MongoDb 기초 #1

**목차**

[1. 배경지식 2](#_Toc508029775)

[1) MongoDb의 장점 2](#_Toc508029776)

[2. DB 3](#_Toc508029777)

[1) db생성 3](#_Toc508029778)

[2) db 제거 3](#_Toc508029779)

[3. Collection 8](#_Toc508029782)

[1) Collection 생성 8](#_Toc508029783)

[2) Collection 제거 10](#_Toc508029784)

[4. Document 16](#_Toc508029790)

[1) Document 생성 16](#_Toc508029791)

[2) Document 제거 18](#_Toc508029792)

[5. find 20](#_Toc508029793)

[1) find 문법 20](#_Toc508029794)

[2) find 함수의 구조 23](#_Toc508029795)

[3) 연산자 24](#_Toc508029796)

[4) 논리 연산자 24](#_Toc508029796)

[5) 정규식 24](#_Toc508029796)

[6. Cursor 26](#_Toc508029797)

[1) Cusor 문법 26](#_Toc508029798)

[2) Sort 27](#_Toc508029799)

[3) Limit 27](#_Toc508029799)

[4) Skip 27](#_Toc508029799)

[2) ShowPage 27](#_Toc508029799)

[7. Update 26](#_Toc508029797)

[1) Update 문법 26](#_Toc508029798)

[2) Update 함수의 구조 27](#_Toc508029799)

[3) Update 27](#_Toc508029799)

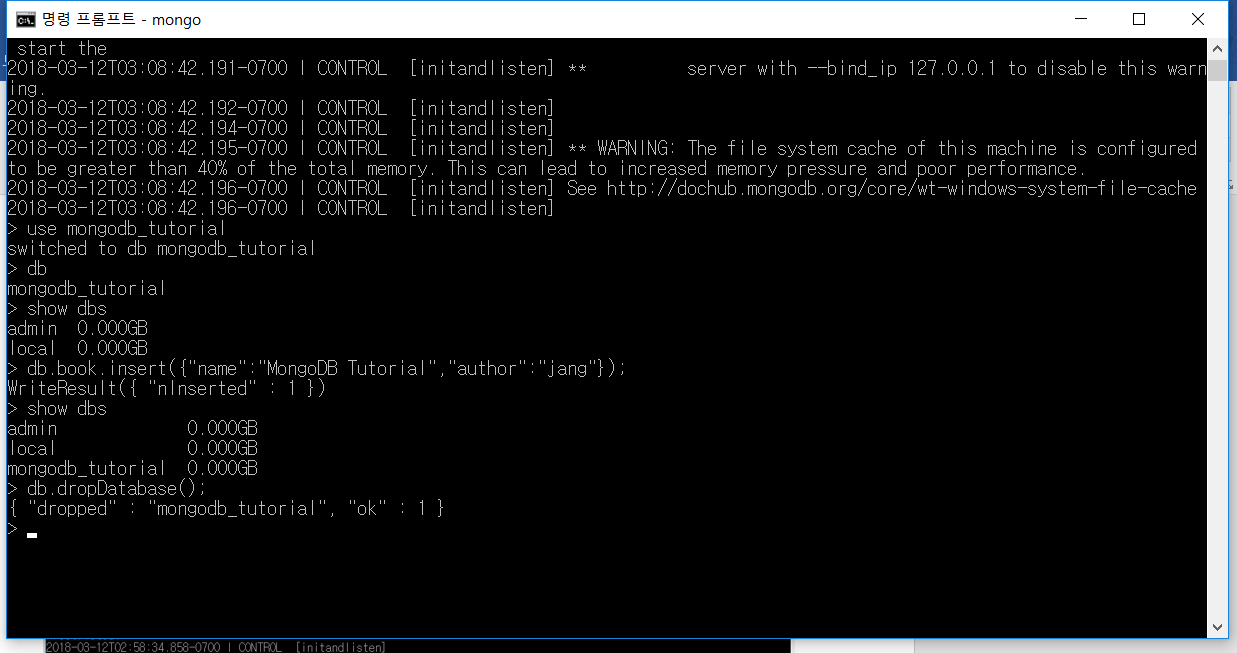
# MongoDb의 장점

MongoDB를 통하여 얻을 수 있는 장점은 NoSQL과 거의 다를 바가 없다.

