

MONGO DB

KH정보교육원

2

3. Mongo DB

MongoDB 특징

3

□ MONGODB?

- MongoDB는 C++로 작성된 오픈소스 문서지향 (Document-Oriented) 적 Cross-platform 데이터베이스

□ NoSQL?

- Not Only SQL 기존의 RDBMS의 한계를 극복하기 위해 만들어진 새로운 형태의 데이터저장소 입니다. 관계형 DB가 아니므로, RDMS처럼 고정된 스키마 및 JOIN 이 존재하지 않습니다.

MongoDB 특징

4

Document?

Document Oriented 데이터베이스?

Document는 RDMS의 record(개체) 와 비슷한 개념, 이의 데이터 구조는 한개이상의 key-value pair 으로 이뤄져있습니다



MongoDB 특징

5

```
{
  "_id": ObjectId("5099803df3f4948bd2f98391"),
  "username": "velopert",
  "name": { first: "M.J.", last: "Kim" }
}
```

- `_id`, `username`, `name` 은 key 이고 그 오른쪽에 있는 값들은 value 입니다.
- `_id` 는 12bytes의 hexadecimal(16진수) 값으로서, 각 document의 유일함(uniqueness)을 제공합니다.(rdms의 unique)
이 값의 첫 4bytes 는현재 timestamp, (5099803d)
- 다음 3bytes는 machine id, (f3f494)
- 다음 2bytes는 MongoDB 서버의 프로세스id, (8bd2)
- 마지막 3bytes는 순차번호입니다(f98391) 추가될때마다 값이 높아진다.
- Document는 동적(dynamic)의 schema 를 갖고있습니다. 같은 Collection 안에 있는 Document끼리 다른 schema 를 갖고 있을 수 있다, 즉 서로 다른 데이터 (즉 다른 key) 들을 가지고 있을 수 있습니다.

MongoDB 특징

6

□ Collection?

- Collection은 MongoDB Document의 그룹입니다. Document들이 Collection내부에 위치하고 있습니다. RDMS의 table과 비슷한 개념입니다만 RDMS와 달리 schema를 따로 가지고 있지않습니다. Document 부분설명에 나와있듯이 각 Document들이 동적인 schema를 가지고 있다.

□ Database?

- Database는 Collection들의 물리적인 컨테이너입니다. 각 Database는 파일시스템에 여러파일들로 저장됩니다.

RDBMS VS NOSQL

7

Relational Database Management System (관계형 데이터베이스 관리 시스템) 행과 열로 된 2차원의 table로 데이터를 관리하는 데이터베이스 시스템입니다. Mysql, Oracle Database, DB2 등 시스템들이 RDMS에 속함

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple / Row	Document
Column	Key / Field
Table Join	Embedded Documents
Primary Key	Primary Key (_id)
Database Server & Client	
mysqld	mongod
mysql	mongo

MongoDB 장점

8

- Schema-less (Schema가 없다. 같은 Collection 안에 있을지라도 다른 Schema를 가지고 있을 수 있다)
- 각 객체의 구조가 뚜렷하다
- 복잡한 JOIN 이 없다.
- Deep Query ability (문서지향적 Query Language 를 사용하여 SQL 만큼 강력한 Query 성능을 제공한다.
- 어플리케이션에서 사용되는 객체를 데이터베이스에 추가 할 때 Conversion / Mapping이 불필요하다.

MongoDB 설치하기

9

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

The screenshot shows the MongoDB Download Center interface. At the top, there is a navigation bar with links for Products, Solutions, Customers, Resources, Learn, What is MongoDB?, Contact, Search, Sign In, and a Try Free button. Below the navigation bar is a large banner with the text "MongoDB Download Center". Under the banner, there are three tabs: Cloud, Server, and Tools. The Server tab is selected. Below the tabs, there is a section titled "Select the server you would like to run:". There are two options: "MongoDB Community Server" (highlighted with a green bar and the text "FEATURE RICH. DEVELOPER READY.") and "MongoDB Enterprise Server" (with a grey bar and the text "ADVANCED FEATURES. PERFORMANCE GRADE."). Below these options, there are three dropdown menus: "Version" (set to 4.0.10 (current release)), "OS" (set to Windows 64-bit x64), and "Package" (set to ZIP). To the right of these dropdowns is a large green "Download" button. Further to the right, there is a list of links: "Release notes", "Changelog", "All version binaries", and "Installation instructions".

mongoDB. Products Solutions Customers Resources Learn What is MongoDB? Contact Search Sign In Try Free

MongoDB Download Center

Cloud Server Tools

Select the server you would like to run:

MongoDB Community Server
FEATURE RICH. DEVELOPER READY.

MongoDB Enterprise Server
ADVANCED FEATURES. PERFORMANCE GRADE.

