

The background of the image shows a rural landscape with rolling hills covered in agricultural fields. Several white wind turbines are scattered across the hills, some on the left side and more concentrated on the right side. The sky is a clear, vibrant blue.

MongoDB^(ver.3.2.0)

목차 (Table of Contents)

1. MongoDB 소개
2. MongoDB 시작하기
3. 인덱스 (색인)
4. 조합
5. Java API
6. MongoDB 활용



1. MongoDB 소개

-
- 1.1 MongoDB 개요
 - 1.2 MongoDB 주요 특징
 - 1.3 MongoDB 기본 개념

1.1 MongoDB 개요 (1/2)

✓ MongoDB

- 고성능, 고가용성, 자동 스케일링을 제공하는 문서 데이터베이스입니다.
- C++로 개발하였습니다.
- 2007년 10gen에서 개발하여 2009년 오픈 소스로 변경(AGPL 라이선스)
- 2015년 3월, 3.0 버전 출시
- 2017년 9월 현재 3.4.7 버전 출시

✓ MongoDB를 사용하는 곳

- Metlife
- Salesforce
- CISCO
- McAfee
- foursquare
- facebook
- ...



1.1 MongoDB 개요 (2/2) – 문서형 DB

✓ MongoDB에서 레코드는 문서(document)입니다.

- 문서는 필드와 값의 쌍으로 구성된 데이터 구조를 가지고 있습니다.

✓ 문서 구조

- MongoDB의 문서 형식은 BSON(JSON을 바이너리화 한 것)입니다.
- 필드의 값은 다른 문서나 배열, 문서의 배열 등을 포함할 수 있습니다

✓ 문서의 장점

- 문서는 많은 프로그래밍 언어들이 가진 데이터 타입과 일치합니다. (예를 들어 객체)
- 내장된 문서나 배열 등은 JOIN에 따른 부담을 줄여줍니다.
- 동적 스키마를 통해 뛰어나게 다형성을 제공합니다.

```
{  
    name : "sue",  
    age : 26,  
    status : "A",  
    groups : ["news", "sports"]  
}
```

1.2 MongoDB 주요 특징

✓ 다양한 데이터 모델

- 행(row)이라는 개념보다 유연한 모델인 문서(document)를 사용합니다.
- 내장 문서와 배열을 문서에 사용할 수 있어 복잡한 계층 관계를 하나의 레코드로 표현할 수 있습니다.
- 문서는 스키마가 없으므로 계속 바뀌는 데이터 모델을 유연하게 처리할 수 있습니다.

✓ 손쉬운 확장

- 데이터베이스의 확장을 위해 분산 확장(scale-out)을 제공합니다.

✓ 다양한 기능

- 다양한 색인(Indexing) 기능
- 저장 자바스크립트(Stored JavaScript)
- 조합(Aggregation) 기능
- 고정 크기 컬렉션 (capped collection)
- 파일 저장소

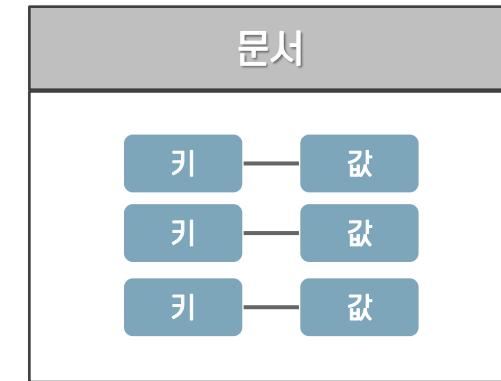
✓ 고성능

✓ 간편한 관리

- 가능한 많은 설정을 알아서 처리하고 사용자는 미세한 수정을 할 수 있습니다.

1.3 MongoDB 기본 개념 (1/4) – 문서 (Document)

- ✓ 문서는 정렬된 키와 연결된 값의 집합입니다.
- ✓ 값은 여러 데이터 타입 중 하나가 될 수 있습니다.
- ✓ 문서의 키는 문자열이며 중복될 수 없습니다.
- ✓ 데이터 타입과 대소문자를 구분합니다.



✓ 문서의 예

- `{'name' : 'namoosori', 'course' : 'MongoDB'}` vs `{'course' : 'MongoDB', 'name' : 'namoosori'}`
→ 키의 정렬 순서가 다르므로 서로 다른 문서이다.
- `{'since' : 2014}` vs `{'since' : '2014'}`
→ 값의 데이터 타입이 다르므로 서로 다른 문서이다.
- `{'Since' : 2014}` vs `{'since' : 2014}`
→ 키의 대소문자가 다르므로 서로 다른 문서이다.

✓ 잘못된 문서의 예

- `{'course' : 'MongoDB', 'course' : 'MongoDB Basic'}`
→ 키가 중복되었으므로 올바른 문서가 아니다.

1.3 MongoDB 기본 개념 (2/4) – 컬렉션 (Collection)

✓ 컬렉션은 문서의 모음입니다.

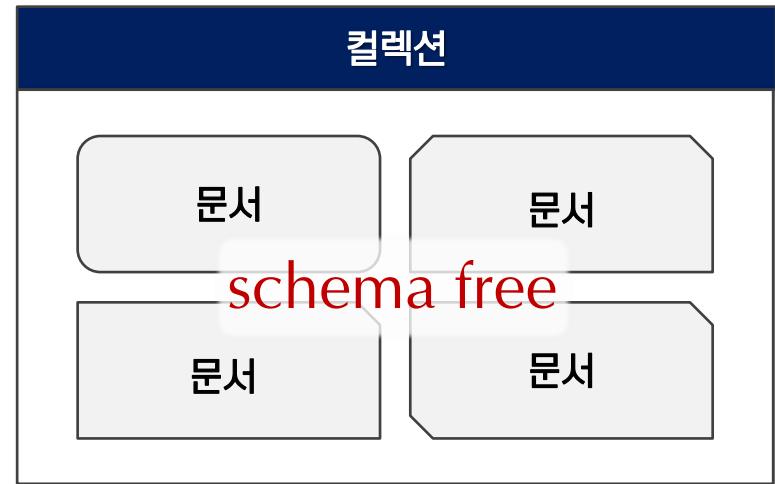
- 문서는 관계형 DB의 '행(row)'을 의미하고, 컬렉션은 '테이블'에 해당합니다.

✓ 컬렉션은 스키마가 없습니다.

- 즉, 하나의 컬렉션 내 문서들은 모두 다른 구조를 가질 수 있습니다.
- 컬렉션이 스키마를 강제하지 않으므로 높은 유연성을 제공합니다.

✓ 네이밍

- 컬렉션은 이름으로 식별합니다. (몇 가지 예외를 제외하고 어떠한 UTF-8 문자열도 사용 가능)
- 컬렉션을 체계화하기 위하여 서브-컬렉션을 사용합니다. (예, blog.posts와 blog.authors)



1.3 MongoDB 기본 개념 (3/4) – 데이터베이스 (Database)

- ✓ MongoDB는 여러 문서를 컬렉션으로 모으고, 여러 컬렉션을 데이터베이스에 담습니다.
- ✓ 각 데이터베이스는 자체 권한을 가지며, 디스크에 분리된 파일로 저장합니다.
- ✓ 데이터베이스는 이름으로 식별합니다. (데이터베이스의 이름은 파일시스템 상의 파일명으로 사용된다.)
- ✓ 예약된 데이터베이스 이름
 - admin : 인증의 관점에서 'root' 데이터베이스, 데이터베이스 목록 조회, 서버 전역에 걸쳐 실행하는 명령어 실행 가능
 - local : 복제되지 않는 데이터베이스로 특정 서버에만 저장하는 컬렉션에 사용함
 - config : 샤딩 설정 시 샤프 정보를 저장하는데 사용함



1.3 MongoDB 기본 개념 (4/4) – SQL과 MongoDB 용어 비교

- ✓ 관계형 DB의 SQL에서 사용하는 용어와 MongoDB의 용어의 차이를 비교하면 다음과 같습니다.

SQL	MongoDB
데이터베이스	데이터베이스
테이블 (Table)	컬렉션 (Collection)
행 (Row)	문서 (BSON)
열 (Column)	필드
테이블 조인 (Join)	내장 문서와 링킹(Linking)
색인(Index)	색인
주-키(Primary Key)	주-키는 _id 필드로 자동 설정됨
집합(예: group by)	집합 프레임워크



2. MongoDB 시작하기

-
- 2.1 다운로드 및 설치
 - 2.2 서버 구동 (mongod)
 - 2.3 MongoDB 웰
 - 2.4 기본적인 CRUD
 - 2.5 _id와 ObjectId
 - 2.6 Operators

2.1 다운로드 및 설치 (1/2)

- ✓ MongoDB 다운로드 페이지에서 자신의 운영체제에 맞는 설치파일을 다운로드 합니다. (예, 64-bit zip)
- ✓ <http://www.mongodb.org/downloads>

The screenshot shows the MongoDB download page with the 'Community Server' tab selected. The main content area displays the 'Current Stable Release (3.2.8)' information, including the release date (07/12/2016), release notes, and changelog. Below this, there are four download links for Windows, Linux, OSX, and Solaris. A dropdown menu labeled 'Version:' is open, showing 'Windows Server 2008 R2 64-bit and later, with SSL support'. Another dropdown menu labeled 'Installation Package:' is also visible, with a green button labeled 'DOWNLOAD (msi)'.

2.1 다운로드 및 설치 [2/2]

✓ 설치파일(zip)의 압축을 해제합니다.

- (예: c:\mongodb)

✓ 데이터가 저장되는 디렉토리를 생성합니다.