* Schema-less. 같은 Collection 안에 존재하여도 다른 Schema를 가질 수 있다.(Dynamic Schema)
* 각 객체의 구조가 뚜렷하다.
* 복잡한 Join 연산이 필요 없다.
* Deep Query-Ability, 즉 복잡한 쿼리에 대한 대응이 MongoDB에서 Document 기반의 Query Language를 사용하여 Relational Database에서 사용했던 SQL 문장의 성능을 제공한다.
* 어플리케이션에서 사용하는 객체를 데이터베이스에 추가할 때 데이터에 대한 변환과 매핑 작업이 딱히 필요 없다.

1.DB

# DB 생성



# 명령어

|  |  |
| --- | --- |
| 1 | use mongodb\_tutorial |

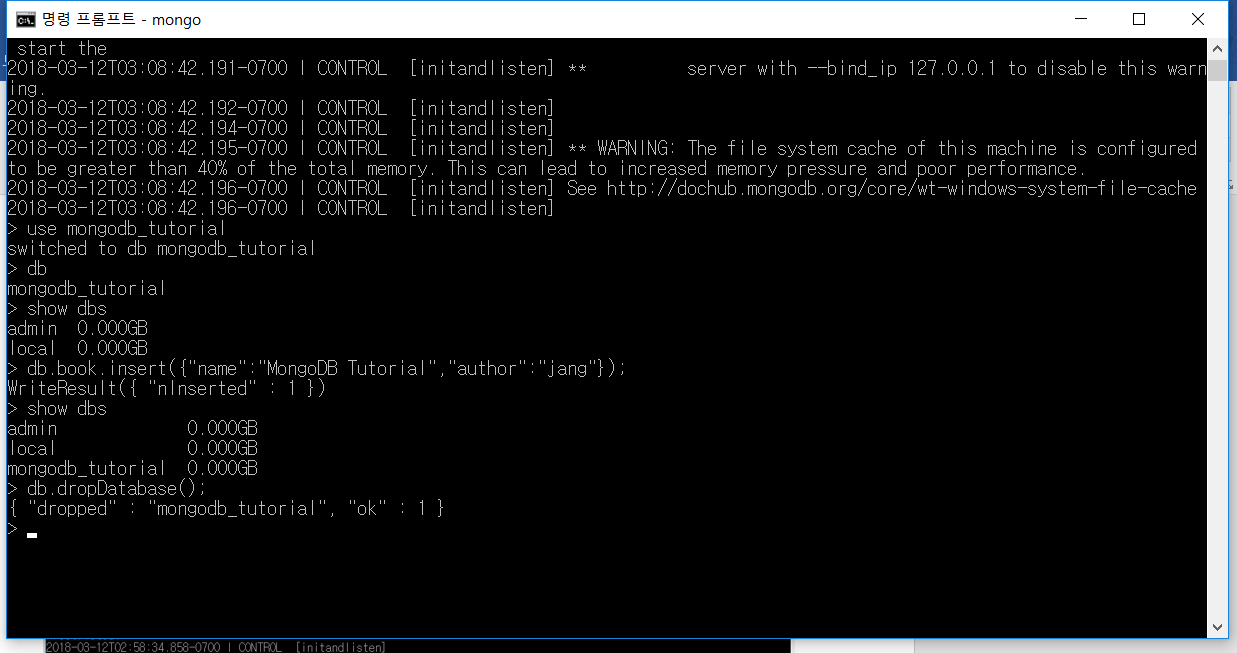
명령어 use mongodb\_tutorial 🡪 mongodb tutorial 이라는 데이터베이스를 생성합니다.

명령어 db 🡪 현재 사용중인 데이터베이스를 확인합니다.

명령어 show dbs 🡪 내가 만든 데이터베이스 리스트 확인 ( 리스트를 확인하려면 최소 한 개의 Document를 추가해야함

현재 MongoDB에 존재하는 데이터베이스의 이름이 mongo\_tutorial이 존재한다면 이를 반환하고 없다면 새롭게 생성을 해 주는 역할로 보면 된다.