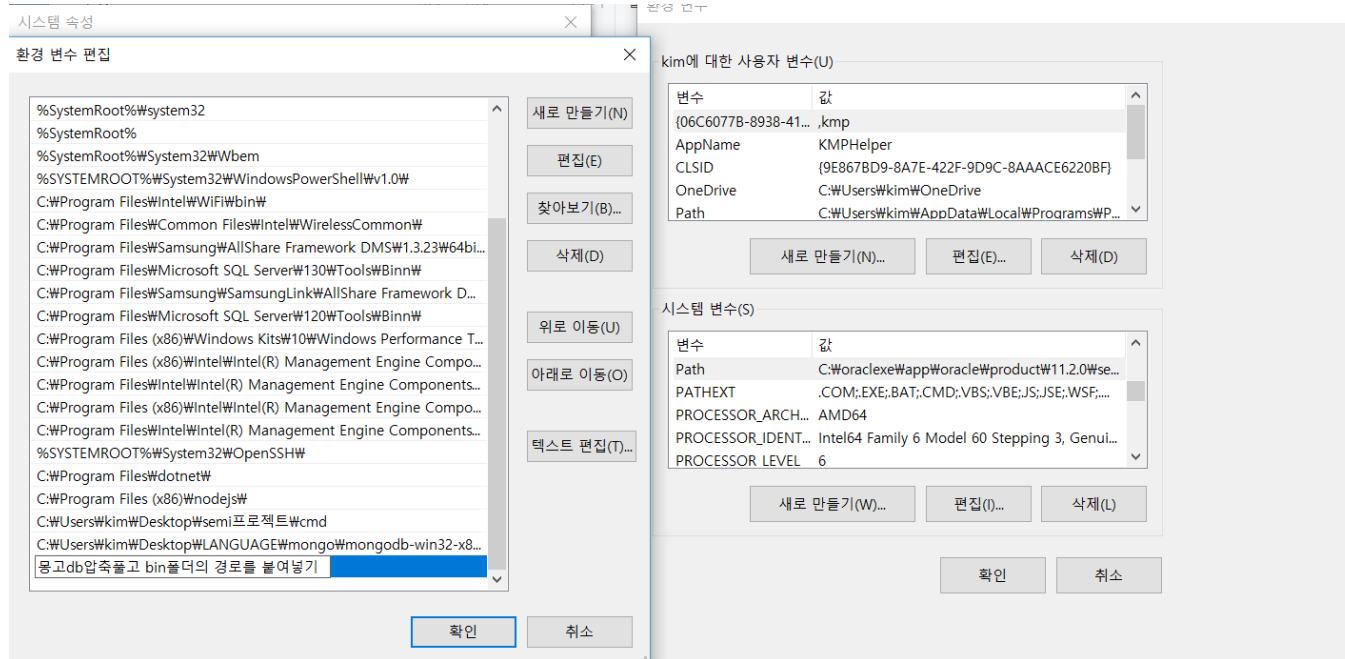
Version: 4.0.10 (current release) OS: Windows 64-bit x64 Package: ZIP

Download

- Release notes
- Changelog
- All version binaries
- Installation instructions

MongoDB 설치하기

10



[내컴퓨터] 우클릭 -> [고급시스템설정] -> [환경변수(N)]

시스템 환경변수 path에 MongoDB 설치폴더 등록

예) C:\Users\kim\Desktop\LANGUAGE\mongo\mongodb

-win32-x86_64-2008plus-ssl-4.0.10\mongodb-win32-x86_64-2008plus-ssl-4.0.10\bin

MongoDB 서버 실행

11

1. 데이터를 저장할 폴더 생성

C:\Users\Wkim> mongod --dbpath (자신이 만들어놓은 폴더의 경로)
"waiting for connections on port 27017" 연결성공

자신이 만들어놓은 data폴더에 아래와 같은 파일들이 생성됨

LANGUAGE > mongo > mongodb-win32-x86_64-2008plus-ssl-4.0.10 > mongodb-win32-x86_64-2008plus-ssl-4.0.10 > data >

이름	수정한 날짜	유형	크기
diagnostic.data	2019-07-11 오후 5...	파일 폴더	
journal	2019-07-11 오후 5...	파일 폴더	
_mdb_catalog	2019-07-11 오후 5...	WT 파일	16KB
collection-0--9085534076351479125	2019-07-11 오후 5...	WT 파일	16KB
collection-2--9085534076351479125	2019-07-11 오후 5...	WT 파일	16KB
collection-4--9085534076351479125	2019-07-11 오후 5...	WT 파일	4KB
index-1--9085534076351479125	2019-07-11 오후 5...	WT 파일	16KB
index-3--9085534076351479125	2019-07-11 오후 5...	WT 파일	16KB
index-5--9085534076351479125	2019-07-11 오후 5...	WT 파일	4KB
index-6--9085534076351479125	2019-07-11 오후 5...	WT 파일	4KB
mongod.lock	2019-07-11 오후 5...	LOCK 파일	1KB
sizeStorer	2019-07-11 오후 5...	WT 파일	16KB
storage.bson	2019-07-11 오후 5...	BSON 파일	1KB
WiredTiger	2019-07-11 오후 5...	파일	1KB
WiredTiger.lock	2019-07-11 오후 5...	LOCK 파일	1KB
WiredTiger.turtle	2019-07-11 오후 5...	TURTLE 파일	2KB
WiredTiger	2019-07-11 오후 5...	WT 파일	60KB
WiredTigerLAS	2019-07-11 오후 5...	WT 파일	4KB