- MongoDB에서 사용하는 디폴드 저장 디렉토리는 c:\data\db 입니다.
- 커맨드 창(cmd)에서 아래와 같이 디렉토리를 생성합니다.
- mkdir c:\data\db

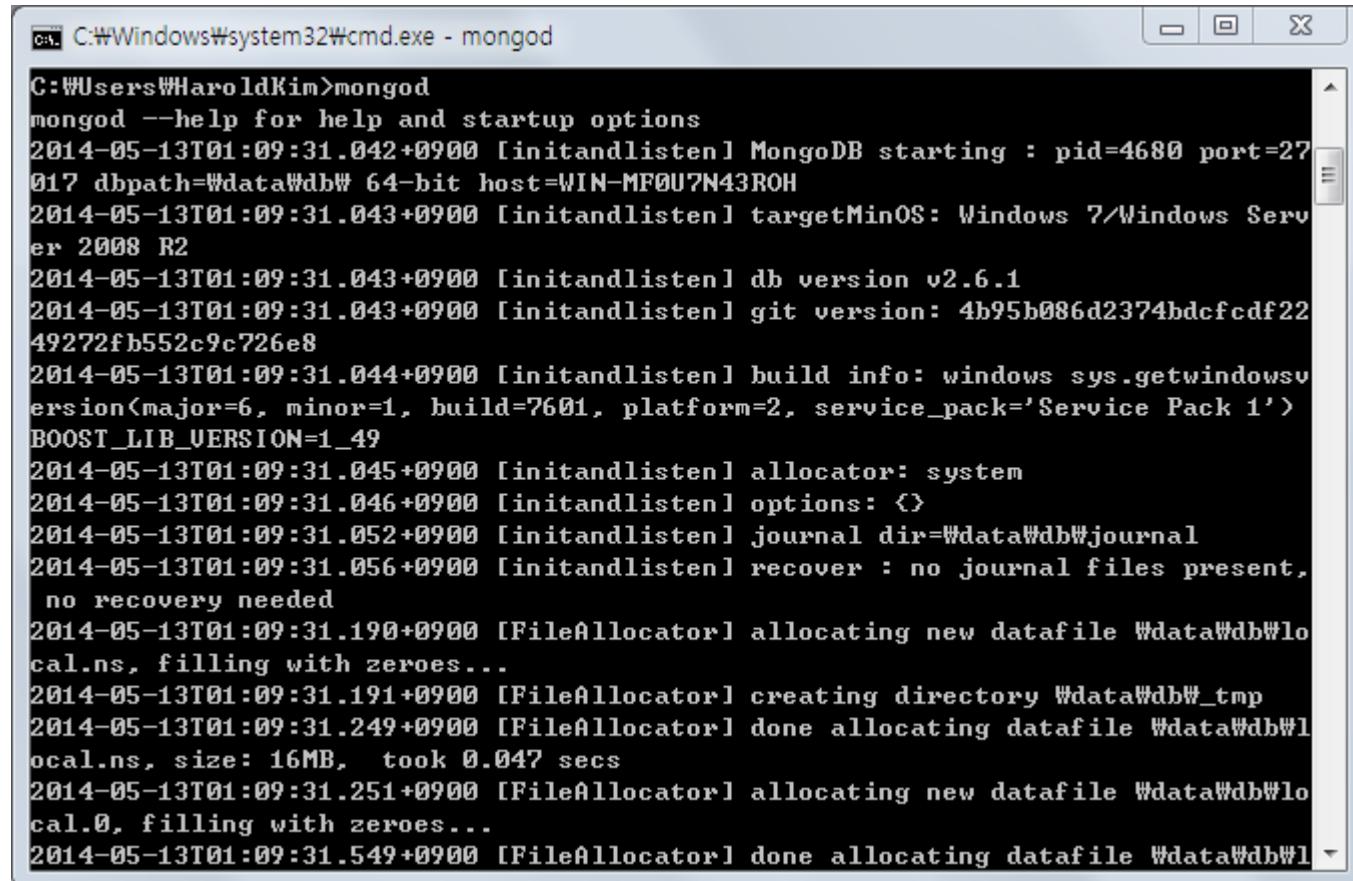
✓ 환경변수 PATH에 mongodb의 bin 경로를 추가합니다.

- MongoDB 관련 명령어를 편리하게 실행할 수 있도록 환경변수를 추가합니다.
- Path : c:\mongodb\bin;...

2.2 서버 구동 (mongod)

✓ 커맨드 창(cmd)에서 mongod 명령을 실행하여 MongoDB 서버를 구동합니다.

- mongod



The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe - mongod'. The command entered is 'C:\Users\HaroldKim>mongod'. The output log is as follows:

```
C:\Users\HaroldKim>mongod
mongod --help for help and startup options
2014-05-13T01:09:31.042+0900 [initandlisten] MongoDB starting : pid=4680 port=27017 dbpath=\data\db\ 64-bit host=WIN-MFOU7N43ROH
2014-05-13T01:09:31.043+0900 [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2014-05-13T01:09:31.043+0900 [initandlisten] db version v2.6.1
2014-05-13T01:09:31.043+0900 [initandlisten] git version: 4b95b086d2374bdcfcdf2249272fb552c9c726e8
2014-05-13T01:09:31.044+0900 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1')
BOOST_LIB_VERSION=1_49
2014-05-13T01:09:31.045+0900 [initandlisten] allocator: system
2014-05-13T01:09:31.046+0900 [initandlisten] options: <>
2014-05-13T01:09:31.052+0900 [initandlisten] journal dir=\data\db\journal
2014-05-13T01:09:31.056+0900 [initandlisten] recover : no journal files present,
no recovery needed
2014-05-13T01:09:31.190+0900 [FileAllocator] allocating new datafile \data\db\local.ns, filling with zeroes...
2014-05-13T01:09:31.191+0900 [FileAllocator] creating directory \data\db\_tmp
2014-05-13T01:09:31.249+0900 [FileAllocator] done allocating datafile \data\db\local.ns, size: 16MB, took 0.047 secs
2014-05-13T01:09:31.251+0900 [FileAllocator] allocating new datafile \data\db\local.0, filling with zeroes...
2014-05-13T01:09:31.549+0900 [FileAllocator] done allocating datafile \data\db\local.0
```

2.3 MongoDB 쉘 [1/2]

✓ MongoDB는 커맨드라인을 통해 MongoDB 인스턴스와 상호작용하는 자바스크립트 쉘을 제공합니다.

- MongoDB 쉘은 관리 기능을 수행, 실행 중인 인스턴스를 확인하거나 간단한 기능을 테스트할 때 유용합니다.
- 커맨드 창에서 mongo 명령어를 실행합니다.

```
C:\...> mongo  
MongoDB shell version: 3.0.0  
connecting to: test  
>
```

- 쉘에서 표준 자바스크립트의 모든 기능을 활용할 수 있습니다.

```
> function sayHello(name) {  
... return "Hello " + name;  
... }  
>  
> sayHello("mongoDB");  
Hello mongoDB  
>
```

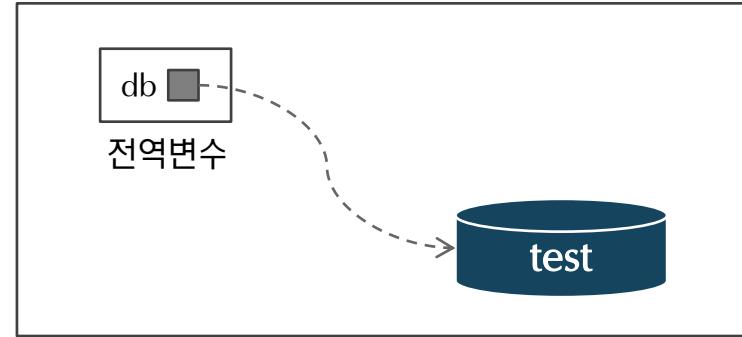
2.3 MongoDB 쉘 (2/2)

✓ MongoDB 쉘은 독자적으로 사용할 수 있는 MongoDB 클라이언트입니다.

✓ MongoDB에 접근하는 변수 (db)

- MongoDB 쉘이 시작할 때, MongoDB 서버의 test 데이터베이스에 연결하고 이 데이터베이스 연결을 전역변수 db에 할당합니다

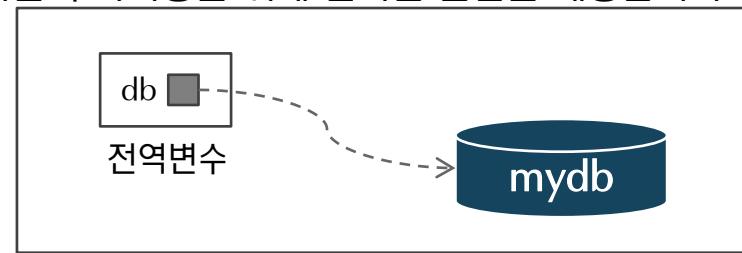
```
C:\...> mongo  
MongoDB shell version: 2.6.1  
connecting to: test  
> db  
test  
>
```



✓ 데이터베이스 선택 (use)

- 쉘은 자바스크립트 구문으로는 유효하지 않지만, SQL 사용자에게 친숙한 추가기능을 위해 편리한 문법을 제공합니다.

```
> use mydb  
switched to db mydb  
>
```



✓ 컬렉션에 접근하기

- db.컬렉션명
- db.getCollection("컬렉션명")
 - 컬렉션명이 db의 프로퍼티명과 일치하는 경우, 'db.컬렉션명'으로는 접근할 수 없으므로 getCollection() 메소드를 사용합니다.

2.4 기본적인 CRUD (1/5)

✓ 문서 추가 (insert)

- insert 기능은 컬렉션에 문서를 추가합니다.

✓ 예시

- 새로운 문서를 나타내는 자바스크립트 객체를 생성합니다.

```
> sportsCommunity = { "name" : "스포츠 커뮤니티",
... "description" : "운동을 통해 몸과 마음을 단련하세요",
... "openDate" : new Date() }

{
    "name" : "스포츠 커뮤니티",
    "description" : "운동을 통해 몸과 마음을 단련하세요",
    "openDate" : ISODate("2014-05-13T14:36:04.927Z")
}
```

- insert 메소드로 새로운 문서를 컬렉션에 추가합니다. find 메소드로 컬렉션에 저장된 문서를 확인합니다.

```
> db.community.insert(sportsCommunity)
WriteResult({ "nInserted" : 1 })
> db.community.find()
{ "_id" : ObjectId("53722e663fac77b1ada967fd"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과
마음을 단련하세요", "openDate" : ISODate("2014-05-13T14:36:04.927Z" })
>
```

2.4 기본적인 CRUD (2/5)

✓ 조회하기 (find)

- find 메소드는 컬렉션 내 모든 문서를 반환합니다. findOne 메소드는 하나의 문서만 조회합니다.

```
> db.community.findOne()
{
    "_id" : ObjectId("53722e663fac77b1ada967fd"),
    "name" : "스포츠 커뮤니티",
    "description" : "운동을 통해 몸과 마음을 단련하세요",
    "openDate" : ISODate("2014-05-13T14:36:04.927Z")
}
```

✓ 쿼리 조건으로 조회

- find 및 findOne 메소드는 문서의 형태로 쿼리 조건을 전달받아 조건에 일치하는 문서 목록을 조회합니다.
- 단, 웹 화면에는 최대 20개의 문서만 보여줍니다.

```
> db.community.find({"name":"Developer 커뮤니티"})
{ "_id" : ObjectId("537232793fac77b1ada967fe"), "name" : "Developer 커뮤니티", "description" : "개발자들의 정보를 공유합니다." }
```

2.4 기본적인 CRUD (3/5)

✓ 수정하기 (update)

- update 메소드는 기존의 문서를 수정합니다.
- update 메소드의 첫 번째 인자는 수정하려는 문서를 찾을 조건이고, 두 번째는 치환할 새로운 문서입니다.