# DB 제거



# 명령어

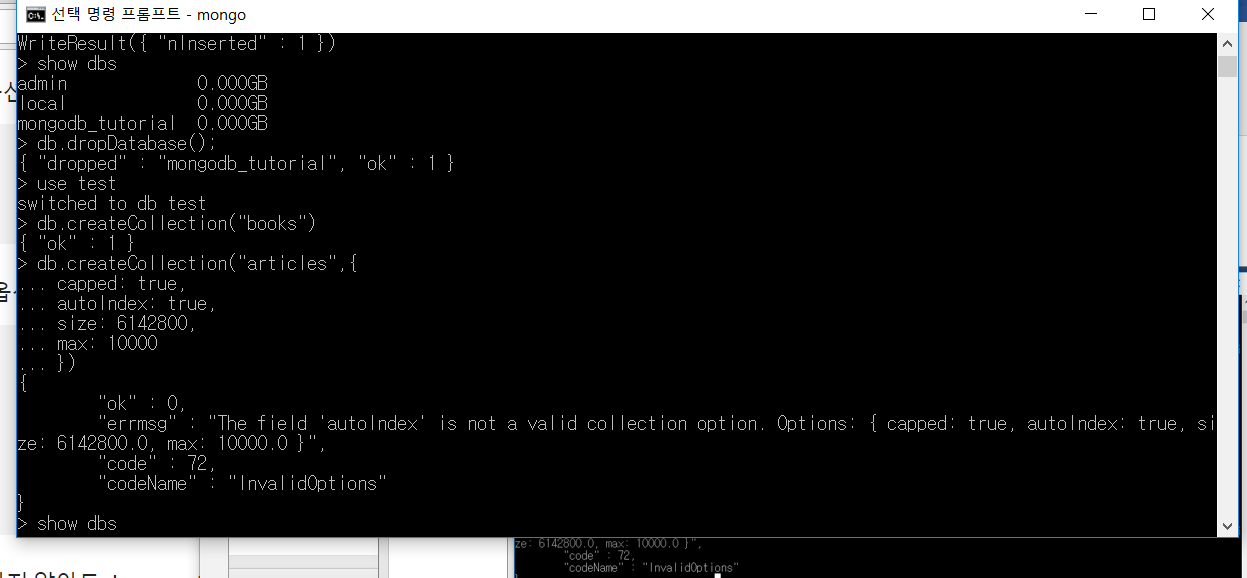
|  |  |
| --- | --- |
| 1 | Db.dropDatabase(); |

명령어 db.dropDatabase() 데이터 베이스 제거 이 명령어를 삭제하기전 use name으로 선택 후 삭제

실제로 db라는 명령어 중에 dropDatabase 명령어는 말 그대로 Database를 영구 삭제를 하는 역할을 해 준다. 이는 Relational Database 명령어 중에 Drop를 활용해서 쓰는 원리와 같다고 볼 수 있는데 그렇지만 db 명령어를 확인해서 삭제를 하는 것을 염두를 하자.

2.Colletion

# Collection 생성



# 명령어

|  |  |
| --- | --- |
| 1 | db.createCollection(“books”) |

Database 내부에서 Collection을 생성하는 경우에는 현재 가리키는 db를 통해서 createCollection 함수를 이용해서 Collection을 생성하면 된다. 우선 방금 전에 새로 만들었던 database인 example01를 통해서 collection을 추가해보도록 하자. 초반에는 use example01를 통해서 연산자 db가 이를 가리키게끔 해야 한다. 기본적인 명령어는 아래의 형식처럼 구성이 되어 있는데 option에 쓰는 값들은 생략 해도 무관은 하다. 그러나 Capped Collection을 통하여 collection에 한계를 주는 경우에는 써 줘야 한다.

Collection을 만드는 3가지 방법들이 있는데

위의 첫번쨰 방법과 default 설정말고 설정을 정해서 생성하는 방법

|  |
| --- |
| > db.createCollection(“music02”, { capped : true, size : 5242880, max : 150 })  [db.createCollection(콜렉션\_이름, {옵션 : 옵션 값 … })] |

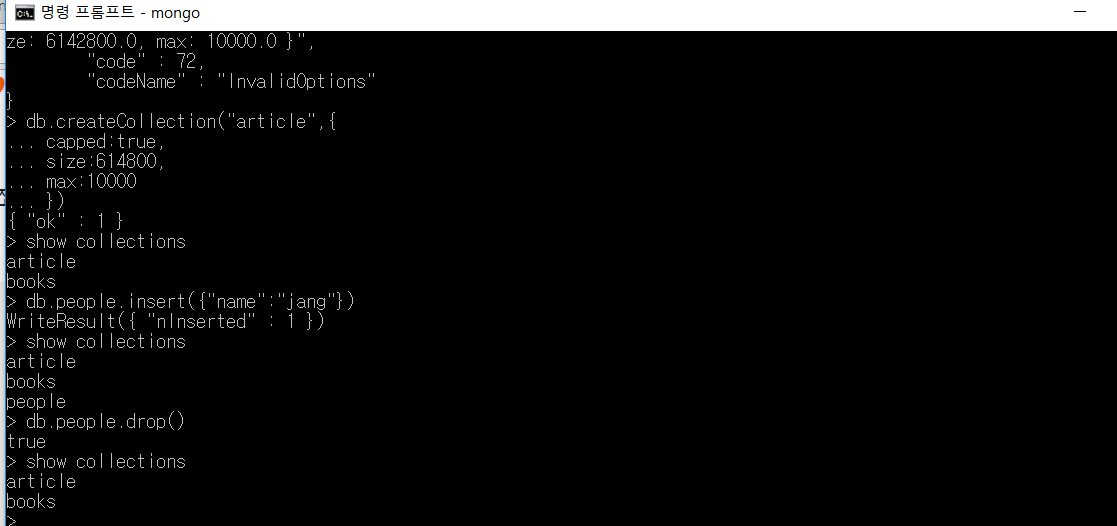
이처럼 작성을 하게 된다면 music02 Collection은 Document의 개수가 150개로 한정이 되며, Collection 내부에 있는 총합 용량의 제한이 5MB가 된다