- 3. 새로운 cmd 실행(기존 cmd는 서버창으로서 새로운cmd를 실행함)
- mongo(환경변수를 설정했을시)
 cd (bin폴더경로) -> mongo (환경변수를 설정하지 않았을때)

cmd 명령 프롬프트 - mongo

```
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\kim>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d0db76de-073d-4678-a882-bac17c167c3c") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-07-11T01:19:00.754-0700 | CONTROL | [initandlisten]
2019-07-11T01:19:00.754-0700 | CONTROL | [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-07-11T01:19:00.757-0700 | CONTROL | [initandlisten] **      Read and write access to data and configuration is unrestricted.
2019-07-11T01:19:00.759-0700 | CONTROL | [initandlisten]
2019-07-11T01:19:00.762-0700 | CONTROL | [initandlisten] ** WARNING: This server is bound to localhost.
2019-07-11T01:19:00.764-0700 | CONTROL | [initandlisten] **      Remote systems will be unable to connect to this server.
2019-07-11T01:19:00.766-0700 | CONTROL | [initandlisten] **      Start the server with --bind_ip <address> to specify which IP
2019-07-11T01:19:00.767-0700 | CONTROL | [initandlisten] **      addresses it should serve responses from, or with --bind_ip_all to
2019-07-11T01:19:00.768-0700 | CONTROL | [initandlisten] **      bind to all interfaces. If this behavior is desired, start the
2019-07-11T01:19:00.768-0700 | CONTROL | [initandlisten] **      server with --bind_ip 127.0.0.1 to disable this warning.
2019-07-11T01:19:00.769-0700 | CONTROL | [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> db
test
```

오류발생시

13

- #msvcp140.dll 오류발생
- <https://www.microsoft.com/ko-kr/download/details.aspx?id=48145>

원하는 다운로드 선택

<input type="checkbox"/> 파일 이름	크기
<input type="checkbox"/> vc_redist.x64.exe	13.9 MB
<input type="checkbox"/> vc_redist.x86.exe	13.1 MB

DataBase 생성 제거

14

- **Database 생성: *use***
- *>use DATABASE_NAME*
- **현재 사용중 DB조회:db**
- *>db*
- **전체 DB조회:show dbs**
- *>show dbs*
- **DataBase 제거:*db.dropDatabase()***
- *>use 원하는 데이터베이스이름*
- *>db.dropDatabase();*

예제

15

```
> use mongodb_tutorial
switched to db mongodb_tutorial
> db
mongodb_tutorial
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
> db.book.insert({"name": "MongoDB Tutorial", "author": "velopert"});
WriteResult({ "nInserted" : 1 })
> show dbs
admin          0.000GB
config         0.000GB
local          0.000GB
mongodb_tutorial 0.000GB
test           0.000GB
> use mongodb_tutorial
switched to db mongodb_tutorial
> db.dropDatabase();
{ "dropped" : "mongodb_tutorial", "ok" : 1 }
```

Collection 생성, 제거

16

- Collection 생성:
- *db.createCollection(name, [options])*
- Collection 제거:
- *db.COLLECTION_NAME.drop()*

Option:

Field	Type	설명
capped	Boolean	이 값을 true 로 설정하면 capped collection 을 활성화 시킵니다. Capped collection 이란, 고정된 크기(fixed size) 를 가진 컬렉션으로서, size 가 초과되면 가장 오래된 데이터를 덮어씁니다. 이 값을 true로 설정하면 size 값을 꼭 설정해야 합니다.
autoIndex	Boolean	이 값을 true로 설정하면, _id 필드에 index를 자동으로 생성합니다. 기본값은 false 입니다.
size	number	Capped collection 을 위해 해당 컬렉션의 최대 사이즈(maximum size)를 ~ bytes 로 지정합니다.
max	number	해당 컬렉션에 추가 할 수 있는 최대 갯수를 설정합니다.

예제

17

```
> use test
switched to db test
> db.createCollection("books")
{ "ok" : 1 }
> db.createCollection("articles", {
...   capped: true,
...   size: 6142800,
...   max: 10000
... })
{ "ok" : 1 }
> db.people.insert({"name": "kh"})
WriteResult({ "nInserted" : 1 })
> show collections
articles
books
people
> use test
switched to db test
> show collections
articles
books
people
> db.people.drop()
true
> show collections
articles
books
>
```

Document 추가, 조회, 제거

18

- Document 추가:
- *db.COLLECTION_NAME.insert(document)*
- Document 조회:
- *db.COLLECTION_NAME.find()*
- Document 제거
- *:db.COLLECTION_NAME.remove(criteria, justOne)*

parameter	type	설명
*criteria	document	삭제 할 데이터의 기준 값 (criteria) 입니다. 이 값이 {} 이면 컬렉션의 모든 데이터를 제거합니다.
justOne	boolean	선택적(Optional) 매개변수이며 이 값이 true 면 1개 의 다큐먼트만 제거합니다. 이 매개변수가 생략되면 기본값은 false 로 서, criteria에 해당되는 모든 다큐먼트를 제거합니다.