✓ 예시

- 우선, 기존 문서를 조회하여 내용을 수정하고 update 메소드를 호출하여 새로운 내용으로 수정합니다.

```
> query = {"name" : "Developer 커뮤니티"}  
{ "name" : "Developer 커뮤니티" }  
> devCommunity = db.community.findOne(query)  
{  
    "_id" : ObjectId("537232793fac77b1ada967fe"),  
    "name" : "Developer 커뮤니티",  
    "description" : "개발자들의 정보를 공유합니다."  
}  
> devCommunity.members = []  
[]  
> devCommunity.members.push({"email" :  
    "hyunohkim@nextree.co.kr", "name": "김현민"})  
1
```

```
> db.community.update(query, devCommunity)  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.community.findOne(query)  
{  
    "_id" : ObjectId("537232793fac77b1ada967fe"),  
    "name" : "Developer 커뮤니티",  
    "description" : "개발자들의 정보를 공유합니다.",  
    "members" : [  
        {  
            "email" : "hyunohkim@nextree.co.kr",  
            "name" : "김현민"  
        }  
    ]  
}
```

2.4 기본적인 CRUD (4/5)

✓ 문서의 특정 부분만 갱신하기 위해 제한자를 사용할 수 있습니다.

- \$set : 해당 키의 값만 갱신합니다.
- \$inc : 해당 키의 값을 증가 또는 감소시킵니다.
- \$push : 지정된 키가 존재하면 배열의 끝에 추가하고, 없으면 새로운 배열을 만들어 추가합니다.
- \$addToSet : 배열 내에 같은 값이 존재하지 않는 경우에만 추가합니다.
- \$pop : 배열의 요소를 제거합니다. {key : 1} 인 경우 배열의 끝에서, {key : -1} 인 경우 배열의 처음 요소를 제거합니다.
- \$pull : 주어진 조건에 일치하는 배열의 요소를 제거합니다.

✓ 갱신입력

- 갱신 조건에 일치하는 문서가 발견되지 않으면, 쿼리 문서와 갱신 문서를 합친 새로운 문서를 입력합니다.(세번째 인자를 true로 세팅)
- db.summary.update({'count' : 25}, {'\$inc' : { 'count' : 3}}, **true**)
→ 해당 조건을 만족하는 문서가 없는 경우 count : 28인 문서가 추가됩니다.

✓ 다중문서 갱신

- 네번째 인자를 true로 세팅하면 갱신 조건에 일치하는 모든 문서를 갱신합니다.
- db.employee.update({'deparment':'DEV', {'\$inc' : {'salary' : 10000}}}, false, **true**)
→ 개발 부서(DEV)의 모든 직원들의 연봉을 10000원 인상한다.

2.4 기본적인 CRUD (4/5)

✓ 지우기 (remove)

- remove 메소드는 데이터베이스에서 문서를 완전히 삭제합니다.
- 삭제할 조건을 매개변수에 지정할 수 있고, 매개변수 없이 사용할 경우 컬렉션 내 모든 문서를 삭제합니다.
 - db.[컬렉션명].remove({})

```
> query = {"name" : "Developer 커뮤니티"}  
{ "name" : "Developer 커뮤니티" }  
> db.community.find().count()  
2  
> db.community.remove(query)  
WriteResult({ "nRemoved" : 1 })  
> db.community.find().count()  
1  
> db.community.find(query)  
>
```

2.5 _id와 ObjectId

- ✓ 하나의 컬렉션에서 모든 문서는 각 문서의 고유한 식별자인 '_id' 키를 가집니다.
- ✓ '_id' 키의 값은 어떤 데이터 타입이라도 될 수 있으나 ObjectId 타입이 기본입니다.
- ✓ ObjectId
 - 12바이트 값으로 24자리 16진수 문자열로 표현할 수 있습니다.

0	1	2	3	4	5	6	7	8	9	10	11
Timestamp			Machine			PID		Increment			

- Timestamp : 1970년 1월 1일 이후 초단위 타임스탬프
- Machine : 컴퓨터의 고유 식별자 (보통 서버 호스트명의 해쉬 값)
- PID : 생성 프로세스의 식별자(PID)
- Increment : 1초내 단일 프로세스의 고유성을 책임지는 증가 카운터

- ✓ 자동으로 생성된_id 값 예시

```
{  
  "_id" : ObjectId("537232793fac77b1ada967fe"),  
  "name" : "Developer 커뮤니티",  
  "description" : "개발자들의 정보를 공유합니다.",  
}
```

2.6 Operators [1/7] – 비교 연산자

✓ 문서를 입력한 값과 비교하여 조회하기 위한 연산자입니다.

✓ 연산자 종류

- \$gt - 기준값 보다 크다
- \$gte - 기준값 보다 크거나 같다
- \$lt - 기준값 보다 작다
- \$lte - 기준값 보다 작거나 같다
- \$ne - 같지 않다
- \$nin - 존재하지 않는다

```
> db.user.insert({"name":"한성민", "age":40})
WriteResult({ "nInserted" : 1 })
> db.user.insert({"name":"송태훈", "age":49})
WriteResult({ "nInserted" : 1 })
> db.user.insert({"name":"김현민", "age":35})
WriteResult({ "nInserted" : 1 })
> db.user.find({"age":{$gt:40}})
{ "_id" : ObjectId("5445122986371da11f88be49"), "name" : "송태훈", "age" : 49 }
>
```

2.6 Operators [2/7] – 판단 연산자

✓ 여러 가지 조건을 연결하거나 존재 여부를 판단하는 연산자입니다.

✓ 연산자 종류

- \$and : 여러 조건을 모두 만족
- \$or : 여러 조건 중 하나 이상 만족
- \$nor : Not OR
- \$not : 조건을 만족 시키지 않는 것

```
> db.user.find({$and : [{"age": {$gt:35}}, {"age": {$lt:45}}]})  
{ "_id" : ObjectId("5445122986371da11f88be48"), "name" : "한성민", "age" : 40 }  
  
> db.user.find({$or : [{"age":35}, {"age":49}]})  
{ "_id" : ObjectId("5445122986371da11f88be49"), "name" : "송태훈", "age" : 49 }  
{ "_id" : ObjectId("5445122b86371da11f88be4a"), "name" : "김현민", "age" : 35 }  
>
```

2.6 Operators [3/7] – Update 연산자(\$set)

✓ 특정 field의 값을 추가하거나 교체하는 연산자입니다.

✓ 연산자의 동작

- 해당 field가 존재하지 않으면 field를 추가하여 값을 저장함.
- 해당 field가 존재하면 field의 값이 변경됨.
- '.'(Dot)로 연결된 field는 해당 path로 찾아감.

```
> comm_admin = {"name":"홍길동", "email":"hong@nextree.co.kr"}  
  
> db.community.update(query, {$set : {"admin" : comm_admin}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
>  
> db.community.find(query)  
{ "_id" : ObjectId("5441ed6f86371da11f88be47"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과  
마음을 단련하세요", "openDate" : ISODate("2014-10-18T04:30:36.156Z"), "members" : [ { "email" :  
"syhan@nextree.co.kr", "name" : "한성진" } ], "admin" : { "name" : "홍길동", "email" : "hong@nextree.co.kr" } }  
>
```

2.6 Operators [4/7] – Update 연산자(\$unset)

- ✓ 특정 field를 제거합니다.
- ✓ 연산자의 동작
 - 해당 field가 존재하지 않으면 아무일도 일어나지 않음.
 - 해당 field가 존재하면 field가 제거됨.

```
> db.community.update(query,{$unset : {"admin" : ""}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
  
> db.community.find(query)  
{ "_id" : ObjectId("5441ed6f86371da11f88be47"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과  
마음을 단련하세요", "openDate" : ISODate("2014-10-18T04:30:36.156Z"), "members" : [ { "email" :  
"syhan@nextree.co.kr", "name" : "한성진" } ] }  
>
```

2.6 Operators (5/7) – Update 연산자(\$push)

✓ 배열 field에 값을 추가합니다.

✓ 연산자의 동작

- 해당 배열 field가 존재하지 않으면 배열 field를 추가하고 값이 추가됨.
- 해당 field가 배열이 아니면 연산자가 실패한다.

```
> comm_member1 = {"name": "송태훈", "email": "song@nextree.co.kr"}  
{ "name": "송태훈", "email": "song@nextree.co.kr" }  
> comm_member2 = {"name": "김현민", "email": "kim@nextree.co.kr"}  
{ "name": "김현민", "email": "kim@nextree.co.kr" }  
>  
> db.community.update(query, {$push : {"members" : comm_member1}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.community.update(query, {$push : {"members" : comm_member2}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
>  
> db.community.find(query)  
{ "_id" : ObjectId("5441ed6f86371da11f88be47"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과  
마음을 단련하세요", "openDate" : ISODate("2014-10-18T04:30:36.156Z"), "members" : [ { "email" :  
"syhan@nextree.co.kr", "name" : "한성진" }, { "name" : "송태훈", "email" : "song@nextree.co.kr" }, { "name" :  
"김현민", "email" : "kim@nextree.co.kr" } ]}  
>
```

2.6 Operators [6/7] – SubDocument 검색

- ✓ MongoDB는 Collection 내의 Document 단위로 검색이 됩니다.
- ✓ SubDocument 또는 배열 내의 SubDocument 단위로 매칭하여 검색 가능합니다.
- ✓ 매칭 조건이 SubDocument에 있는 경우 Dot 표기법에 따라 조회 Query를 구성할 수 있습니다.
- ✓ 배열 내의 SubDocument의 경우 \$elemMatch를 사용할 수 있습니다.

```
> query_member1 = {"name": "스포츠 커뮤니티", "members.email" : "song@nextree.co.kr"}  
> db.community.find(query_member1)  
{ "_id" : ObjectId("5441ed6f86371da11f88be47"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과  
마음을 단련하세요", "openDate" : ISODate("2014-10-18T04:30:36.156Z"), "members" : [ { "email" :  
"syhan@nextree.co.kr", "name" : "한성진" }, { "name" : "송태훈", "email" : "song@nextree.co.kr" }, { "name" :  
"김현민", "email" : "kim@nextree.co.kr" } ] }  
>  
> query_member2 = {"name": "스포츠 커뮤니티", "members" : {$elemMatch : {"email" : "kim@nextree.co.kr"}}}  
> db.community.find(query_member2)  
{ "_id" : ObjectId("5441ed6f86371da11f88be47"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과  
마음을 단련하세요", "openDate" : ISODate("2014-10-18T04:30:36.156Z"), "members" : [ { "email" :  
"syhan@nextree.co.kr", "name" : "한성진" }, { "name" : "송태훈", "email" : "song@nextree.co.kr" }, { "name" :  
"김현민", "email" : "kim@nextree.co.kr" } ] }  
>
```

2.6 Operators (7/7) – Update 연산자(\$pull)

✓ 배열 field에 값을 제거합니다.

