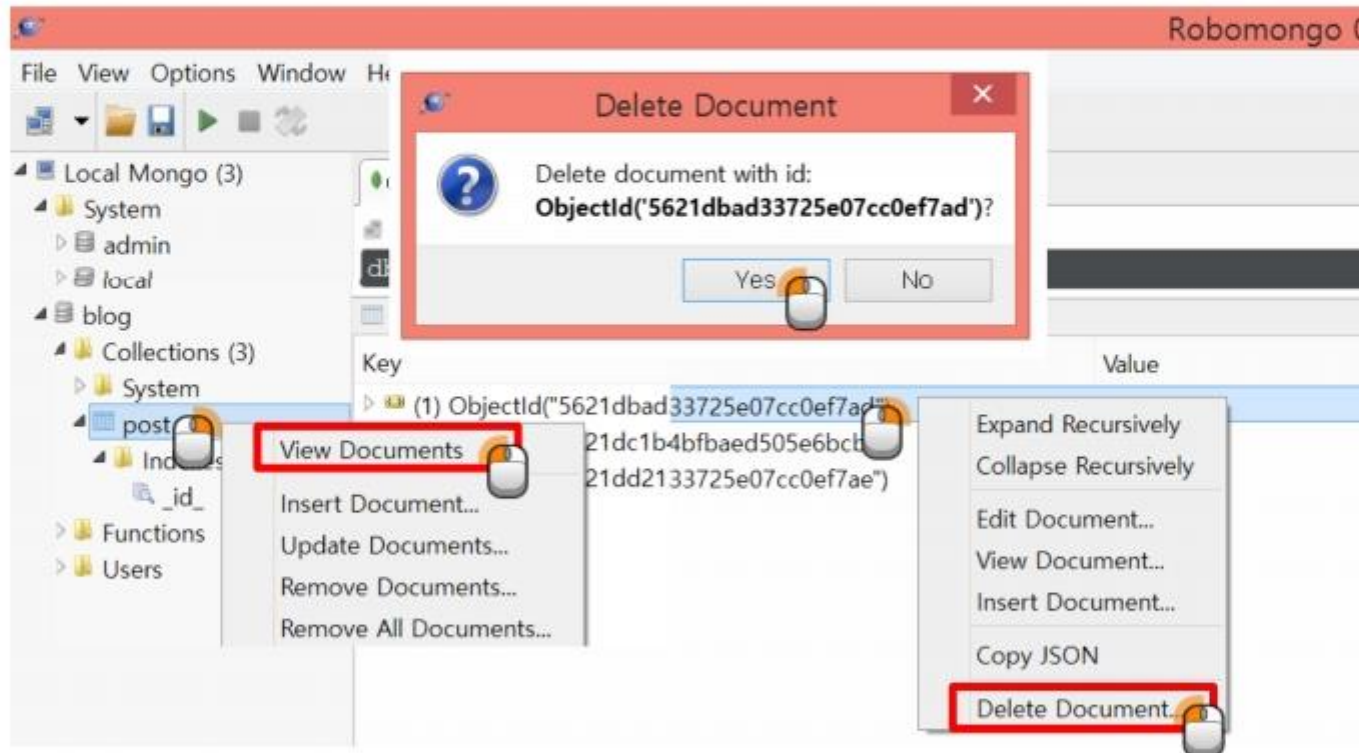


# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ REMOVE

§ MongoDB에서 데이터를 삭제하는 기본적인 방법 - 1 RoboMongo

- 선택 삭제

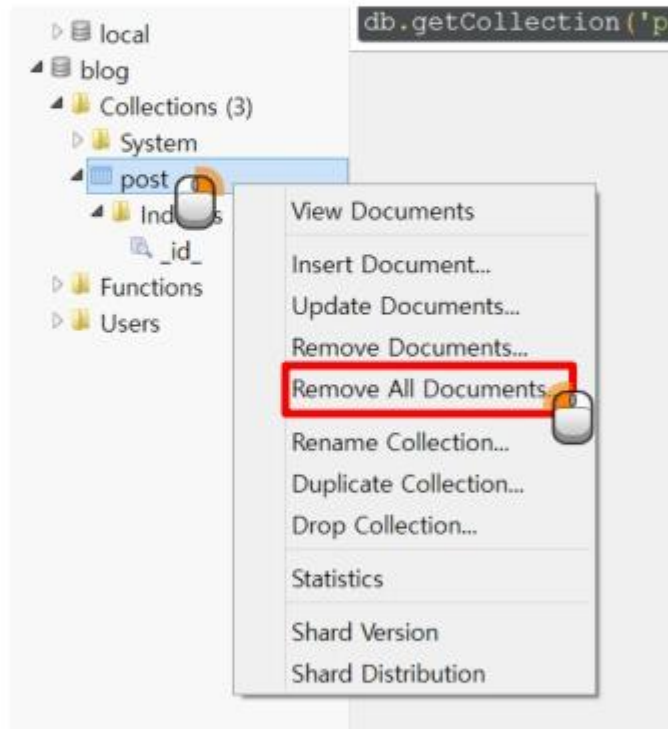


# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ REMOVE

§ MongoDB에 데이터를 추가하는 기본적인 방법 - 1 RoboMongo

- 모두 삭제

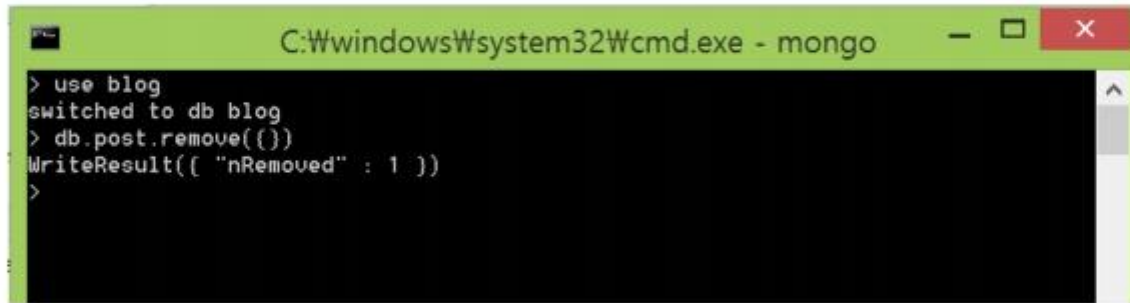


# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ REMOVE

§ MongoDB에 데이터를 추가하는 기본적인 방법 - 2 Mongo Shell

- 모두 삭제



```
C:\Windows\system32\cmd.exe - mongo
> use blog
switched to db blog
> db.post.remove({})
WriteResult({ 'nRemoved' : 1 })
>
```