예제

19

명령 프롬프트 - mongo

```
books
> db.books.insert({"name": "Mongo Guide", "author": "kh"})
WriteResult({"nInserted" : 1 })
> db.books.insert([
... .. {"name": "Book1", "author": "kh"},
... .. {"name": "Book2", "author": "kh"}
... .. ]);
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
> db.books.find()
{ "_id" : ObjectId("5d2723b92c00dbef88e1df92"), "name" : "Mongo Guide", "author" : "kh" }
{ "_id" : ObjectId("5d2723e22c00dbef88e1df93"), "name" : "Book1", "author" : "kh" }
{ "_id" : ObjectId("5d2723e22c00dbef88e1df94"), "name" : "Book2", "author" : "kh" }
> db.books.find({"name": "Book1"})
{ "_id" : ObjectId("5d2723e22c00dbef88e1df93"), "name" : "Book1", "author" : "kh" }
> db.books.find()
{ "_id" : ObjectId("5d2723b92c00dbef88e1df92"), "name" : "Mongo Guide", "author" : "kh" }
{ "_id" : ObjectId("5d2723e22c00dbef88e1df93"), "name" : "Book1", "author" : "kh" }
{ "_id" : ObjectId("5d2723e22c00dbef88e1df94"), "name" : "Book2", "author" : "kh" }
> db.books.remove({"name": "Book1"})
WriteResult({"nRemoved" : 1 })
> db.books.find()
{ "_id" : ObjectId("5d2723b92c00dbef88e1df92"), "name" : "Mongo Guide", "author" : "kh" }
{ "_id" : ObjectId("5d2723e22c00dbef88e1df94"), "name" : "Book2", "author" : "kh" }
```

Document Query(조회)

20

- 반환(return) 값: *cursor*
- criteria에 해당하는 Document들을 선택하여 cursor를 반환합니다. cursor 는 query 요청의 결과값을 가리키는 pointer 입니다. cursor 객체를 통하여 보이는 데이터의 수를 제한 할 수 있고, 데이터를 sort 할 수 도 있습니다.


parameter	Type	설명
query	document	Optional(선택적). 다큐먼트를 조회할 때 기준을 정합니다. 기준이 없이 컬렉션에 있는 모든 다큐먼트를 조회 할때는 이 매개변수를 비우거나 비어있는 다큐먼트{}를 전달하세요.
projection	document	Optional. 다큐먼트를 조회할 때 보여질 field를 정합니다.

테스트용 목업

21

```
□ db.articles.insert([
□   {
□     "title": "article01",
□     "content": "content01",
□     "writer": "kh1",
□     "likes": 0,
□     "comments": []
□   },
□   {
□     "title": "article02",
□     "content": "content02",
□     "writer": "kh2",
□     "likes": 23,
□     "comments": [
□       {
□         "name": "Bravo",
□         "message": "Hey Man!"
□       }
□     ]
□   },
□ ]
□ }
```

```
□ {  
□   "title": "article03",  
□   "content": "content03",  
□   "writer": "kh3",  
□   "likes": 40,  
□   "comments": [  
□     {  
□       "name": "Charlie",  
□       "message": "Hey Man!"  
□     },  
□     {  
□       "name": "Delta",  
□       "message": "Hey Man!"  
□     }  
□   ]  
□ }  
□ ])
```

- 모든 document 조회:
- `db. COLLECTION_NAME.find()`
- 모든 document 세로 조회:
- `db. COLLECTION_NAME.find().pretty()` 

```

> db.articles.find().pretty()
{
  "_id" : ObjectId("5d2726632c00dbef88e1df95"),
  "title" : "article01",
  "content" : "content01",
  "writer" : "kh1",
  "likes" : 0,
  "comments" : [ ]
}

{
  "_id" : ObjectId("5d2726632c00dbef88e1df96"),
  "title" : "article02",
  "content" : "content02",
  "writer" : "kh2",
  "likes" : 23,
  "comments" : [
    {
      "name" : "Bravo",
      "message" : "Hey Man!"
    }
  ]
}

{
  "_id" : ObjectId("5d2726632c00dbef88e1df97"),
  "title" : "article03",
  "content" : "content03",
  "writer" : "kh3",
  "likes" : 40,
  "comments" : [
    {
      "name" : "Charlie",
      "message" : "Hey Man!"
    },
    {
      "name" : "Delta",
      "message" : "Hey Man!"
    }
  ]
}

```


예제

25

- 1. writer 값이 "kh1" 인 Document 조회
- `db.articles.find({ "writer": "kh1" }).pretty()`
- 2. likes 값이 30 이하인 Document 조회
- `db.articles.find({"likes": {$lte: 30}}).pretty()`