✓ 연산자의 동작

- 배열 field에 값들이 있는 경우 값이 같은 것만 제거.
- 배열 field에 SubDocument들이 있는 경우 조건의 field와 값이 정확히 매치되는 것만 제거.

```
> db.community.update(query, {$pull : {"members" : {"email" : "song@nextree.co.kr"}}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
>  
> db.community.find(query)  
{ "_id" : ObjectId("5441ed6f86371da11f88be47"), "name" : "스포츠 커뮤니티", "description" : "운동을 통해 몸과  
마음을 단련하세요", "openDate" : ISODate("2014-10-18T04:30:36.156Z"), "members" : [ { "email" :  
"syhan@nextree.co.kr", "name" : "한성진" }, { "name" : "김현민", "email" : "kim@nextree.co.kr" } ] }  
>
```



3. 인덱스 (색인)

-
- 3.1 인덱스 개요
 - 3.2 B트리
 - 3.3 고유 인덱스
 - 3.4 희소 인덱스
 - 3.5 다중 키 인덱스
 - 3.6 공간 정보 인덱스
 - 3.7 explain 과 hint
 - 3.8 인덱스 변경하기

3.1 인덱스 개요

- ✓ 인덱스(Index)는 특정 문서를 탐색할 때 전부를 탐색하지 않고도 데이터를 찾을 수 있게 합니다.
- ✓ 특정 데이터를 쉽게 추출할 수 있도록 인덱스 데이터를 변경하는 것이 특징입니다.
- ✓ 모든 속성에 인덱스를 삽입할 수 있고 어떤 데이터 유형이라도 인덱스로 설정할 수 있습니다.
- ✓ 관계형 DB와 같이 내부적으로 B-트리(B-tree)로 인덱스를 생성하며, 다중 속성과 고유인덱스를 사용하여 복합 인덱스를 작성할 수 있습니다.

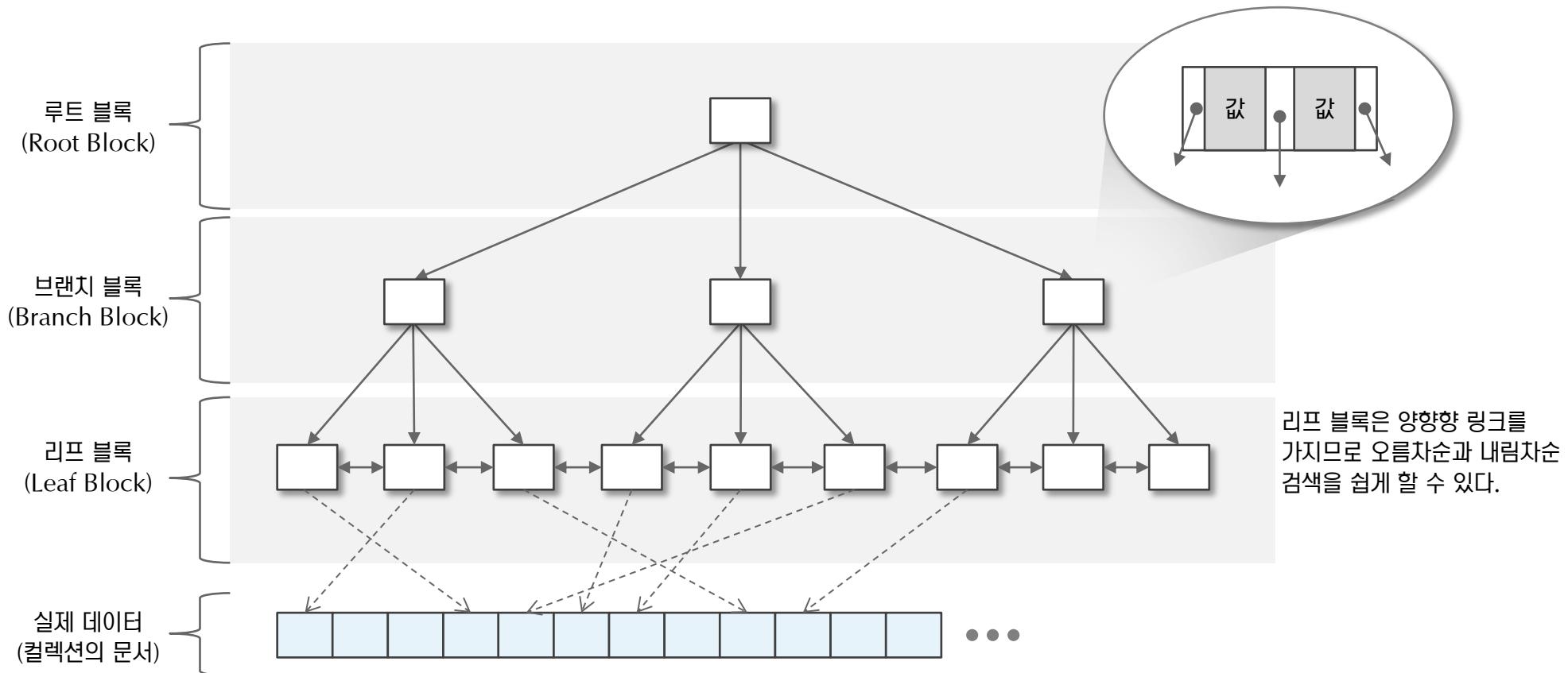


MongoDB의 공간 인덱스를 찾고 싶다. MongoDB 가이드 북에서 해당 내용을 가장 빨리 찾으려면 어떻게 하면 될까?

- ✓ (시간이 많은 경우) 한 장씩 넘기면서 찾는다.
- ✓ (보다 나은 방법) 목차에서 관련주제를 찾아 해당 페이지를 살펴본다.
- ✓ (가장 나은 방법) INDEX 페이지에서 관련 키워드를 찾아 해당 페이지를 살펴본다.

3.2 B트리

- ✓ MongoDB는 관계형 DB와 동일하게 인덱스를 위해 내부적으로 B-트리를 사용합니다.
- ✓ B-트리 인덱스는 브랜치 블록과 가장 아래에 있는 리프 블록으로 구성됩니다.
- ✓ 브랜치 블록 중에서 가장 위에 있는 블록을 루트 블록이라 합니다.



3.3 고유 인덱스 (unique index)

- ✓ 인덱스의 모든 엔트리가 고유해야 하는 경우 고유 인덱스를 사용합니다.
- ✓ 고유 인덱스를 생성하기 위해서는 unique 옵션을 지정합니다.

```
> db.employees.ensureIndex({empName: 1}, {unique: true})
```

- ✓ 고유 인덱스가 설정된 컬렉션에 이미 존재하는 키를 가진 문서를 삽입하면 예외가 발생합니다.
 - 드라이버를 사용하는 경우는 안전 모드를 사용할 때만 예외가 발생한다.

```
E1100 duplicate key error index:  
gardening.user.$username_1 dup key: { : 'Harold' }
```

- ✓ 중복 제거하기 (dropDups 옵션)
 - 중복 키가 존재하는 경우 고유 인덱스를 생성할 수 없다.
 - dropDups 옵션을 사용하면 중복 키를 가진 문서를 자동으로 삭제한 후 인덱스를 생성할 수 있다.
 - 중복된 문서 중에서 어떤 데이터를 삭제할지는 임의로 결정되므로 이 옵션은 주의해서 사용한다.

```
> db.employees.ensureIndex({empName: 1}, {unique: true, dropDups: true})
```

3.4 희소 인덱스 (sparse index)

- ✓ 밀집(dense) 인덱스가 적합하지 않은 경우 희소 인덱스를 사용합니다.
- ✓ 희소 인덱스에서는 인덱스의 키가 널이 아닌 값을 가지고 있는 문서만 존재합니다.
- ✓ 희소 인덱스를 생성하기 위해서는 {sparse : true} 옵션을 지정합니다.

```
> db.employees.ensureIndex({projectIds: 1}, {unique: true, sparse: true})
```

✓ 희소 인덱스가 필요한 경우

- 모든 도큐먼트가 다 가지고 있지는 않은 필드에 대해 고유 인덱스를 생성할 때
- 컬렉션에서 많은 수의 도큐먼트가 인덱스 키를 가지고 있지 않은 경우
(단, 값을 갖지 않는 필드에 대한 질의는 거의 하지 않는다는 가정하에...)

밀집인덱스 (dense index)

인덱스는 기본적으로 밀집(dense)하도록 설정되어 있다. 밀집 인덱스란 컬렉션 내의 한 문서가 인덱스 키를 갖지 않더라도 인덱스에는 해당 엔트리가 존재함을 의미한다. 예를 들어, 직원 컬렉션에 있는 소속 프로젝트(projectIds)를 인덱스 키로 설정했는데, 프로젝트를 수행하지 않는 직원이 있는 경우, projectIds 인덱스는 널 값을 갖는 엔트리를 갖게 된다.
아래와 같이 널 값에 대해 질의를 할 수 있다.

```
db.employee.find({projectIds: null})
```

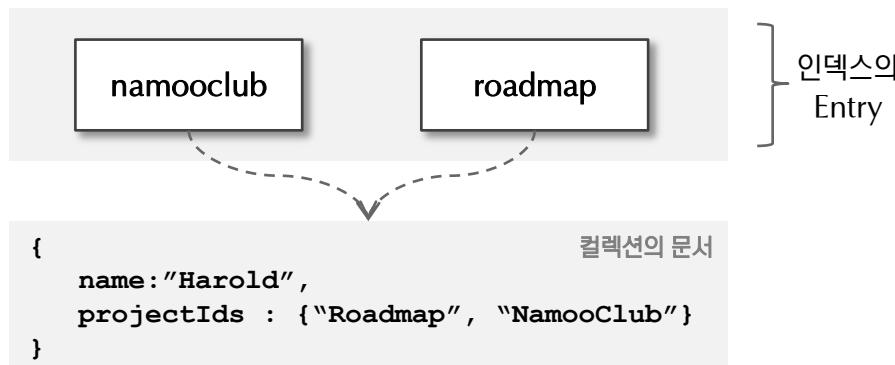
3.5 다중 키 인덱스 (multikey index)

- ✓ 인덱스 키로 사용된 필드의 값이 배열인 경우 다중 키 인덱스가 사용됩니다.
- ✓ 인덱스 내의 여러 개의 엔트리가 동일한 문서를 가리킬 수 있다는 의미입니다.