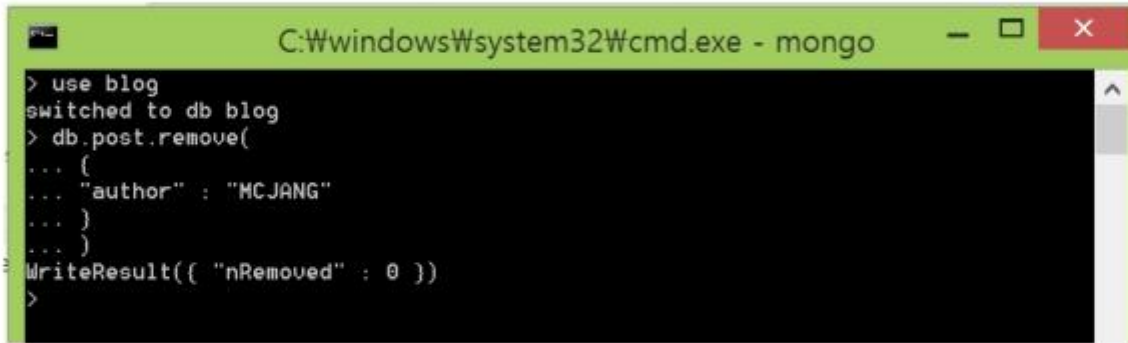
- > use blog
- switched to db blog
- > **db.post.remove({})**
- WriteResult({ 'nRemoved' : 1 })
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 삽입과 저장

## ■ REMOVE

§ MongoDB에 데이터를 추가하는 기본적인 방법 - 2 Mongo Shell

- 선택 삭제



```
C:\windows\system32\cmd.exe - mongo
> use blog
switched to db blog
> db.post.remove(
... {
... "author" : "MCJANG"
... }
... )
WriteResult({ "nRemoved" : 0 })
>
```

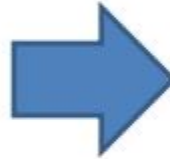
- > use blog
- switched to db blog
- > db.post.remove(
- ... {
- ... "author" : "MCJANG"
- ... }
- ... )
- WriteResult({ "nRemoved" : 0 })
- >

# 문서의 갱신

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ 문서 구조 변경 사례

```
{
  "_id" : ObjectId("562448800bcfc3fab5842c70"),
  "subject" : "Hello, MongoDB",
  "content" : "MongoDB is awesome!",
  "author" : "mcjang",
  "tags" : [
    "mongoDB",
    "awesone",
    "robomongo"
  ]
}
```

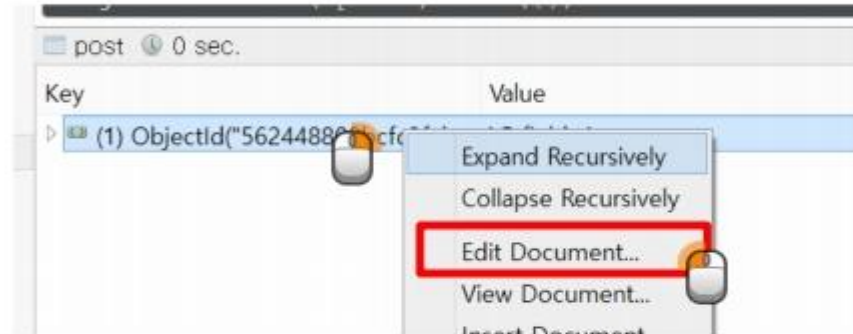
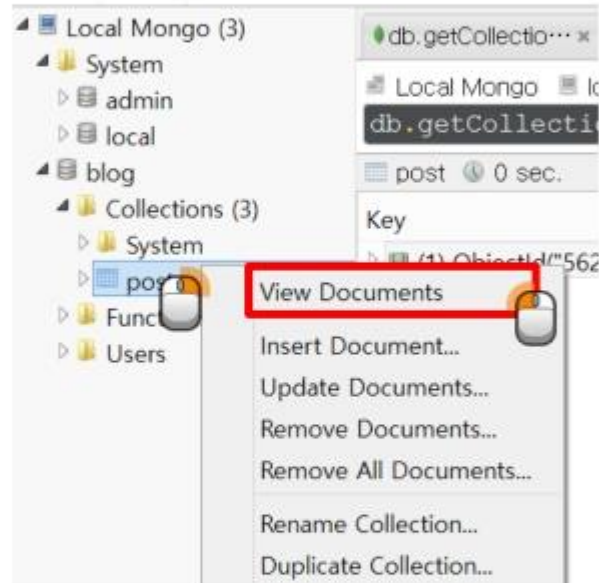


```
{
  "_id" : ObjectId("562448800bcfc3fab5842c70"),
  "subject" : "Hello, MongoDB",
  "content" : "MongoDB is awesome!",
  "author" : "mcjang",
  "tags" : [
    "mongoDB",
    "awesone",
    "robomongo"
  ],
  "replies" : [
    {
      "author" : "guest",
      "content" : "Is it fast?"
    },
    {
      "author" : "mcjang",
      "content" : "Yeah!"
    }
  ]
}
```