그 다음 document를 같이 넣으면서 생성하는 방법

|  |  |
| --- | --- |
| 1 | Db.people.insert({“name”:”jang”}) |

이런 방법들이 있다.

# Collection 제거



# 명령어

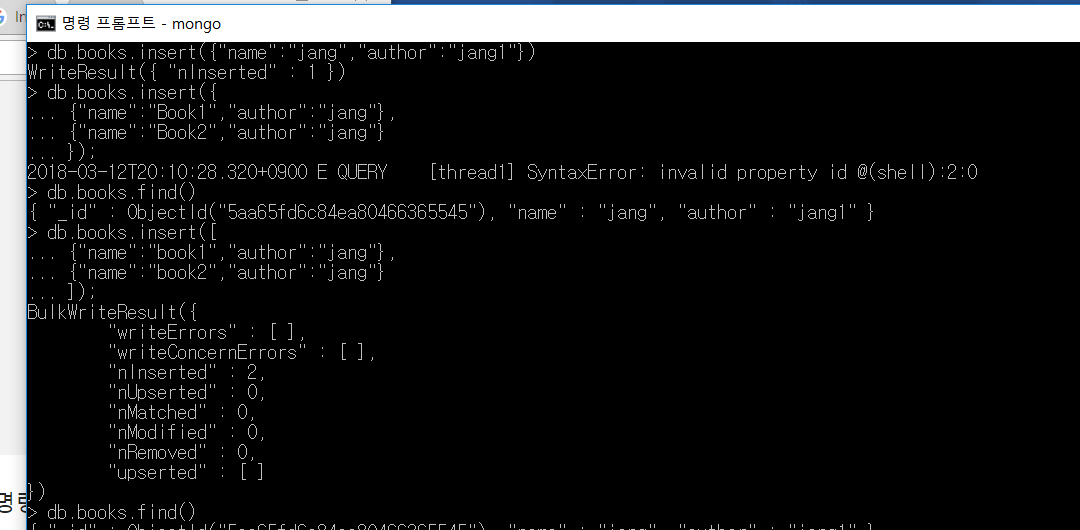
|  |  |
| --- | --- |
| 1 | db.people.drop() |

마지막에 생성한 people collection을 지웠음

방금 삭제된 Collection이 처리가 되었으면 어떤 Collection들이 남아 있는지 여부를 참고할 때 쓰는 show collections로 작성해서 확인하면 된다. 현재 db가 무엇을 가리키는지에 대해 확인을 하고 작성을 한다

3.Document

# Document 생성





# 명령어

|  |  |
| --- | --- |
| 1 | db.books.insert({"name":"jang","author":"jang1"}) |

한 개의 document 추가

# 명령어

|  |  |
| --- | --- |
| 1  2  3  4  5 | db.books.insert({  ... {"name":"Book1","author":"jang"},  ... {"name":"Book2","author":"jang"}  ... }); |