✓ 다중 키 인덱스가 사용되는 예

```
> db.employees.ensureIndex({ "projectIds" : 1 })
> db.employees.find({ "projectIds": "namooclub" })
```

- 직원이 수행하는 프로젝트(projectIds)를 나타내는 필드가 인덱스 키로 설정되어 있다.
- 한 직원이 여러 개의 프로젝트를 수행하는 경우, 해당 배열의 모든 값은 인덱스의 엔트리로 생성된다.
- 'roadmap' 또는 'namooclub'로 문서를 찾는 경우 인덱스를 통해 조회된다.



3.6 공간 정보 인덱스 [1/4]

- ✓ MongoDB는 공간 정보 인덱스라는, 좌표 평면 쿼리에 대한 특별한 형태의 인덱스를 제공합니다.
- ✓ 예를 들어, 현재 위치에서 가장 가까운 N개를 찾는 쿼리를 위해 인덱스를 적용할 수 있습니다.
- ✓ 공간 정보 인덱스 생성하기
 - 공간 정보 인덱스는 인덱스 키에 대한 값으로 1이나 -1 대신 '2d'를 설정한다.

```
> db.map.ensureIndex({ "gps" : "2d" })
```

- ✓ 공간 정보를 나타내는 필드의 값
 - 공간정보를 나타내는 필드의 값(예, gps)은 쌍으로 이루어진 값을 가져야 한다.
 - 두 요소를 가진 배열, 두 개의 키로 이루어진 내장 문서 등 (키로는 어떤 것도 쓸 수 있다)

```
{ "gps" : [0, 100] }  
{ "gps" : { "x" : -30, "y" : 30} }  
{ "gps" : { "latitude" : -180, "longitude" : 180} }
```

3.6 공간 정보 인덱스 [2/4]

✓ 좌표상으로 가장 가까운 문서 찾기

- \$near 조건절은 좌표를 지정하여, 거리가 가까운 순으로 모든 문서를 찾는다.
- 자원을 절약하기 위해 limit 함수로 대상을 줄일 수 있다.

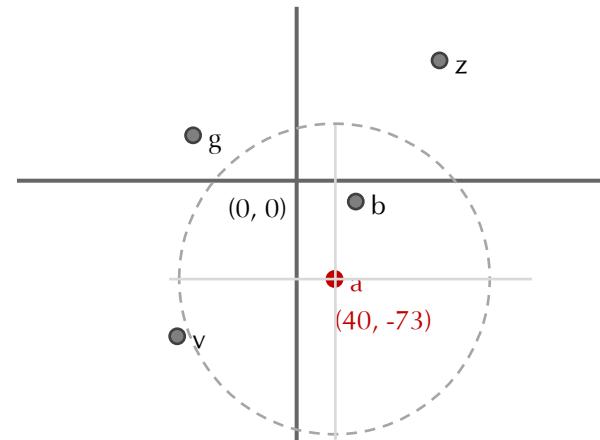
```
> db.map.find({"gps" : {"$near" : [40, -73]}}).limit(10)
```

→ 좌표 (40, -73)에서 가장 가까운 10개의 문서를 조회한다. limit을 생략하는 경우 가까운 순으로 모든 문서가 조회된다.

✓ geoNear 명령어

- 위 쿼리는 geoNear 명령어를 사용하여 나타낼 수도 있다.
- geoNear 명령어는 추가적으로 쿼리 지점으로부터 각 결과 문서의 거리를 반환한다.

```
> db.runCommand({geoNear : "map", near : [40, -73], num : 10});
```



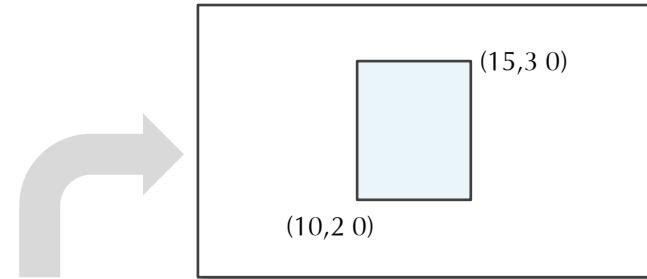
쿼리는 결과는
b → v → g → z
순으로 조회된다.

3.6 공간 정보 인덱스 (3/4)

✓ 특정 모양 내의 문서 찾기

- \$within 조건절을 사용하여 특정한 모양 내의 문서를 찾을 수 있다.
 - 자세한 내용은 온라인 설명서를 참고한다.

<http://docs.mongodb.org/manual/applications/geospatial-indexes/>



✓ 사각형 모양 내의 모든 문서 찾기

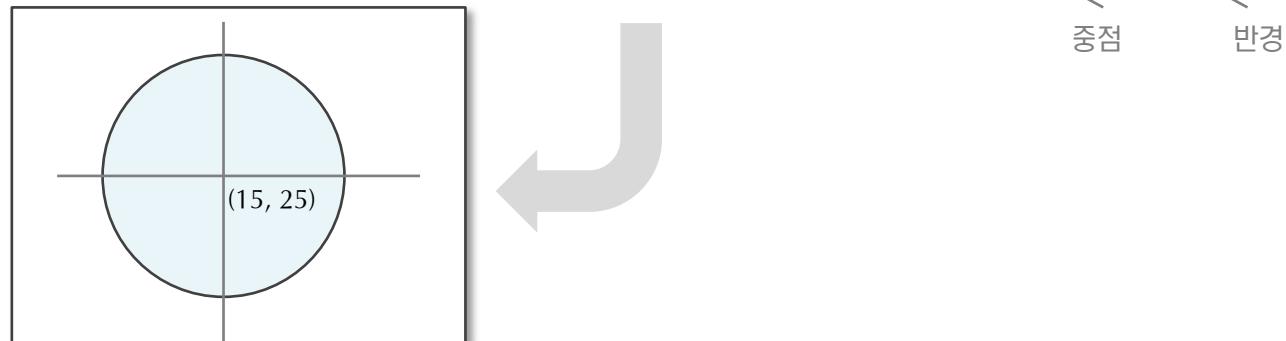
- \$box 옵션은 사각형을 나타내기 위해 두 좌표(좌하단, 우상단)을 사용한다.

```
db.map.find({"gps" : {"$within" : {"$box" : [[10, 20], [15, 30]]}}})
```

✓ 원 안의 모든 문서 찾기

- `$center` 옵션은 중점과 반경을 사용하여 원을 나타낸다.

```
db.map.find({ "gps" : { "$within" : { "$center" : [[15, 25], 5] } } })
```



3.6 공간 정보 인덱스 (4/4)

✓ 복합 공간 정보 인덱스

- 단순한 위치 뿐만 아니라 복잡한 조건으로 검색하기 위해 복합 인덱스를 사용할 수 있다.
- 예를 들어, 사용자가 위치한 곳으로부터 가장 가까운 곳에 위치한 커피숍 찾기 등

✓ 복합 인덱스 생성

```
> db.map.ensureIndex({"location" : "2d", "category" : 1})
```

✓ 가장 가까운 커피숍 찾기

```
> query = {"location" : {"$near" : [-70, 30]}, "desc" : "coffeeshop"}  
> db.map.find(query).limit(1)
```

3.7 explain과 hint (1/3)

- ✓ 인덱스가 제대로 수행되는지 확인하기 위하여 explain 기능을 사용할 수 있습니다.
- ✓ explain은 쿼리에서 사용하는 인덱스에 대한 정보와 시간에 대한 통계 및 살펴본 문서의 수를 반환합니다.
- ✓ 수행결과 분석 #1
 - 인덱스를 사용하지 않은 경우

```
> db.employees.find().explain()  
{  
  "cursor" : "BasicCursor", .....  
  "indexBounds" : [ ], .....  
  "nscanned" : 64, .....  
  "n" : 64, .....  
  "millis" : 0, .....  
  "allPlans" : [  
    {  
      "cursor" : "BasicCursor",  
      "indexBounds" : [ ]  
    }  
  ]  
}
```

▶ 쿼리가 인덱스를 사용하지 않았음을 의미한다.

▶ 데이터베이스가 스캔한 문서의 수
 가능한 이 두 개의 수는
 반환받은 문서의 수

▶ 가깝도록 하는 것이 좋다.

▶ 데이터베이스가 이 쿼리를 수행하는데 걸린 1/1000초 단위의 시간
(0을 목표로 하는 것이 좋다)

3.7 explain과 hint (2/3)

✓ 수행결과 분석 #2

- 인덱스를 사용한 경우

```
> query = {"age" : {"$gt" : 20, "$lt" : 30}};  
> db.employees.find(query).explain()  
{  
    "cursor" : "BtreeCursor age_1", .....  
    "indexBounds" : [  
        [  
            {"age" : 20}, {"age" : 30}  
        ]  
    ],  
    "nscanned" : 14,  
    "nscannedObjects" : 12  
    "n" : 12,  
    "millis" : 1,  
    "allPlans" : [  
        {  
            "cursor" : "BtreeCursor age_1",  
            "indexBounds" : [  
                [  
                    {"age" : 20}, {"age" : 30}  
                ]  
            ],  
        }  
    ]  
}
```

Btree 인덱스가 사용되었고
인덱스의 이름은 age_1이다.

allPlans는 MongoDB가 고려한
모든 실행 계획의 목록을 보여준다.

3.7 explain과 hint (3/3)

- ✓ hint 기능은 MongoDB가 쿼리에서 원하는 것과 다른 인덱스를 사용할 경우, 특정 인덱스를 사용하도록 강제할 수 있습니다.

```
> db.employees.find({ "age" : 14, "username" : /.*/ }).hint({ "username" : 1, "age" : 1 })
```

- MongoDB는 어떤 인덱스를 사용할지를 선택하기 위한 뛰어난 쿼리 옵티마이저를 가지고 있다. 대부분의 상황에서 hint는 불필요하다.

✓ 쿼리 옵티마이저의 동작과정

- 처음에 쿼리를 실행하면 쿼리 옵티마이저는 몇 개의 실행 계획을 동시에 시도한다.
- 가장 먼저 완료된 쿼리를 사용하고, 나머지 쿼리 실행은 종료한다.
- 이 쿼리 계획은 향후에 같은 키에 대한 쿼리를 할 때를 위해 기억해둔다.
- 쿼리 옵티마이저는 새로운 데이터가 추가되어 과거에 기억한 계획이 더 이상 최적이 아닌 경우를 위해 주기적으로 다른 계획을 시도한다.

3.8 인덱스 변경하기

✓ 인덱스 메타정보

- 인덱스에 대한 메타정보는 각 데이터베이스의 예약된 컬렉션인 system.indexes 컬렉션에 저장된다.
- system.indexes 컬렉션에는 임의로 문서를 추가하거나 삭제할 수 없으며, 명령어를 통해서만 조작할 수 있다.
- system.namespaces 컬렉션은 인덱스의 이름 목록을 갖고 있다.