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ 문서 구조 변경 사례

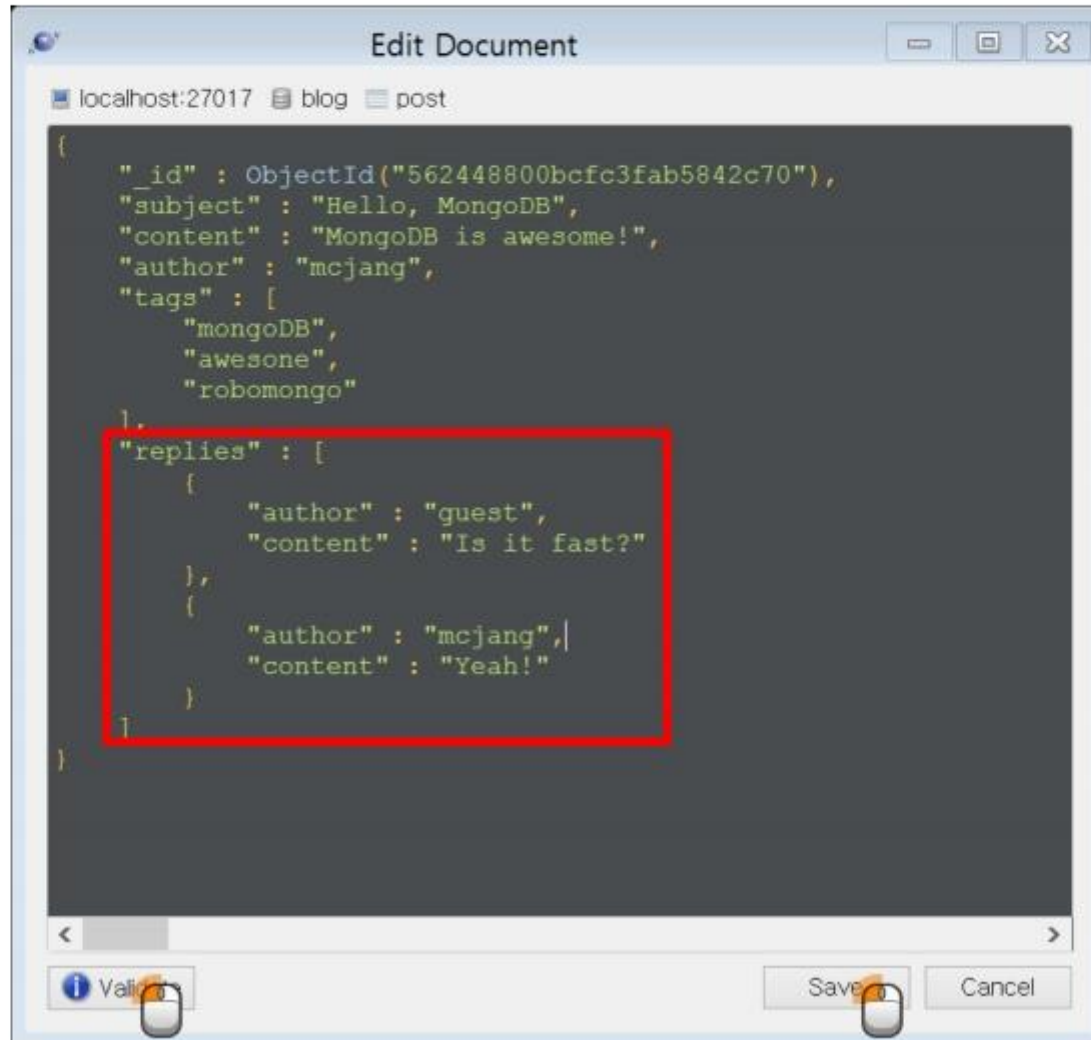
§ Robomongo를 이용한 구조 변경



# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ 문서 구조 변경 사례

§ Robomongo를 이용한 구조 변경

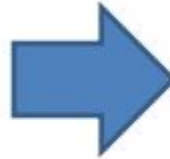




# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ 문서 구조 변경 사례

```
{
  "_id" : ObjectId("562448800bcfc3fab5842c70"),
  "subject" : "Hello, MongoDB",
  "content" : "MongoDB is awesome!",
  "author" : " guest",
  "tags" : [
    "mongoDB",
    "awesone",
    "robomongo"
  ]
}
```



```
{
  "_id" : ObjectId("562448800bcfc3fab5842c70"),
  "subject" : "Hello, MongoDB",
  "content" : "MongoDB is awesome!",
  "author" : "mcjang",
  "tags" : [
    "mongoDB",
    "awesone",
    "robomongo"
  ],
  "replies" : [
    {
      "author" : "guest",
      "content" : "Is it fast?"
    },
    {
      "author" : "mcjang",
      "content" : "Yeah!"
    }
  ]
}
```

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ 문서 구조 변경 사례

§ Mongo Shell을 이용한 구조 변경

§ > **db.post.findOne({"author" : "guest"})**

```
§ {  
§   "_id" : ObjectId("56258b770bcfc3fab5842c71"),  
§   "subject" : "Hello, MongoDB",  
§   "content" : "MongoDB is awesome!",  
§   "author" : "guest",  
§   "tags" : [  
§       "mongoDB",  
§       "awesone",  
§       "robomongo"  
§   ]  
§ }
```

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ 문서 구조 변경 사례

§ Mongo Shell을 이용한 구조 변경

§ > **var post = db.post.findOne({"author" : "guest"})**

§ > **post.replies = [{"author" : "guest", "content" : "It is fast?"}, {"author" : "mcjang", "content" : "yeah!"}]**

§ [

§ {

§     "author" : "guest",

§     "content" : "It is fast?"

§ },

§ {

§     "author" : "mcjang",

§     "content" : "yeah!"

§ }

§ ]

§ > **db.post.update({"author" : "guest"}, post)**

§ WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

§ >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$inc 제한자

§ 값 증가: \$inc

§ 사용 용도: 게시글 조회수 증가, 사용자 Point 증감, 시퀀스 등등

§ 시퀀스 만들기

- > **db.createCollection("seq")** // 시퀀스 문서만 담을 컬렉션 생성
- { " ok " : 1 }
- > **var seq = {"name" : "post", "val " : 0}** // 시퀀스 객체 생성
- > **db.seq.insert(seq)** // 시퀀스 문서 생성
- **WriteResult({ "nInserted" : 1 })**
- >

§ 시퀀스 값 증가시키기

- > **db.seq.update({"name" : "post"}, {"\$inc" : {"val" : 1}})** // \$inc 제한자를 이용해 값 1 증가시키기
- **WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })**
- >

§ 증가값 가져오기

- > **db.seq.findOne({"name" : "post"})**
- { "\_id" : ObjectId("56258f1ec8677ca2752803cf"), "name" : "post", "val" : 1 }
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$set 제한자

§ 값 증가: \$set

§ 사용 용도: 필드 추가(필드가 없을 경우) 및 필드 값 수정(필드가 존재할 경우)