비교(Comparison) 연산자

operator	설명
\$eq	(equals) 주어진 값과 일치하는 값
\$gt	(greater than) 주어진 값보다 큰 값
\$gte	(greather than or equals) 주어진 값보다 크거나 같은 값
\$lt	(less than) 주어진 값보다 작은 값
\$lte	(less than or equals) 주어진 값보다 작거나 같은 값
\$ne	(not equal) 주어진 값과 일치하지 않는 값
\$in	주어진 배열 안에 속하는 값
\$nin	주어진 배열 안에 속하지 않는 값

```
> db.articles.find({"writer": "kh1"}).pretty()
{
  "_id" : ObjectId("5d27e56577d4134997e020d0"),
  "title" : "article01",
  "content" : "content01",
  "writer" : "kh1",
  "likes" : 0,
  "comments" : [ ]
}

> db.articles.find({"likes": {$lte: 30}}).pretty()
{
  "_id" : ObjectId("5d27e56577d4134997e020d0"),
  "title" : "article01",
  "content" : "content01",
  "writer" : "kh1",
  "likes" : 0,
  "comments" : [ ]
}

{
  "_id" : ObjectId("5d27e56577d4134997e020d1"),
  "title" : "article02",
  "content" : "content02",
  "writer" : "kh2",
  "likes" : 23,
  "comments" : [
    {
      "name" : "Bravo",
      "message" : "Hey Man!"
    }
  ]
}
```

- 3. likes 값이 10보다 크고 30보다 작은 Document 조회
- `db.articles.find({ "likes": { $gt: 10, $lt: 30 } }).pretty()`
- 논리연산자

operator	설명
\$or	주어진 조건중 하나라도 true 일 때 true
\$and	주어진 모든 조건이 true 일 때 true
\$not	주어진 조건이 false 일 때 true
\$nor	주어진 모든 조건이 false 일때 true

- 4. writer 값이 "KH1" 이고 likes 값이 10 미만인 Document 조회
- `db.articles.find({ $and: [{ "writer": "kh1" }, { "likes": { $lt: 10 } }] })`

\$where 연산자, \$elemMatch 연산자

28

- \$where 연산자를 통하여 javascript expression 을 사용할 수 있습니다.
- 예)
- comments field 가 비어있는 Document 조회:
- `>db.articles.find({ $where: "this.comments.length == 0" })`
- \$elemMatch 연산자는 Embedded Documents 배열을 쿼리할때 사용됩니다. 저희 mock-up data 에서는 comments 가 Embedded Document에 속합니다.
- comments 중 "Charlie" 가 작성한 댓글이 있는 Document 조회
- `> db.articles.find({ "comments": { $elemMatch: { "name": "Charlie" } } })`

```

> db.articles.find( { "likes": { $gt: 10, $lt: 30 } } ).pretty()
{
  "_id" : ObjectId("5d27e56577d4134997e020d1"),
  "title" : "article02",
  "content" : "content02",
  "writer" : "kh2",
  "likes" : 23,
  "comments" : [
    {
      "name" : "Bravo",
      "message" : "Hey Man!"
    }
  ]
}
> db.articles.find( { $and: [ { "writer": "kh1" }, { "likes": { $lt: 10 } } ] } )
{ "_id" : ObjectId("5d27e56577d4134997e020d0"), "title" : "article01", "content" : "content01", "writer" : "kh1", "likes" : 0, "comments" : [ ] }
> db.articles.find( { $where: "this.comments.length == 0" } ).pretty()
{
  "_id" : ObjectId("5d27e56577d4134997e020d0"),
  "title" : "article01",
  "content" : "content01",
  "writer" : "kh1",
  "likes" : 0,
  "comments" : [ ]
}
> db.articles.find( { "comments": { $elemMatch: { "name": "Charlie" } } } ).pretty()
{
  "_id" : ObjectId("5d27e56577d4134997e020d2"),
  "title" : "article03",
  "content" : "content03",
  "writer" : "kh3",
  "likes" : 40,
  "comments" : [
    {
      "name" : "Charlie",
      "message" : "Hey Man!"
    },
    {
      "name" : "Delta",
      "message" : "Hey Man!"
    }
  ]
}

```

projection?, \$slice 연산자

30

□ Projection

- find() 메소드의 두번째 parameter
- 쿼리의 결과값에서 보여질 field를 정한다.
- 예) article의 title과 content 만 조회

- ```
> db.articles.find({ } , { "_id": false, "title": true, "content": true })
```

## □ \$slice 연산자

- Embedded Document 배열을 읽을때 limit 설정
- title 값이 article03 인 Document 에서 댓글은 하나만 보이게 출력
- ```
> db.articles.find({"title": "article03"}, {comments: {$slice: 1}}).pretty()
```

```
> db.articles.find( { } , { "_id": false, "title": true, "content": true } )
{
  "title" : "article01", "content" : "content01" }
{
  "title" : "article02", "content" : "content02" }
{
  "title" : "article03", "content" : "content03" }
> db.articles.find({"title": "article03"}, {comments: {$slice: 1}}).pretty()
{
  "_id" : ObjectId("5d27e56577d4134997e020d2"),
  "title" : "article03",
  "content" : "content03",
  "writer" : "kh3",
  "likes" : 40,
  "comments" : [
    {
      "name" : "Charlie",
      "message" : "Hey Man!"
    }
  ]
}
```

Sort(),limit(),skip()