✓ 인덱스 추가하기

- ensureIndex 명령어는 컬렉션에 새로운 인덱스를 추가한다.
- 인덱스의 생성은 오랜 시간이 걸리고 자원을 많이 사용한다.
- {"background" : true} 옵션을 사용하면 데이터베이스가 다른 요청을 처리하는 동안 백그라운드에서 인덱스를 생성할 수 있다.

```
db.employees.ensureIndex({ "name" : 1}, { "background" : true })
```

✓ 인덱스 삭제하기

- dropIndexes 명령어는 컬렉션의 인덱스를 삭제한다.

```
db.employees.dropIndex("name_1")
```

또는

```
db.runCommand({ "dropIndexes" : "employees", "index" : "name_1" })
```

- 모든 인덱스 제거

- dropIndexes 명령 수행 시 "index"의 값을 "*"로 넣는다.
- 또 다른 방법은 컬렉션을 제거하는 것이다. 컬렉션이 제거되면 모든 문서와 인덱스가 삭제된다.

→ 정확한 인덱스명을 알기 위해서는 system.indexes 컬렉션을 확인한다.

컬렉션의 이름

인덱스의 이름



4. 조합(Aggregate)

-
- 4.1 간단한 조합(Aggregate)
 - 4.2 맵-리듀스

4.1 간단한 조합 (1/3) – count : 컬렉션 내 문서의 수

- ✓ count 는 컬렉션 내의 문서 수를 반환하는 Aggregate 도구입니다.
 - 컬렉션 내 전체 문서의 수를 세는 것은 컬렉션의 크기와 상관없이 아주 빠릅니다.

```
> db.clubs.count()  
0  
> db.clubs.insert({ "name" : "MongoDB" })  
> db.clubs.count()  
1
```

- ✓ 조건을 만족하는 문서의 수
 - 쿼리를 전달할 수 있으며, 해당 쿼리의 결과 수를 반환합니다.
 - 단, 쿼리 조건을 추가하면 count는 느려집니다.

```
> db.clubs.insert("name" : "NoSQL")  
> db.clubs.count()  
2  
> db.clubs.count({ "name" : "NoSQL" })  
1
```

4.1 간단한 조합 (2/3) – distinct : 주어진 키의 고유한 값

- ✓ distinct 명령어는 주어진 키의 고유한 값을 찾습니다.

```
> db.컬렉션명.distinct(키)
```

또는

```
> db.runCommand({ "distinct" : 컬렉션명, "key" : 키명 })
```

- 동일한 명령으로 db.컬렉션명.distinct('키명')을 사용할 수 있다.

- ✓ 예를 들어 컬렉션에 다음과 같은 문서가 있는 경우, manager 키에 대하여 distinct를 수행한 결과는 다음과 같습니다.

```
{ "name" : "MongoDB", "manager" : "haroldkim" }  
{ "name" : "NoSQL" , "manager" : "haroldkim" }  
{ "name" : "BigData", "manager" : "tsong" }  
{ "name" : "SpringData", "manager" : "hkkang" }
```



```
> db.clubs.distinct("manager")  
{ "values" : ["haroldkim", "hkkang", "tsong"] , "ok" : 1 }
```

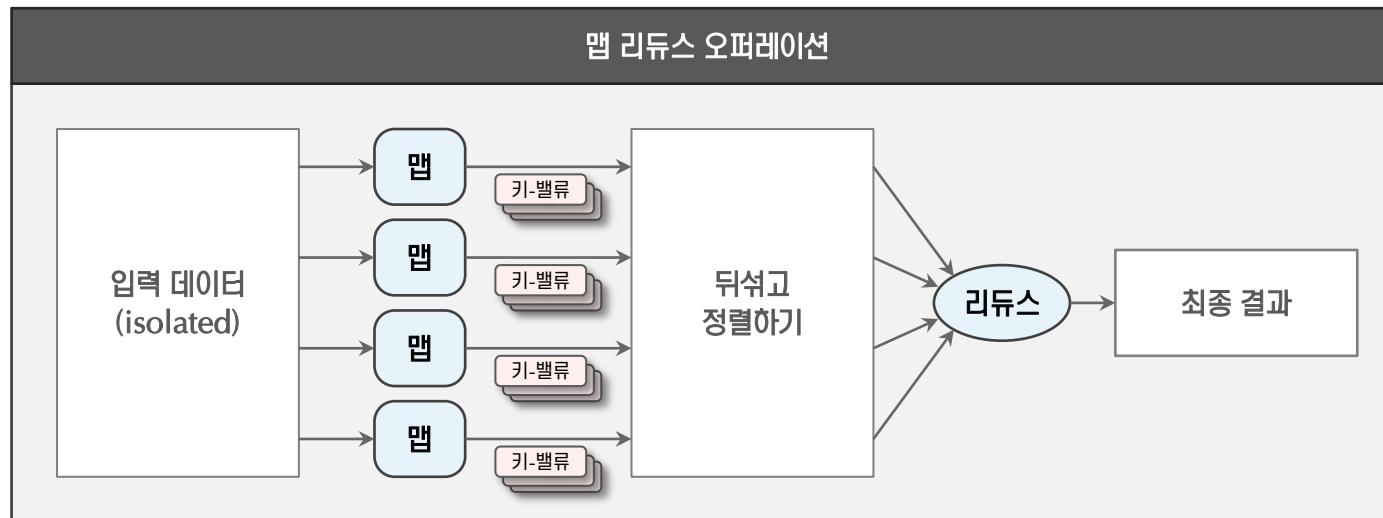
4.1 간단한 조합 (3/3) – group

- ✓ group 명령어는 문서들을 선택한 키를 기준으로 그룹으로 묶어 조합을 낼 때 사용합니다. (SQL의 GROUP BY와 유사)
- ✓ group 명령 인자들
 - key : 데이터를 어떻게 그룹으로 묶을 것인지를 정함
 - reduce : 결과 값에 대해 그룹으로 묶는 자바스크립트 함수
 - initial : 각 키 값에 대해 처음 반복할 때 리듀스 함수에게 제공하는 초기 문서
 - finalize : 그룹 명령의 결과 값을 반환하기 전에 각 그룹의 결과 값에 적용하는 자바스크립트 함수 (Optional)
- ✓ 예시 : 사용자 별로 작성한 리뷰의 수와 추천 수를 조합하여 반환합니다.

```
> result = db.reviews.group({  
  key      : {user_id : true},  
  initial  : {reviews : 0, votes : 0.0},  
  reduce   : function(doc, aggregator) {  
    aggregator.reviews += 1;  
    aggregator.votes += doc.votes;  
  },  
  finalize : function(doc) {  
    doc.average_votes = doc.votes / doc.reviews;  
  }  
)
```

4.2 맵-리듀스 (1/2)

- ✓ 맵-리듀스(map-reduce)는 group의 좀 더 유연한 버전입니다.
- ✓ 그룹 키를 세밀하게 제어할 수 있고, 새 컬렉션에 결과를 저장하는 것과 같은 다양한 출력 옵션을 가질 수 있어 데이터를 좀 더 유연한 방법으로 검색할 수 있습니다.



4.2 맵-리듀스 (2/2)

✓ 간단한 예제

- 컬렉션에서 모든 키 찾아 사용된 횟수 카운트하기

```
> map = function() {
    for (var key in this) {
        emit(key, {count : 1});
    }
};

> reduce = function(key, emits) {
    total = 0;
    for (var i in emits) {
        total += emits[i].count;
    }
    return {"count" : total};
};

> mr = db.foo.mapReduce(
    맵 함수 map,
    리듀스 함수 reduce,
    결과를 담을 컬렉션명 "results")
```

맵리듀스 문서는 연산에 대한 메타 정보 묶음을 반환한다.

```
→ {
    "result" : "results", 맵리듀스 결과가 저장된 컬렉션 명
    "timeMillis" : 12, 연산이 수행된 시간(1/1000초 단위)
    "count" : {
        "input" : 6, 맵 함수에 보내진 문서의 수
        "emit" : 14, 맵 함수에서 호출한 emit 수
        "output" : 5 결과 컬렉션에 생성된 문서의 수
    },
    "ok" : true
}
```

결과 컬렉션에서 find 메소드를 수행하면 모든 키와 그 사용 숫자를 볼 수 있다.

```
> db[mr.result].find()
{ "_id" : "_id", "value" : {"count" : 6} }
{ "_id" : "a", "value" : {"count" : 4} }
{ "_id" : "b", "value" : {"count" : 2} }
...
```



5. Java API

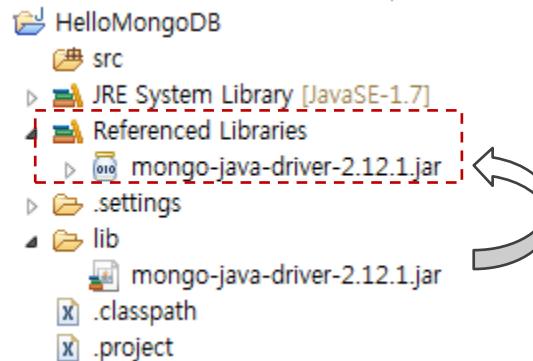
-
- 5.1 Java 드라이버
 - 5.2 인증
 - 5.3 컬렉션 목록조회 및 선택
 - 5.4 문서 추가
 - 5.5 문서 조회
 - 5.6 색인 추가
 - 5.7 데이터베이스 목록조회 및 삭제
 - 5.8 Java API 활용

5.1 Java 드라이버 (1/2)

- ✓ Java 프로그램에서 MongoDB에 연결하여 데이터베이스를 조작하기 위해서는 MongoDB 드라이버가 필요합니다.
- ✓ Java 드라이버를 클래스패스 상에 포함합니다.