§ 사용자의 "최근에 본 영화" 필드 삽입

§ user 컬렉션 생성

- > use blog
- switched to db blog
- > **db.createCollection("user")**
- { "ok" : 1 }
- >

§ 사용자 추가

- > **var user = { "id" : "mcjang", "password" : "1234", "name" : "MCJANG" }**
- > **db.user.insert(user)**
- **WriteResult({ "nInserted" : 1 })**
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$set 제한자

§ “최근에 본 영화” 추가 (추가 입력은 불가능)

- > **db.user.update**({**"id"** : **"mcjang"**}, {**"\$set"** : {**"latestSeenMovie"** : **"추격자"**}})
- **WriteResult**({ **"nMatched"** : 1, **"nUpserted"** : 0, **"nModified"** : 1 })
- > **db.user.findOne**({**"id"** : **"mcjang"**})
- {
  - **"\_id"** : ObjectId(**"56259382c8677ca2752803d0"**),
  - **"id"** : **"mcjang"**,
  - **"password"** : **"1234"**,
  - **"name"** : **"MCJANG"**,
  - **"latestSeenMovie"** : **"추격자"**
- }
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$set 제한자

§ “최근에 본 영화” 수정

- > **db.user.update**({**"id"** : **"mcjang"**}, {**"\$set"** : {**"latestSeenMovie"** : **"미니언즈"**}})
- **WriteResult**({ **"nMatched"** : 1, **"nUpserted"** : 0, **"nModified"** : 1 })
- > **db.user.findOne**({**"id"** : **"mcjang"**})
- {
  - **"\_id"** : ObjectId(**"56259382c8677ca2752803d0"**),
  - **"id"** : **"mcjang"**,
  - **"password"** : **"1234"**,
  - **"name"** : **"MCJANG"**,
  - **"latestSeenMovie"** : **"미니언즈"** // 데이터가 추가되지 않고 수정된다.
- }
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$set 제한자

§ “최근에 본 영화” 수정

- > **db.user.update({"id" : "mcjang"}, {"\$set" : {"latestSeenMovie" : ["베테랑", "미니언즈"]}})**
- WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
- > db.user.findOne({"id" : "mcjang"})
- {
- "\_id" : ObjectId("56259382c8677ca2752803d0"),
- "id" : "mcjang",
- "password" : "1234",
- "name" : "MCJANG",
- **"latestSeenMovie" : [**
- **"베테랑",**
- **"미니언즈"**
- **]**
- }
- >



# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$unset 제한자

§ Field와 값을 모두 삭제함.

§ “최근에 본 영화” 삭제

- > **db.user.update({"id" : "mcjang"}, {"\$unset" : {"latestSeenMovie" : 1}})**
- WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
- > **db.user.findOne({"id" : "mcjang"})**
- {
  - "\_id" : ObjectId("56259382c8677ca2752803d0"),
  - "id" : "mcjang",
  - "password" : "1234",
  - "name" : "MCJANG"
- }
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$push 제한자

§ \$set 제한자와 같은 역할. 단, Field 가 존재할 경우 요소를 배열 끝에 추가한다.

§ “최근에 본 영화” 등록

- > **db.user.update**({ "id" : "mcjang" }, { "\$push" : { "latestSeenMovie" : "베 테 랑" } })
- **WriteResult**({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
- > **db.user.findOne**({ "id" : "mcjang" })
- {
  - "\_id" : ObjectId("56259382c8677ca2752803d0"),
  - "id" : "mcjang",
  - "password" : "1234",
  - "name" : "MCJANG",
  - "latestSeenMovie" : [
    - "베 테 랑"
  - ]
  - }
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$push 제한자

§ \$set 제한자와 같은 역할. 단, Field 가 존재할 경우 요소를 배열 끝에 추가한다.

§ “최근에 본 영화” 등록

- > **db.user.update**({**"id"** : **"mcjang"**}, {**"\$push"** : {**"latestSeenMovie"** : **"치외법권"**}})
- **WriteResult**({ **"nMatched"** : 1, **"nUpserted"** : 0, **"nModified"** : 1 })
- > **db.user.findOne**({**"id"** : **"mcjang"**}) **"}}**
- {
  - **"\_id"** : ObjectId(**"56259382c8677ca2752803d0"**),
  - **"id"** : **"mcjang"**,
  - **"password"** : **"1234"**,
  - **"name"** : **"MCJANG"**,
  - **"latestSeenMovie"** : [
    - **"베테랑"**,
    - **"치외법권"**
  - **]**
- **}**
- **>**

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$pull 제한자

§ 요소를 제거함

§ “최근에 본 영화” 중 베테랑 제거

- > **db.user.update**({**"id"** : **"mcjang"**}, {**"\$pull"** : {**"latestSeenMovie"** : **"베테랑"**}})
- **WriteResult**({ **"nMatched"** : 1, **"nUpserted"** : 0, **"nModified"** : 1 })
- > **db.user.findOne**({**"id"** : **"mcjang"**})
- {
  - **"\_id"** : ObjectId(**"56259382c8677ca2752803d0"**),
  - **"id"** : **"mcjang"**,
  - **"password"** : **"1234"**,
  - **"name"** : **"MCJANG"**,
  - **"latestSeenMovie"** : [
    - **"치외법권"**
  - **]**
- }
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$pull 제한자

§ 포스트에 댓글을 여러 개 등록하고 그 중 하나 삭제하기

§ 시퀀스 증가

- > **db.seq.update({"name" : "post"}, {"\$inc" : {"val" : 1}})**
- **WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })**
- > **var seq = db.seq.findOne({"name" : "post"})**
- > **seq**
- **{ "\_id" : ObjectId("56258f1ec8677ca2752803cf"), "name" : "post", "val" : 1 }**
- > **seq.val**
- **1**
- >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$pull 제한자

§ 포스트에 댓글을 여러 개 등록하고 그 중 하나 삭제하기

§ Post 등록

- > **var post = {**
- ... **"id" : seq.val,**
- ... **"subject" : "MongoDB!",**
- ... **"content" : "Awesome!",**
- ... **"author" : "mcjang"**
- ... **}**
- > **db.post.insert(post);**
- **WriteResult({ "nInserted" : 1 })**
- > **db.post.findOne({"id": 1})**
- {
- **"\_id" : ObjectId("5625b57510aca55475b7f880"),**
- **"id" : 1,**
- **"subject" : "MongoDB!",**
- **"content" : "Awesome!",**
- **"author" : "mcjang"**
- }  
• >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$pull 제한자