32

- find() 메소드를 사용하면 criteria 에 일치하는 모든 document 들을 출력해주기 때문에, 예를들어 페이지 같은 기능을 사용한다면 부적합. 그렇다고 find() 메소드 자체에 어디부터 어디까지 불러오겠다 라고 설정하는 매개변수는 따로 없습니다.
- find() 메소드를 사용했을 시 cursor 형태의 결과값을 반환한다, 이 객체가 가지고 있는 limit() 메소드와 skip() 메소드를 통하여 보이는 출력물의 갯수를 제한 할 수 있고, sort()메소드를 사용하여 데이터를 순서대로 나열 할 수 있습니다.

sampledata

33

```
db.orders.insert(  
  
[  
  { "_id": 1, "item": { "category": "cake", "type": "chiffon" }, "amount":  
10 },  
  { "_id": 2, "item": { "category": "cookies", "type": "chocolate chip" },  
"amount": 50 },  
  { "_id": 3, "item": { "category": "cookies", "type": "chocolate chip" },  
"amount": 15 },  
  { "_id": 4, "item": { "category": "cake", "type": "lemon" }, "amount":  
30 },  
  { "_id": 5, "item": { "category": "cake", "type": "carrot" }, "amount":  
20 },  
  { "_id": 6, "item": { "category": "brownies", "type": "blondie" },  
"amount": 10 }  
])
```

cursor.sort(DOCUMENT)

34

- { KEY: value }
- KEY 는 데이터의 field 이름이고, value 의 값은 1 혹은 -1 입니다. 이 값을 1로 설정하면 오름차순으로, -1로 하면 내림차순으로 정렬합니다.
- _id 의 값을 사용하여 오름차순으로 정렬하기
- >db.orders.find().sort({ "_id": 1 })

- amount 값을 사용하여 오름차순으로 정렬하고, 정렬한 값에서 id 값은 내림차순으로 정렬하기
- > db.orders.find().sort({ "amount": 1, "_id": -1 })

- ***cursor.limit(value)***
- 데이터 갯수를 제한할 때 사용됩니다. value 파라미터는 출력 할 갯수 값 입니다.
- 출력 할 갯수를 3개로 제한하기
- >db.orders.find().limit(3)

- ***cursor.skip(value)***
- 이 메소드는 출력 할 데이터의 시작부분을 설정할 때 사용됩니다. value 값 갯수의 데이터를 생략하고 그 다음 부터 출력합니다.
- 2개의 데이터를 생략하고 그 다음부터 출력
- >db.orders.find().skip(2)

```

> db.orders.find().sort( { "_id": 1 } )
{ "_id" : 1, "item" : { "category" : "cake", "type" : "chiffon" }, "amount" : 10 }
{ "_id" : 2, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 50 }
{ "_id" : 3, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 15 }
{ "_id" : 4, "item" : { "category" : "cake", "type" : "lemon" }, "amount" : 30 }
{ "_id" : 5, "item" : { "category" : "cake", "type" : "carrot" }, "amount" : 20 }
{ "_id" : 6, "item" : { "category" : "brownies", "type" : "blondie" }, "amount" : 10 }
> db.orders.find().sort( { "amount": 1, "_id": -1 } )
{ "_id" : 6, "item" : { "category" : "brownies", "type" : "blondie" }, "amount" : 10 }
{ "_id" : 1, "item" : { "category" : "cake", "type" : "chiffon" }, "amount" : 10 }
{ "_id" : 3, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 15 }
{ "_id" : 5, "item" : { "category" : "cake", "type" : "carrot" }, "amount" : 20 }
{ "_id" : 4, "item" : { "category" : "cake", "type" : "lemon" }, "amount" : 30 }
{ "_id" : 2, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 50 }
> db.orders.find().limit(3)
{ "_id" : 1, "item" : { "category" : "cake", "type" : "chiffon" }, "amount" : 10 }
{ "_id" : 2, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 50 }
{ "_id" : 3, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 15 }
> db.orders.find().skip(2)
{ "_id" : 3, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 15 }
{ "_id" : 4, "item" : { "category" : "cake", "type" : "lemon" }, "amount" : 30 }
{ "_id" : 5, "item" : { "category" : "cake", "type" : "carrot" }, "amount" : 20 }
{ "_id" : 6, "item" : { "category" : "brownies", "type" : "blondie" }, "amount" : 10 }

```

Document Update()

37

- MongoDB에서는 update() 메소드를 통하여 데이터를 수정 할 수 있습니다. 이 메소드의 구조는 다음과 같습니다

```
db.collection.update(  
  <query>,  
  <update>,  
  {  
    upsert: <boolean>,  
    multi: <boolean>,  
    writeConcern: <document>  
  }  
)
```

- Collection 안의 document(들)을 수정합니다. 이 메소드를 통하여 특정 field 를 수정 할 수도 있고 이미 존재하는 document를 대체(replace) 할 수도 있습니다.
- update() 메소드의 기본 옵션으로는 **단 하나**의 document를 수정합니다.

Parameter	Type	설명
*query	document	업데이트 할 document의 criteria 를 정합니다. find() 메소드 에서 사용하는 query 와 같습니다.
*update	document	document에 적용할 변동사항입니다.
upsert	boolean	Optional. (기본값: false) 이 값이 true 로 설정되면 query한 document가 없을 경우, 새로운 document를 추가합니다.
multi	boolean	Optional. (기본값: false) 이 값이 true 로 설정되면, 여러개의 document 를 수정합니다.
writeConcern	document	Optional. wtimeout 등 document 업데이트 할 때 필요한 설정값입니다. 기본 writeConcern을 사용하려면 이 파라미터를 생략하세요. 자세한 내용은 매뉴얼 을 참조해주세요.