✓ 드라이버 다운로드

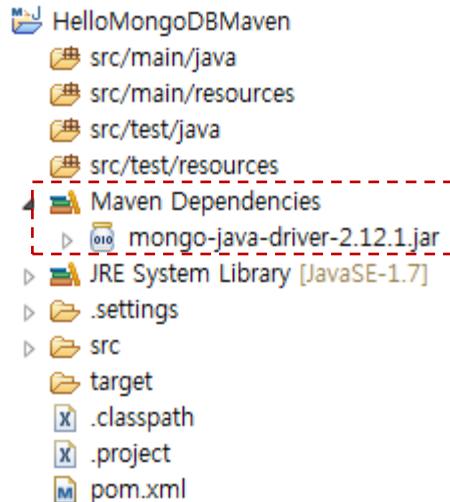
- mongo-java-driver-x.x.x.jar
- <http://docs.mongodb.org/ecosystem/drivers/java/>



lib에 위치시킨 jar 파일에서 우클릭 → BuildPath → Add to BuildPath 클릭

✓ Maven을 사용하는 경우

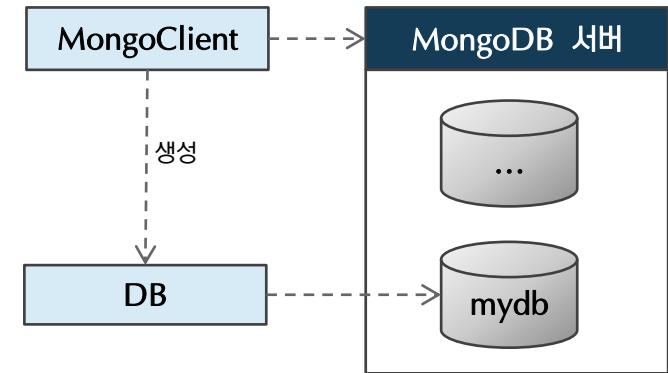
```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>2.12.1</version>
</dependency>
```



5.1 Java 드라이버 [2/2] – MongoDB 연결

- ✓ MongoClient 인스턴스는 데이터베이스 연결 정보를 보여줍니다.
- ✓ MongoClient로 부터 DB 객체를 획득하여 MongoDB 서버와 연결합니다.
- ✓ MongoDB 서버에 접속하는 방법은 여러 가지입니다.

```
// 방법1. 디폴트로 localhost의 27017포트로 접속한다.  
MongoClient mongoClient = new MongoClient();  
  
// 방법2. 서버 주소로 접속하기 (디폴트 포트번호(27017)가 사용된다)  
MongoClient mongoClient = new MongoClient("localhost");  
  
// 방법3. 서버주소와 포트번호로 접속하기  
MongoClient mongoClient = new MongoClient("localhost", "27017");  
  
// 방법4. 복제셋에 연결하기 원하는 경우 복제 서버들을 나열한다  
MongoClient mongoClient = new MongoClient(Arrays.asList(  
    new ServerAddress("localhsot", 27017),  
    new ServerAddress("localhsot", 27018),  
    new ServerAddress("localhsot", 27019)));  
  
// mydb 데이터베이스를 조작할 수 있는 객체 획득  
DB db = mongoClient.getDB( "mydb" );
```



5.2 인증

- ✓ 보안모드에서 동작하는 MongoDB에 인증하기 위하여 이름과 암호를 사용합니다.
- ✓ 여러 개의 데이터베이스에 대한 인증이 필요한 경우, MongoCredential 객체를 배열로 전달합니다.

```
MongoCredential credential = MongoCredential.createMongoCRCredential(  
    "namoouser", "namoodb", "pass1234".toCharArray());  
  
String server = "localhost";  
MongoClient mongoClient = new MongoClient(new ServerAddress(server),  
    Arrays.asList(credential));
```

MongoDB 버전 3.0 이상인 경우 MongoCredential.createMongoCRCredential 대신 MongoCredential.createScramSha1Credential 을 사용합니다. (드라이버 버전 2.13 이상 지원)

데이터베이스 사용자 추가

데이터베이스에 사용자를 추가하기 위해서는 MongoDB 웹에서 createUser 메소드를 사용한다. createUser 메소드의 첫 번째 인자는 사용자 정보이고, 두 번째 인자는 사용자에게 주어질 역할이다.

사용 가능한 역할에 대한 자세한 내용은 <http://docs.mongodb.org/manual/reference/built-in-roles/> 를 참조한다.

```
> use namoodb  
> db.createUser({user : "namoouser", pwd : "pass1234", roles : ["dbOwner"]})
```

5.3 컬렉션 목록 조회 및 선택

- ✓ `getCollectionNames()` 메소드는 데이터베이스가 가진 모든 컬렉션 목록을 조회합니다.

```
Set<String> collectionNames = db.getCollectionNames();
for (String collectionName : collectionNames) {
    System.out.println(collectionName);
}
```

- ✓ `getCollection(String collectionName)` 메소드는 컬렉션에 문서를 추가하거나 쿼리를 수행합니다.

```
DBCollection coll = db.getCollection("communities");
```

5.4 문서 추가

- ✓ 컬렉션 객체의 insert() 메소드를 사용하면 컬렉션에 문서를 추가할 수 있습니다.
- ✓ BasicDBObject 객체로 문서를 생성하고 insert() 메소드로 컬렉션에 추가합니다.

```
// 문서의 생성
BasicDBObject doc = new BasicDBObject("name", "MongoDB").
    append("manager", "hyunohkim").
    append("openDate", new Date());

// 컬렉션에 문서 추가
coll.insert(doc);

// 컬렉션의 문서 개수
System.out.println("Count of documents : " + coll.getCount());
```

5.5 문서 조회

- ✓ 컬렉션 안의 모든 문서를 얻으려면 `find()` 메소드를 사용한다. 이 메소드는 `DBCursor` 객체를 반환하며, 반복문을 사용하여 전체 문서를 살펴볼 수 있습니다.
- ✓ `DBCursor` 객체는 DB 자원이므로 사용이 끝난후에는 `close()` 메소드를 호출하여 자원을 반납합니다.

```
DBCursor cursor = coll.find();
try {
    while (cursor.hasNext()) {
       DBObject document = cursor.next();
        System.out.println(document);
    }
} finally {
    cursor.close();
}
```

5.6 색인 추가

- ✓ MongoDB의 컬렉션에 색인을 생성하려면 해당 필드를 지정하고 오름차순(1)이나 내림차순(-1)을 지정하면 됩니다.
- ✓ `createIndex()` 메소드는 컬렉션에 색인을 추가합니다.

```
coll.createIndex(new BasicDBObject("name", 1));
```

- ✓ `getIndexInfo()` 메소드는 컬렉션에 적용된 색인 목록을 조회합니다.

```
List<DBObject> indexes = coll.getIndexInfo();  
  
for (DBObject index : indexes) {  
    System.out.println(index);  
}
```

5.7 데이터베이스 목록 조회 및 삭제

- ✓ `getDatabaseNames()` 메소드는 유효한 데이터베이스 목록을 조회합니다.

```
for (String dbName : mongoClient.getDatabaseNames()) {  
    System.out.println(dbName);  
}
```

- ✓ `dropDatabase()` 메소드는 데이터베이스 이름을 가지고 데이터베이스를 삭제합니다.

```
mongoClient.dropDatabase("namoodb");
```

5.8 Java API 활용 (1/6) – 생성

- ✓ 컬렉션 이름으로 DBCollection 객체를 가져옵니다.
- ✓ 도메인 객체를 BasicDBObject 타입의 문서 객체로 변환합니다.
- ✓ 문서 객체를 DBCollection API를 이용하여 입력합니다.
- ✓ 입력시 생성된 '_id' 필드는 ObjectId 객체로 꺼내올 수 있습니다.

```
DB db = MongoClientFactory.getDB();
DBCollection collection = db.getCollection(TownerCollectionName);

BasicDBObject doc = createDocument(towner);
collection.insert(doc);

String townerId = doc.getObjectId("_id").toString();
towner.setId(townerId);
```

5.8 Java API 활용 [2/6] – id로 조회

- ✓ Document의 고유 id로 문서 객체 하나를 조회할 수 있습니다.
- ✓ ObjectId 객체로 래핑하여 findOne API를 호출합니다.
- ✓ 조회된 문서객체를 도메인 객체로 변환합니다.

```
DB db = MongoClientFactory.getDB();
DBCollection coll = db.getCollection(TownerCollectionName);
DBObject doc = coll.findOne(new ObjectId(townerId));
return mapToTowner((BasicDBObject) doc);
```

5.8 Java API 활용 (3/6) – 목록 조회

- ✓ 여러 Document를 조회하는 경우 DBCursor를 사용합니다.
- ✓ DBCursor의 hasNext(), next() 를 활용합니다.
- ✓ 매치 조건이 SubDocument에 있는 경우 Dot 표기법을 이용합니다.

```
DB db = MongoClientFactory.getDB();
DBCollection coll = db.getCollection(CLubCollectionName);

DBCursor cursor = null;
List<Club> clubs = new ArrayList<Club>();
try {
    cursor = coll.find(new BasicDBObject("admin.email", adminEmail));
    while (cursor.hasNext()) {
        clubs.add(mapToClub((BasicDBObject) cursor.next()));
    }
} finally {
    if (cursor != null) cursor.close();
}
return clubs;
```

5.8 Java API 활용 [4/6] – 업데이트

- ✓ 해당 Document 전체를 업데이트 하는 경우입니다.
- ✓ Update API를 사용합니다.
- ✓ 업데이트할 대상 객체를 찾기 위한 Query와 업데이트할 문서객체가 필요합니다.

```
DB db = MongoClientFactory.getDB();
DBCollection coll = db.getCollection(TownerCollectionName);
DBObject doc = createDocument(towner);
coll.update(new BasicDBObject("_id", new ObjectId(towner.getId()))), doc);
```

5.8 Java API 활용 (5/6) – 일부 업데이트

- ✓ Document의 일부 field만 업데이트 할 경우 \$set 연산자를 활용합니다.
- ✓ Query 객체 및 업데이트할 객체는 BasicDBObject를 중첩하여 사용할 수 있는데 이는 만들어질 Query 문서 구조와 관련 있습니다.
- ✓ 따라서 Query 문서를 직접 만들어 테스트 후 Query 코드를 작성하면 편리합니다.

```
DBObject adminDoc = createDocument(admin);
DBObject query = new BasicDBObject("_id", new ObjectId(clubId));
DBObject updateDoc = new BasicDBObject(
    "$set", new BasicDBObject("admin", adminDoc));
coll.update(query, updateDoc);
```

5.8 Java API 활용 [6/6] – 삭제

- ✓ 해당 Document를 삭제합니다.
- ✓ Remove API를 사용하며 삭제할 대상 Document를 찾기 위한 Query가 필요합니다.

```
DB db = MongoClientFactory.getDB();
DBCollection coll = db.getCollection(TownerCollectionName);

coll.remove(new BasicDBObject("_id", new ObjectId(towner.getId())));
```