§ 포스트에 댓글을 여러 개 등록하고 그 중 하나 삭제하기

§ 댓글 등록

- `> db.post.update({"id" : 1}, {"$push" : { "replies" : {"id" : 1, "author" : "guest", "content" : "^^" } } })`
- `WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })`
- `> db.post.findOne({"id" : 1})`
- `{`
- `"_id" : ObjectId("5625b57510aca55475b7f880"),`
- `"id" : 1,`
- `"subject" : "MongoDB!",`
- `"content" : "Awesome!",`
- `"author" : "mcjang",`
- `"replies" : [`
- `{`
- `"id" : 1,`
- `"author" : "guest",`
- `"content" : "^^"`
- `}`
- `]`
- `}`
- `>`

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$pull 제한자

- § 포스트에 댓글을 여러 개 등록하고 그 중 하나 삭제하기
- § 댓글 등록

```
• > db.post.update({"id" : 1}, {"$push" : { "replies" : {"id" : 2, "author" : "guest", "content" : "Great! MongoDB" } }})
• WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
• > db.post.findOne({"id" : 1})
• {
•   "_id" : ObjectId("5625b57510aca55475b7f880"),
•   "id" : 1,
•   "subject" : "MongoDB!",
•   "content" : "Awesome!",
•   "author" : "mcjang",
•   "replies" : [
•     {
•       "id" : 1,
•       "author" : "guest",
•       "content" : "^^"
•     },
•     {
•       "id" : 2,
•       "author" : "guest",
•       "content" : "Great! MongoDB"
•     }
•   ]
• }
• >
```



# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ \$pull 제한자

§ 포스트에 댓글을 여러 개 등록하고 그 중 하나 삭제하기

§ Id가 2번인 댓글 삭제하기

- **> db.post.update({"id" : 1}, {"\$pull" : {"replies" : {"id" : 2 }}})**
- **WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })**
- **> db.post.findOne({"id" : 1})**
- {
  - "\_id" : ObjectId("5625bbf80bcfc3fab5842c74"),
  - "id" : 1,
  - "subject" : "MongoDB!",
  - "content" : "Awesome!",
  - "author" : "mcjang",
  - "replies" : [
    - {
    - "id" : 1,
    - "author" : "guest",
    - "content" : "^^"
    - }
- ]
- }  
• >

# 01 문서의 생성, 갱신, 삭제 - 문서의 갱신

## ■ Save

§ 갱신입력: `$set`과 동일한 역할이지만, 더 간단하게 사용할 수 있음.

```
• > var post = db.post.findOne({"id" : 1})
• > post.pageview = 1;
• 1
• > db.post.save(post)
• WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
• > db.post.findOne({"id" : 1})
• {
•   "_id" : ObjectId("5625bbf80bcfc3fab5842c74"),
•   "id" : 1,
•   "subject" : "MongoDB!",
•   "content" : "Awesome!",
•   "author" : "mcjang",
•   "replies" : [
•     {
•       "id" : 1,
•       "author" : "guest",
•       "content" : "^^"
•     }
•   ],
•   "pageview" : 1
• }
• >
```

**1-04**

쿼리하기

## 04 쿼리하기

### ■ 데이터 준비

```
§ > function insertMassivePost() {  
§ ... for (var i = 0; i < 20000; i++) {  
§ ... db.seq.update({"name":"post"}, {"$inc" : {"val" : 1}})  
§ ... var seq = db.seq.findOne({"name" : "post"})  
§ ... var post = {  
§ ... "id" : seq.val,  
§ ... "subject" : "Hello " + seq.val,  
§ ... "content" : "MongoDB " + seq.val,  
§ ... "author" : "mcjang",  
§ ... "pageview" : 0  
§ ... }  
§ ...  
§ ... db.post.insert(post)  
§ ... }  
§ ... }  
§ > insertMassivePost()  
§ >
```

## 04 쿼리하기

### ■ 조건 지정하기

§ \$lt, \$lte, \$gt, \$gte 각각 <, <=, >, >=에 해당하는 비교연산자.

요소명	비교연산자	
키워드	형식	설명
\$lt	<	작다
\$lte	<=	작거나 같다
\$gt	>	크다
\$gte	>=	크거나 같다
\$ne	!=	같지 않다

§ 나이가 10보다 작다.

- `db.c.find( { "age" : { "$lt" : 10 } } )`

§ 나이가 40보다 작고 10보다 크다

- `db.c.find( { "age" : { "$lt" : 40, "$gt" : 10 } } )`

## 04 쿼리하기

### ■ 조건 지정하기

§ \$in, \$nin, \$or, \$not 각각 In, Not In, Or, Not 에 해당 하는 논리 연산자

요소명	논리연산자	
키워드	형식	설명
\$in	IN	여러 개 중 하나
\$nin	NOT IN	여러 개 가 하나라도 없는 것
\$or	OR	OR 연산
\$not	! (NOT)	NOT 연산

§ 사는 지역이 광역시인 모든 것.

- `db.c.find( { "city" : { "$in" : [ "서울", "대전", "대구", "부산", "울산", "인천", "광주" ] } } )`

§ 사는 지역이 광역시 이거나, 제주도 인 모든 것

- `db.c.find( { "$or" : [  
    "city" : {  
        "$in" : [ "서울", "대전", "대구", "부산", "울산", "인천", "광주" ]  
    },  
    { "city" : "제주" }  
] } )`

§ 태어난 해가 짝수해 인 모든 것 : birthYear 를 2로 나눈 나머지 값이 1이 아닌 모든 것

- `db.c.find( { "birthYear" : { "$not" : { "$mod" : [ 2, 1 ] } } } )`

## 04 쿼리하기

### ■ 조건 지정하기

§ 배열에 쿼리하기.

```
{
  "_id" : ObjectId("556e67ef35316c8fcfee226e"),
  "fruits" : [
    "banana", "apple", "peach"
  ]
}
```

§ **fruits** 값에 **banana**가 있는 것 찾아내기

- `db.c.find( { "fruits" : "banana" } )`

§ **fruits** 값에 **banana, peach** 가 있는 것 찾아내기

- `db.c.find( { "fruits" : { "$all" : ["banana", "peach"] } } )`

§ **\$size**

§ **fruits** 값이 3인 것 찾아내기

- `db.c.find( { "fruits" : { "$size" : 3 } } )`

## 04 쿼리하기

### ■ 조건 지정하기

§ **\$slice (between)**

§ 배열 요소의 부분 집합을 가져옴.