Sample 데이터 추가

39

- ❑ db.people.insert([
 - ❑ { name: "Abet", age: 19 },
 - ❑ { name: "Betty", age: 20 },
 - ❑ { name: "Charlie", age: 23, skills: ["mongodb", "nodejs"] },
 - ❑ { name: "David", age: 23, score: 20 }]
- ❑])

- 예제1. 특정 field 업데이트 하기
- 특정 field의 값을 수정할 땐 \$set 연산자
- #Abet document 의 age를 20으로 변경한다
- `>db.people.update({ name: "Abet" }, { $set: { age: 20 } })`
- 예제2. document를 replace 하기
- #Betty document를 새로운 document로 대체한다.
- `>db.people.update({ name: "Betty" }, { "name": "Betty 2nd", age: 1 })`
- 예제3. 특정 field를 제거하기 1=true
- #David document의 score field를 제거한다
- `>db.people.update({ name: "David" }, { $unset: { score: 1 } })`
- 예제4. criteria에 해당되는 document가 존재하지 않는다면 새로 추가 하기
- #upsert 옵션을 설정하여 Elly document가 존재하지 않으면 새로 추가
- `>db.people.update({ name: "Elly" }, { name: "Elly", age: 17 }, { upsert: true })`




```

> db.people.update( { name: "Abet" }, { $set: { age: 20 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find().pretty()
{
  "_id" : ObjectId("5d27ec8e77d4134997e020d3"),
  "name" : "Abet",
  "age" : 20
}
{
  "_id" : ObjectId("5d27ec8e77d4134997e020d4"),
  "name" : "Betty",
  "age" : 20
}
{
  "_id" : ObjectId("5d27ec8e77d4134997e020d5"),
  "name" : "Charlie",
  "age" : 23,
  "skills" : [
    "mongodb",
    "nodejs"
  ]
}
{
  "_id" : ObjectId("5d27ec8e77d4134997e020d6"),
  "name" : "David",
  "age" : 23,
  "score" : 20
}
> db.people.update( { name: "Betty" }, { "name": "Betty 2nd", age: 1 })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.update( { name: "David" }, { $unset: { score: 1 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.update( { name: "Elly" }, { name: "Elly", age: 17 }, { upsert: true } )
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("5d281b72c11b60383d646be9")
})

```

- 예제5. 여러 document의 특정 field를 수정하기
- #age가 20 보다 낮거나 같은 document의 score를 10으로 설정
- >db.people.update(
 - ... { age: { \$lte: 20 } },
 - ... { \$set: { score: 10 } },
 - ... { multi: true }
 - ...)
- 예제6-1. 배열 에 값 추가하기
- #Charlie document의 skills 배열에 "angularjs" 추가
- db.people.update(
 - ... { name: "Charlie" },
 - ... { \$push: { skills: "angularjs" } }
 - ...)

```

> db.people.update(
...   { age: { $lte: 20 } },
...   { $set: { score: 10 } },
...   { multi: true }
... )
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
> db.people.update(
...   { age: { $lte: 20 } },
...   { $set: { score: 10 } },
...   { multi: true }
... )
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 0 })
> db.people.update(
...   { name: "Charlie" },
...   { $push: { skills: "angularjs" } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.update(
...   { name: "Charlie" },
...   { $push: {
...     skills: {
...       $each: [ "c++", "java" ],
...       $sort: 1
...     }
...   }
...   }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

```

- 예제 7-1. 배열에 값 제거하기
- Charlie document에서 skills 값의 mongodb 제거
- >db.people.update(
□ ... { name: "Charlie" },
□ ... { \$pull: { skills: "mongodb" } }
□ ...)
- 예제 7-2. 배열에서 값 여러개 제거하기
- Charlie document에서 skills 배열 중 "angularjs" 와 "java" 제거
- db.people.update(
□ ... { name: "Charlie" },
□ ... { \$pull: { skills: { \$in: ["angularjs", "java"] } } }
□ ...)