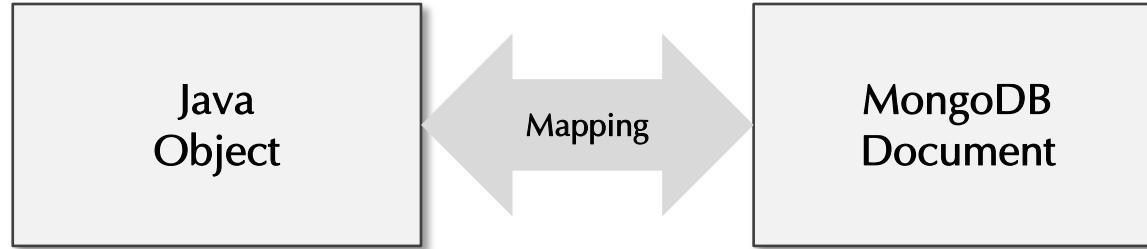


6. MongoDB 활용

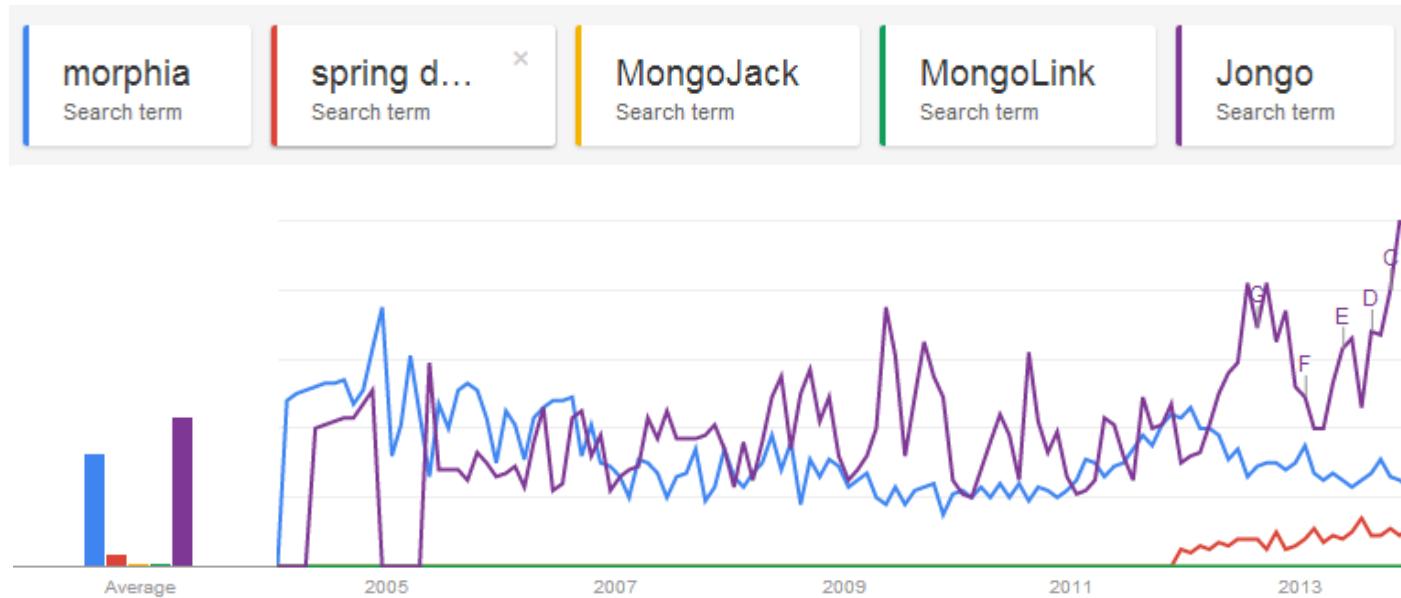
-
- 6.1 POJO 매퍼 개요
 - 6.2 Jongo
 - 6.3 Morphia
 - 6.4 Spring Data MongoDB

6.1 POJO 매퍼 개요

- ✓ POJO 매퍼는 MongoDB 문서와 Java 객체간의 상호 변환을 위한 맵핑 기능을 제공합니다.
- ✓ morphia, Spring Data MongoDB, MongoJack, MonoLink, Jongo 등이 있습니다.



- ✓ 구글 트렌드 비교



6.2 Jongo

- ✓ Jongo는 MongoDB 셸에서 수행하는 쿼리를 Java에서 그대로 사용할 수 있도록 합니다.
- ✓ 문서를 추가하거나 조회할 때 컬렉션의 문서와 Java 객체를 자동으로 변환합니다.
- ✓ <http://jongo.org/>



Query in **Java** as in **Mongo shell**

SHELL

```
db.friends.find({age: {$gt: 18}})
```

JAVA DRIVER

```
friends.find(new BasicDBObject("age", new BasicDBObject("$gt", 18)))
```

JONGO

```
friends.find("{age: {$gt: 18}}").as(Friend.class)
```

6.3 Morphia

- ✓ Morphia는 MongoDB문서와 Java 객체간의 매핑을 제공하는 경량 라이브러리입니다.
- ✓ <https://github.com/mongodb/morphia/wiki>

The screenshot shows the GitHub repository page for 'mongodb / morphia'. The page has a header with a search bar and navigation links for Explore, Gist, Blog, and Help. Below the header, it displays the repository name 'PUBLIC mongodb / morphia'. The main content area is titled 'Home' and shows a recent edit by 'evanchooly' on Feb 21 with 8 commits. The page then features a large section titled 'Morphia' which describes the library as a lightweight type-safe library for mapping Java objects to/from MongoDB. It highlights features like a typesafe and fluent Query API with runtime validation, annotations instead of XML files, and comfort for JPA developers. Below this is a 'Features' section listing benefits such as lifecycle support, integration with DI frameworks, extension points, and advanced mapper support for raw conversion. At the bottom, there's a note to continue reading the QuickStart or looking at annotations.

This repository Explore Gist Blog Help

PUBLIC [mongodb / morphia](#) [Watch](#)

Home

evanchooly edited this page on Feb 21 · 8 commits

Morphia

Morphia is a lightweight type-safe library for mapping Java objects to/from MongoDB. Morphia provides a typesafe, and fluent [Query](#) API support with (runtime) validation. Morphia uses annotations so there are no XML files to manage or update. Morphia should feel very comfortable for any developer with JPA experience.

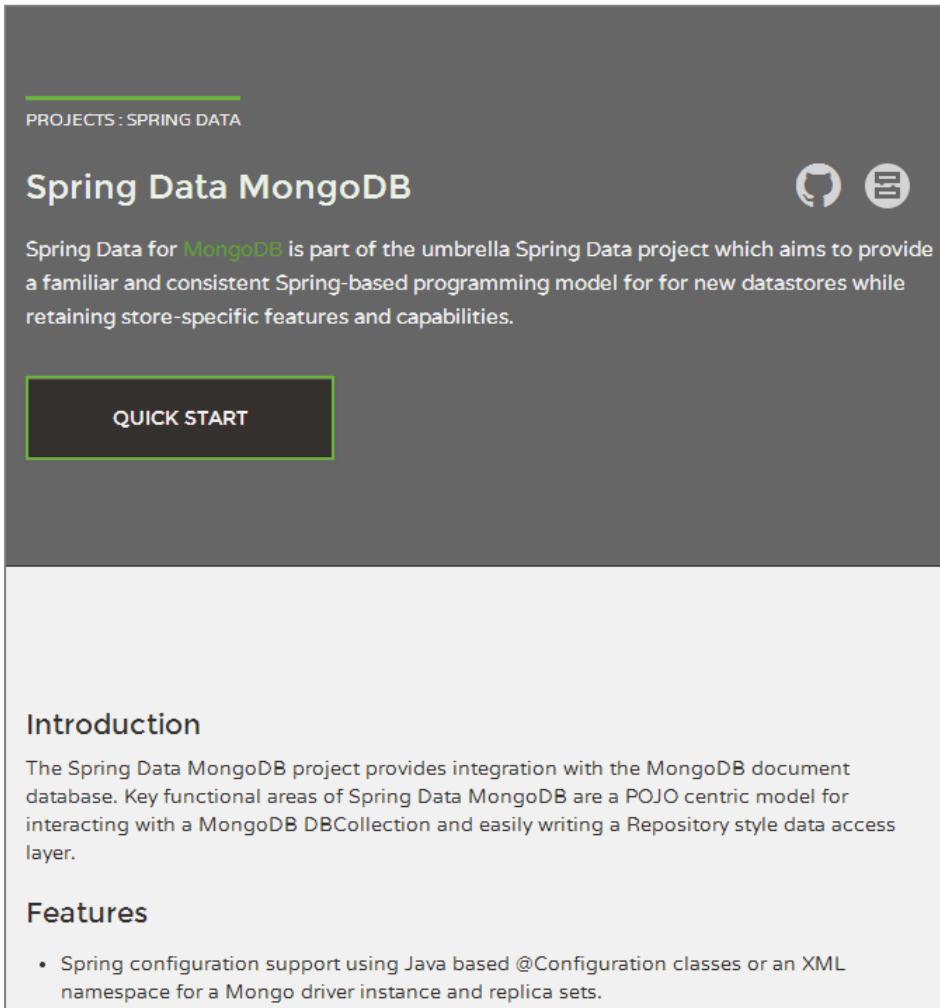
Features

- [Lifecycle Method/Event Support](#)
- Works great with Guice, Spring, and other DI frameworks.
- Many extension points (new annotations, converters, mapping behavior, logging, etc.)
- Does not store Null/Empty values (by default).
- GWT support (entities are just POJOs) -- (GWT ignores annotations)
- Advanced mapper which allows raw conversion, `void toObject(DBObject)` or `DBObject fromObject(Object)`

Please continue by reading the [QuickStart](#) or looking at a list of [the annotations](#).

6.4 Spring Data MongoDB

- ✓ Spring 프레임워크의 안정적인 지원이 가능하며 다른 Spring 기술과 자연스럽게 연계가 가능합니다.
- ✓ Data Access 계층의 일관된 접근방식을 취하므로 Spring Data에서 지원하는 다른 Database와의 기술이전이 쉽습니다



The screenshot shows the official Spring Data MongoDB project page. At the top, it says "PROJECTS : SPRING DATA" and "Spring Data MongoDB". Below that, there's a brief description: "Spring Data for MongoDB is part of the umbrella Spring Data project which aims to provide a familiar and consistent Spring-based programming model for new datastores while retaining store-specific features and capabilities." A "QUICK START" button is visible. The main content area is titled "Introduction" and describes the project's purpose: "The Spring Data MongoDB project provides integration with the MongoDB document database. Key functional areas of Spring Data MongoDB are a POJO centric model for interacting with a MongoDB DBCollection and easily writing a Repository style data access layer." Below this is a "Features" section with a single bullet point: "Spring configuration support using Java based @Configuration classes or an XML namespace for a Mongo driver instance and replica sets."

✓ 토의

감사합니다...

- ❖ 송태국(tsong@nextree.co.kr)
- ❖ 넥스트리컨설팅(주)
- ❖ www.nextree.co.kr