§ 블로그 게시물의 먼저 달린 댓글 열 개를 받아오기

- **db.c.findOne( {...조건...}, { "comment" : { "\$slice" : 10 } } )**

§ 블로그 게시물의 나중에 달린 댓글 열 개를 받아오기

- **db.c.findOne( {...조건...}, { "comment" : { "\$slice" : -10 } } )**

§ 블로그 게시물의 댓글 중 10 부터 20까지를 가져오기

- **db.c.findOne( {...조건...}, { "comment" : { "\$slice" : [10, 10] } } )**



## 04 쿼리하기

### ■ 조건 지정하기

§ 내장 문서에 쿼리하기

§ Case 1

```
{
  "_id" : ObjectId("556e67ef35316c8fcfee226e"),
  "name" : {
    "first" : "jang",
    "last" : "mc"
  }
}
```

§ name 문서 안의 first 가 jang 이고 last 가 mc 인 문서 가져오기

- `db.c.find ( { "name.first" : "jang", "name.last" : "mc" } )`

## 04 쿼리하기

### ■ 조건 지정하기

§ 내장 문서에 쿼리하기

§ Case 2

```
{
  "_id" : ObjectId("556e67ef35316c8fcfee226e"),
  "comments" : [
    {
      "author" : "mcjang",
      "recommend" : 10,
      "comment" : "hhhh"
    },
    {
      "author" : "mcyou",
      "recommend" : 50,
      "comment" : "k"
    }
  ]
}
```

§ author가 mcjang 이고, recommend 가 50인 댓글 가져오기

- `db.c.find ( { "comments" : { "$elemMatch" : { "author" : "mcjang", "recommend" : 50 } } } )`

§ Case 1 의 쿼리는 쓸 수 없다.

## 04 쿼리하기

### ■ 조건 지정하기

#### § 결과 수 제한하기

- 전체 조회한 것 중 30개만 가져온다.
  - `db.c.find().limit(30);`

#### § 결과 건너뛰기

- 조건과 맞는 처음 3개를 건너뛰고 그 나머지 결과를 반환.
  - `db.c.find().skip(3);`

#### § 정렬하기

- `username`은 오름차순, `age` 는 내림차순 정렬하기
  - `db.c.find().sort({"username" : 1, "age" : -1})`

#### § 결과를 제한하고 정렬하기

- 상품명이 `mp3` 이고 가격기준 내림차순 정렬해 한 페이지당 50개씩 보여주기
  - `db.c.find({"desc" : "mp3"}).limit(50).sort({"price" : -1})`
- 다음페이지
  - `db.c.find({"desc" : "mp3"}).limit(50).skip(50).sort({"price" : -1})`

*1-05*

색인

## 05 색인

### ■ 쿼리 수행 속도의 개선

§ 아래와 같이 쿼리 할 때 쿼리하는 **Key**에 색인을 생성하게 되면, 쿼리의 속도를 개선시킬 수 있다.

- `db.c.find( { "username" : "mcjang" } )`

§ 색인 방법

- `db.c.ensureIndex( { "username" : 1 } ) // 1 : ASC, -1 : DESC`

§ 색인은 컬렉션에 한번만 생성하면 됨.

- 동일한 색인을 생성하려고 시도하면, 아무런 일도 일어나지 않는다.

§ **Key**에 색인을 생성하면 그 **Key**에 대한 쿼리가 빨라진다.

- → 자주 사용되는 쿼리에 대해서는 색인을 해주어야 함.
- `db.c.find( { "username" : "mcjang", "age" : 50 } )`
  - `db.c.ensureIndex( { "username" : 1, "age" : 1 } )`

§ 모든 **Key**에 색인을 추가하게 되면 **Database**에 무리가 갈 수 있으니 필요한 **Key**에만 색인을 추가한다.

- 색인이 너무 많게 되면, **Insert**, **Update**, **Delete**에 모두 부담이 된다.

**1-06**

집계

## 06 집계

---

- RDB의 집계 함수기능을 제공함.
- Count
- Distinct
- Group
- Map Reduce

## 06 집계

### ■ Count

- § Collection 내의 문서의 수를 반환.
- § SQL 의 count와 같은 역할을 수행한다.
- § `db.foo.count()` 혹은 `db.foo.count( { "name" : "mcjang" } )`
- § Count는 문서의 수와 상관없이 매우 빠른 연산을 수행하지만
- § 조건이 추가될 경우 속도가 느려진다.

### ■ Distinct

- § Collection 내의 주어진 Key의 고유한 값을 반환함.
- § People Collection 에 다음과 같은 문서가 있다고 가정
  - `{ "name" : "Ada", "age" : 20 }`
  - `{ "name" : "Fred", "age" : 35 }`
  - `{ "name" : "Susan", "age" : 60 }`
  - `{ "name" : "Andy", "age" : 35 }`
- § age key 에 distinct를 호출
  - `db.runCommand( { "distinct" : "people", "key" : "age" } )`
  - 결과 → `{ "values" : [20, 35, 60], "ok" : 1 }`



## 06 집계

### ■ Group

- § SQL의 **Group By**와 동일함.
- § **Distinct**와 동일하게 **runCommand** 로 실행함.
- § **Group**의 포맷

```
{
  group:
  {
    ns: <namespace>,
    key: <key>,
    $reduce: <reduce function>,
    $keyf: <key function>,
    cond: <query>,
    finalize: <finalize function>
  }
}
```

### ■ Group

- § “ns” → 어떤 컬렉션에서 **Group**을 수행할 것인지 결정
- § “key” → 지정한 컬렉션에서 문서를 그룹핑할 키를 지정. **Group by column** 과 동일함.
- § “\$reduce” → 컬렉션 내의 각 문서들에 대해서 한번씩 호출한다. **Function**의 인자는 총 2개인데, 하나는 해당 문서를 받고, 또 다른 하나는 현재까지 누적 계산된 문서를 받는다. 누적계산은 사용자가 정의할 수 있다.
- § “\$keyf ” → 함수를 키로 사용한다. 하나의 키가 아닌 복잡한 조건의 **Key**를 사용하려 할 때 쓴다. “필수 아님”
- § “cond” → 그룹핑할 표본을 정의한다.
- § “finalize” → SQL의 **Having**과 같음. 그룹핑 될 결과로 다시 한번 연산할 때 사용한다.
- § “initial” → 누적 계산할 항목을 정의한다.