- \$sort 값을 내림차순으로 정렬하려면 -1 로 하면 됩니다.
- 배열이 document의 배열이고 그 embedded document의 특정 field에 따라서 정렬을 할 때는 다음과 같이 설정하면 됩니다.

```
> db.people.update(
...   { name: "Charlie" },
...   { $pull: { skills: { $in: ["angularjs", "java" ] } } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.update(
...   { name: "Charlie" },
...   { $pull: { skills: "mongodb" } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Index란?

46

- Index는 MongoDB에서 데이터 쿼리를 더욱 효율적으로 할 수 있게 해줍니다. 인덱스가 없이는, MongoDB는 collection scan – 컬렉션의 데이터를 하나하나 조회 – 방식으로 스캔을 하게 됩니다. 만약 document의 갯수가 매우 많다면, 많은 만큼 속도가 느려지겠죠? 이 부분을 향상시키기 위하여 인덱스를 사용하면 더 적은 횟수의 조회로 원하는 데이터를 찾을 수 있습니다.

Index의 종류

47

- **기본 인덱스 _id**
- 모든 MongoDB의 컬렉션은 기본적으로 _id 필드에 인덱스가 존재합니다. 만약에 컬렉션을 만들 때 _id 필드를 따로 지정하지 않으면 mongod 드라이버가 자동으로 _id 필드 값을 ObjectId로 설정해줍니다.
- _id 인덱스는 unique(유일)하고 이는 MongoDB 클라이언트가 같은 _id를 가진 문서를 중복적으로 추가하는 것을 방지합니다.

- **Single(단일) 필드 인덱스**
- MongoDB 드라이버가 지정하는 _id 인덱스 외에도, 사용자가 지정 할 수 있는 단일 필드 인덱스가 있습니다.

- **Compound (복합) 필드 인덱스**
- 두개 이상의 필드를 사용하는 인덱스를 복합 인덱스라고 부릅니다. 다음 이미지와 같이 첫번째 필드 (userid)는 오름차순으로, 두번째 필드 (score)는 내림차순으로 정렬 해야 하는 상황이 있을때 사용합니다.

- **Multikey 인덱스**
- 필드 타입이 배열인 필드에 인덱스를 적용 할 때는 Multikey 인덱스가 사용됩니다. 이 인덱스를 통하여 배열에 특정 값이 포함되어 있는 document를 효율적으로 스캔 할 수 있습니다.
- **Geospatial(공간적) Index**
- 지도의 좌표와 같은 데이터를 효율적으로 쿼리하기 위해서 (예: 특정 좌표 반경 x 에 해당되는 데이터를 찾을 때) 사용되는 인덱스입니다. 자세한 사항은 매뉴얼을 참고해주세요.
- **Text 인덱스**
- 텍스트 관련 데이터를 효율적으로 쿼리하기 위한 인덱스입니다.
- **해쉬 (hashed) 인덱스**
- 이 인덱스를 사용하면 B Tree가 아닌 Hash 자료구조를 사용합니다. Hash는 검색 효율이 B Tree보다 좋지만, 정렬을 하지 않습니다.

인덱스 생성

49

- 인덱스를 생성 할 땐, 다음과 같은 `createIndex()` 메소드를 사용합니다. 파라미터는 인덱스를 적용할 필드를 전달합니다. 값을 1로하면 오름차순으로, -1로 하면 내림차순으로 정렬합니다.
- `db.COLLECTION.createIndex({ KEY: 1 })`
- `db.report.find({ score: 57 })`
- `db.report.createIndex({ age: 1, score: -1})`

인덱스 속성

50

- 인덱스에 속성을 추가 할 땐 `createIndex()` 메소드의 두번째 인자에 속성값을 `document` 타입으로 전달해주면 됩니다.
- `db.COLLECTION.createIndex({ KEY: 1 }, { PROPERTY: true })`
- **Unique (유일함) 속성**
- `id` 필드처럼 컬렉션에 단 한개의 값만 존재 할 수 있는 속성입니다.
- `db.userinfo.createIndex({ email: 1 }, { unique: true })`
- **Partial (부분적) 속성**
- 예제5 visitors 값이 1000 보다 높은 document에만 name 필드에 인덱스 적용
- `db.store.createIndex(`
- `{ name: 1 },`
- `{ partialFilterExpression: { visitors: { $gt: 1000 } } }`
- `)`

인덱스 조회, 제거

51

- 생성된 인덱스를 조회 할 땐 `getIndexes()` 메소드를 사용합니다.
- `db.COLLECTION.getIndexes()`
- 인덱스를 제거 할 땐 `dropIndex()` 메소드를 사용합니다.
`db.COLLECTION.dropIndex({ KEY: 1 })`

Mongoose

52

- Mongoose는 MongoDB 기반 ODM(Object Data Mapping) Node.JS 전용 라이브러리입니다. ODM은 데이터베이스와 객체지향 프로그래밍 언어 사이 호환되지 않는 데이터를 변환하는 프로그래밍 기법입니다. 즉 MongoDB 에 있는 데이터를 여러분의 Application에서 JavaScript 객체로 사용 할 수 있도록 해줍니다.

CRUD 명령어 (생성, 읽기, 변경, 삭제)

53

□ Create

save 명령

```
// save one user  
db.users.save({ name: 'Chris' });  
  
// save multiple users  
db.users.save([ { name: 'Chris'}, { name: 'Holly' } ]);
```

□ Read

find 명령

```
// show all users  
db.users.find();  
  
// find a specific user  
db.users.find({ name: 'Holly' });
```

□ Update

update 명령

```
db.users.update({ name: 'Holly' }, { name: 'Holly Lloyd' });
```

□ Delete

remove 명령

```
// remove all  
db.users.remove({});  
  
// remove one  
db.users.remove({ name: 'Holly' });
```

SQL: 테이블(table)과 레코드(record)
NoSQL: 컬렉션(collection)과 문서(document)

- Save 와 insert의 차이점
- Save: _id가 동일해도 기존데이터에 덮어씌워저장
- Insert: _id가 동일한값을 저장하려하면 오류발생