## 06 집계

### ■ Group

§ 데이터 준비

```
var post = {};  
function addPost() {  
    for(var i = 0; i < 100; i++) {  
        for(var j = 0; j < 100; j++) {  
            post.number = i;  
            post.postNumber = j;  
            post.date = new Date();  
            post.comment = "Hi hello" + i + j;  
  
            db.post.insert(post);  
        }  
    }  
}  
  
addPost();
```

## 06 집계

### ■ Group

§ 등록된 글 중 "number" 별 등록 개수를 구하는 Query.

```
db.runCommand ( { "group" : {  
  "ns" : "post",  
  "key" : { "number" : 1 },  
  "initial" : { "count" : 0 },  
  "$reduce" : function (curr, result) {  
    if (curr.number == result.number) {  
      result.count += 1;  
    }  
  }  
}} )
```

## 06 집계

### ■ Group

§ 등록된 글 중 90이상의 "number" 별 등록 개수를 구하는 Query.

```
db.runCommand ( { "group" : {  
  "ns" : "post",  
  "key" : { "number" : 1 },  
  "initial" : { "count" : 0 },  
  "$reduce" : function (curr, result) {  
    if (curr.number == result.number) {  
      result.count += 1;  
    }  
  },  
  "condition" : { "number" : { "$gte" : 90 } }  
}})
```

## 06 집계

### ■ Group

§ 자료 준비 (1 / 2)

```
var post = {};  
function addPost() {  
    for(var i = 0; i < 50; i++) {  
        post.number = 0;  
        post.tags = ["NoSQL", "MySQL", "Oracle"];  
        db.test1.insert(post);  
    }  
    for(var i = 0; i < 15; i++) {  
        post.number = 0;  
        post.tags = ["NoSQL"];  
        db.test1.insert(post);  
    }  
    for(var i = 0; i < 5; i++) {  
        post.number = 0;  
        post.tags = ["Oracle"];  
        db.test1.insert(post);  
    }  
}
```

### ■ Group

§ 자료 준비 (2 / 2)

```
for(var i = 0; i < 10; i++) {  
    post.number = 1;  
    post.tags = ["NoSQL", "Winter"];  
    db.test1.insert(post);  
}  
for(var i = 0; i < 5; i++) {  
    post.number = 1;  
    post.tags = ["NoSQL"];  
    db.test1.insert(post);  
}  
for(var i = 0; i < 15; i++) {  
    post.number = 2;  
    post.tags = ["Spring"];  
    db.test1.insert(post);  
}  
}
```

### ■ Group

§ 등록된 글들의 tag별 개수 구하기

```
db.runCommand ( { "group" : {  
  "ns" : "test1",  
  "key" : { "number" : 1 },  
  "initial" : { "tags" : {} },  
  "$reduce" : function (curr, result) {  
    for(i in curr.tags) {  
      if(curr.tags[i] in result.tags) {  
        result.tags[curr.tags[i]]++;  
      }  
      else {  
        result.tags[curr.tags[i]] = 1;  
      }  
    }  
  }  
}}
```



### ■ Group

§ 등록된 글들의 **tag**별 개수 구하고 그 중 가장 큰 수의 **Tag** 만 가져오기 (1/2)

```
db.runCommand ( { "group" : {
  "ns" : "test1",
  "key" : { "number" : 1 },
  "initial" : { "tags" : {} },
  "$reduce" : function (curr, result) {
    for(i in curr.tags) {
      if(curr.tags[i] in result.tags) {
        result.tags[curr.tags[i]]++;
      }
      else {
        result.tags[curr.tags[i]] = 1;
      }
    }
  },
}
```

## 06 집계

### ■ Group

§ 등록된 글들의 **tag**별 개수 구하고 그 중 가장 큰 수의 **Tag** 만 가져오기 (2/2)

```
"finalize" : function(result) {  
    var mostPopular = 0;  
    for(i in result.tags) {  
        if(result.tags[i] > mostPopular) {  
            result.tag = i;  
            mostPopular = result.tags[i];  
        }  
    }  
    delete result.tags  
}  
})  
})
```

## 06 집계

### ■ 맵리듀스

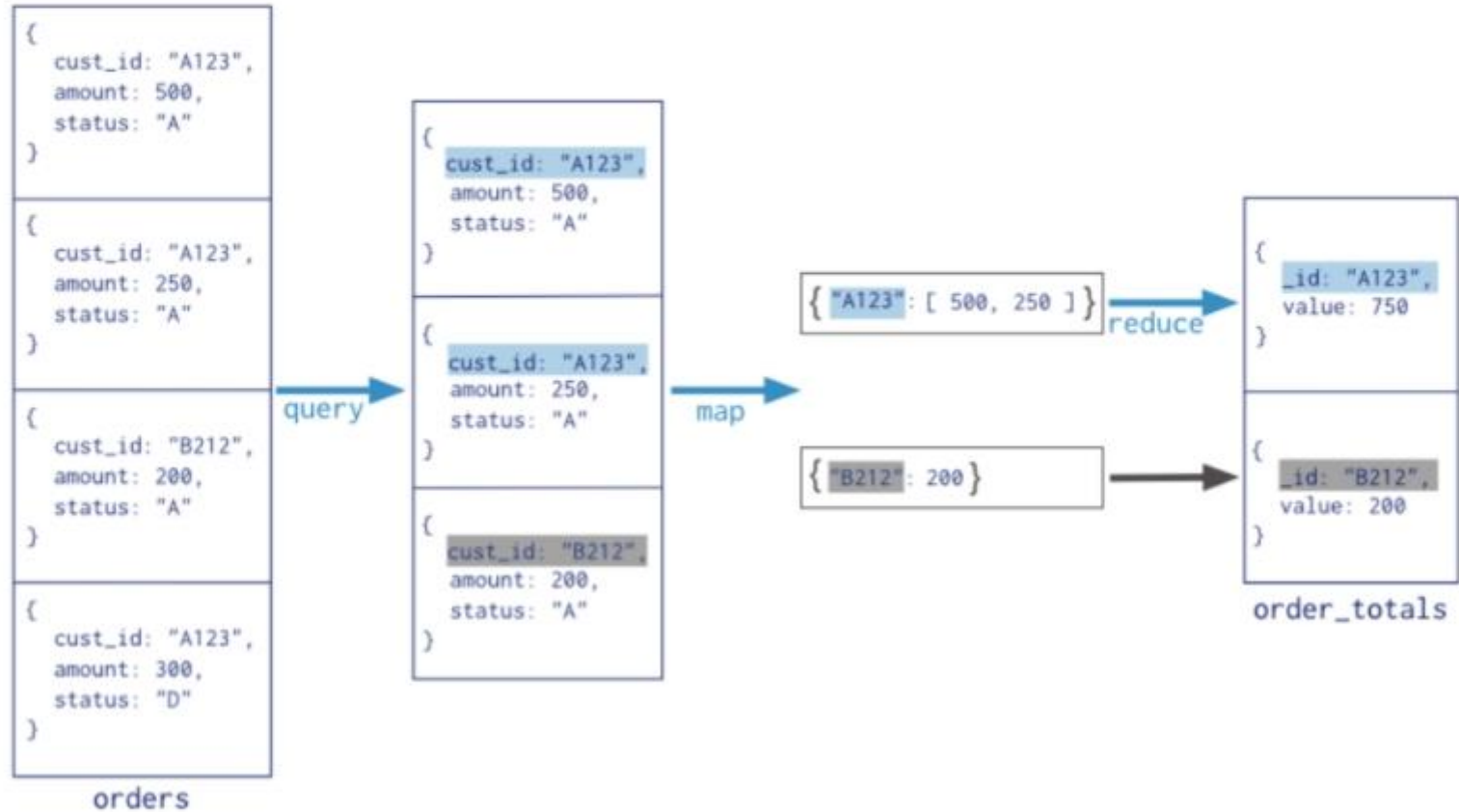
§ 대용량 데이터베이스를 응축(직속/집계)하기 위한 데이터 프로세싱 패러다임.

```
Collection
  ↓
db.orders.mapReduce(
  map    → function() { emit( this.cust_id, this.amount ); },
  reduce → function(key, values) { return Array.sum( values ) },
  {
    query: { status: "A" },
    out: "order_totals"
  }
)
```

## 06 집계

### ■ 맵리듀스

§ **mapReduce** 가 동작하는 방식



## 06 집계

### ■ 맵리듀스

#### § 자료 준비

```
> db.test1.find({}, {_id : 0});
{ "cust_id" : "A123", "amount" : 500, "status" : "A" }
{ "cust_id" : "A123", "amount" : 250, "status" : "A" }
{ "cust_id" : "B212", "amount" : 200, "status" : "A" }
{ "cust_id" : "A123", "amount" : 300, "status" : "D" }
>
```

```
> db.test1.mapReduce(
... function() {emit(this.cust_id, this.amount)},
... function(key, values) {return Array.sum(values)},
... {
...   query : {status : "A"},
...   out: "order_totals"
... }
... )
{
  "result" : "order_totals",
  "timeMillis" : 24,
  "counts" : {
    "input" : 3,
    "emit" : 3,
    "reduce" : 1,
    "output" : 2
  },
  "ok" : 1
}
```

```
> db.order_totals.find()
{ "_id" : "A123", "value" : 750 }
{ "_id" : "B212", "value" : 200 }
>
```

**1-07**

관리

## ■ Collection 삭제

### § Drop

- `db.collection.drop()` 혹은 `db.runCommand( { "drop" : "collectionName" } )`

## ■ MongoDB Version 정보 및 OS 정보

### § buildInfo

- `db.runCommand( { "buildInfo" : 1 } )`

## ■ Collection 의 정보

### § collStats

- `db.runCommand( { "collStats" : "collectionName" } )`

## ■ 데이터베이스 삭제

### § dropDatabase (현재 데이터베이스의 모든 데이터를 삭제함)

- `db.runCommand( { "dropDatabase" : 1 } )`

## ■ 색인 삭제

### § dropIndexes

- `db.runCommand( { "dropIndexes" : "collectionName", "index" : "indexName" } )`
- `db.runCommand( { "dropIndexes" : "collectionName", "index" : "*" } )` // ← 모두 삭제

## 07 관리

### ■ Collection 이름 변경

- § `renameCollection` (컬렉션 이름을 a 에서 b로 변경한다)
  - `db.runCommand( { "renameCollection" : "a", "to" : "b" } )`

### ■ 사용자 추가하기

- § 단순히 **mongod** 를 실행하면 인증 과정 없이 바로 사용가능하다.
  - § 인증 및 인가 측면에서 굉장히 위험한 방법. 따라서 인증 및 인가에 대한 **Rule** 이 필요하다.
  - § 가장 간단한 방법은 "사용자"를 추가하고, 사용자 이외엔 접근이 불가능하도록 하는 것,
- § `db.createUser()` 명령어로 사용자 추가

```
{ user: "<name>",  
  pwd: "<cleartext password>",  
  customData: { <any information> },  
  roles: [  
    { role: "<role>", db: "<database>" } | "<role>",  
    ...  
  ]  
}
```



## 07 관리

### ■ 사용자 추가하기

```
use admin
db.createUser(
{
  user : "userName",
  pwd : "password",
  roles :
  [
    {
      role : "readWrite", db : "databaseName"
    }
  ]
})
```

- § **Roles** 에 접근해야 할 **DB**명과 접근권한을 추가해서 설정한다.
- § 현재 실행중인 **mongod** 를 중지하고 "**mongod.exe -auth**" 로 재시작한다.

## 07 관리

---

- 사용자 접속하기

```
use admin  
db.auth("userName", "password");
```

§ **Admin**을 통해서 인증을 해 로그인

§ 이후 인가된 **DB**로 **use DBName** 해서 이동한다.

**THANK YOU**