Document 다중 추가

괄호 확인 필수

db.books.find()로 colletion의 document 확인 가능

# Document 제거



# 명령어

|  |  |
| --- | --- |
| 1 | db.books.remove({"name":"book1"}) |

명령어 db.books.remove({"name":"book1"}) 로 name book1의 document를 삭제

Relational Database에서 생각했던 Delete 문과 비슷하게 생각할 수 있는데 삭제 Document 조건은 Delete 문에서 Where 문과 같이 적용할 수 있다고 보면 되고, 1개만 삭제 여부는 삭제 Document 조건에 해당되는 Document 목록들 중에서 가장 오래 된 데이터를 삭제하게 된다. 그러면 Collection 내부에 존재하는 Document들을 모조리 삭제하는 경우에는 어떻게 써야 할까? 바로 삭제 Document 조건에 { } 를 써서 모두를 삭제하는 방법이나 그냥 remove() 함수를 작성해 삭제를 하는 방법이 있다. 이러한 방법이 Relational Database에서 Delete 문에 어느 조건 없이 삭제하는 Truncate와 같은 원리로 상기할 수 있다. 고로 Truncate와 Drop은 서로 다른 개념이다. Truncate는 Table에 현존하는 Record들만 모두 삭제를 하는 개념이고, Drop은 Table와 그 속에 있는 Record를 아예 없애는 개념이다.

4. Find

# Find 문법

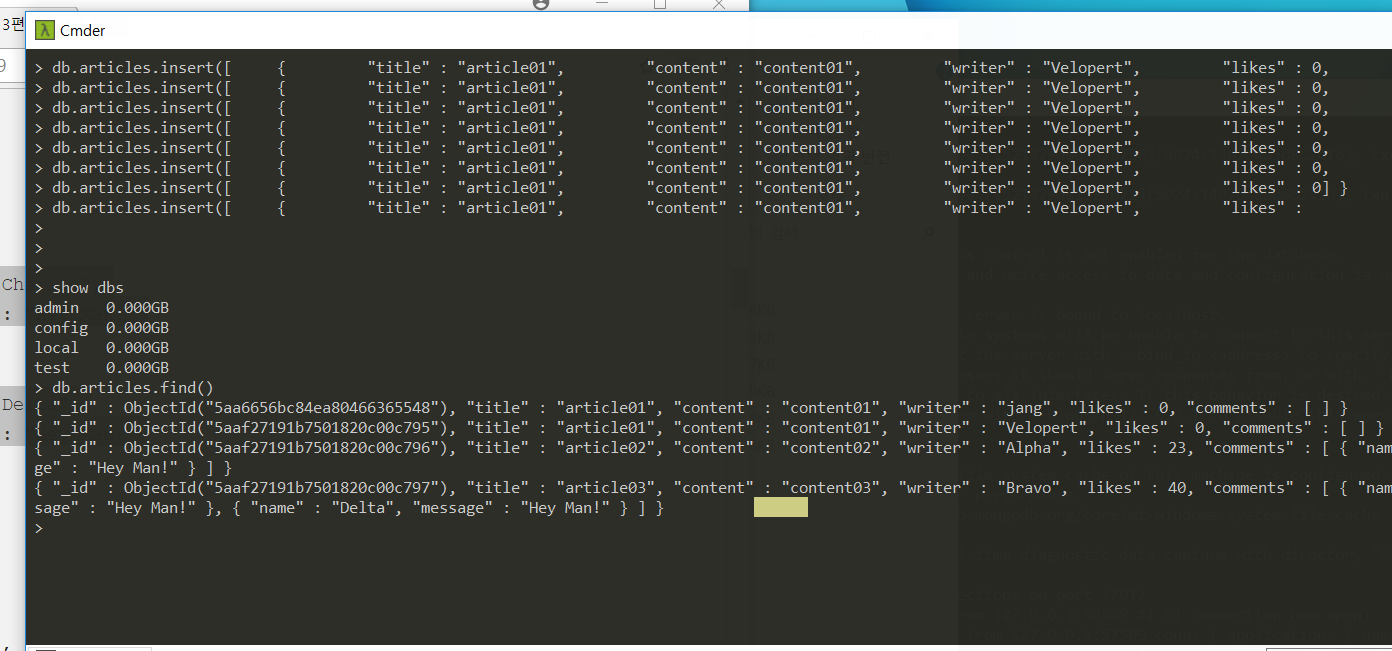
이제 몽고 디비에서 document 조회 기능하고 find 메소드를 알아 볼 것이다 .

# 명령어

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | db.articles.insert([  {  "title" : "article01",  "content" : "content01",  "writer" : "Velopert",  "likes" : 0,  "comments" : [ ]  },  {  "title" : "article02",  "content" : "content02",  "writer" : "Alpha",  "likes" : 23,  "comments" : [  {  "name" : "Bravo",  "message" : "Hey Man!"  }  ]  },  {  "title" : "article03",  "content" : "content03",  "writer" : "Bravo",  "likes" : 40,  "comments" : [  {  "name" : "Charlie",  "message" : "Hey Man!"  },  {  "name" : "Delta",  "message" : "Hey Man!"  }  ]  }  ]) |

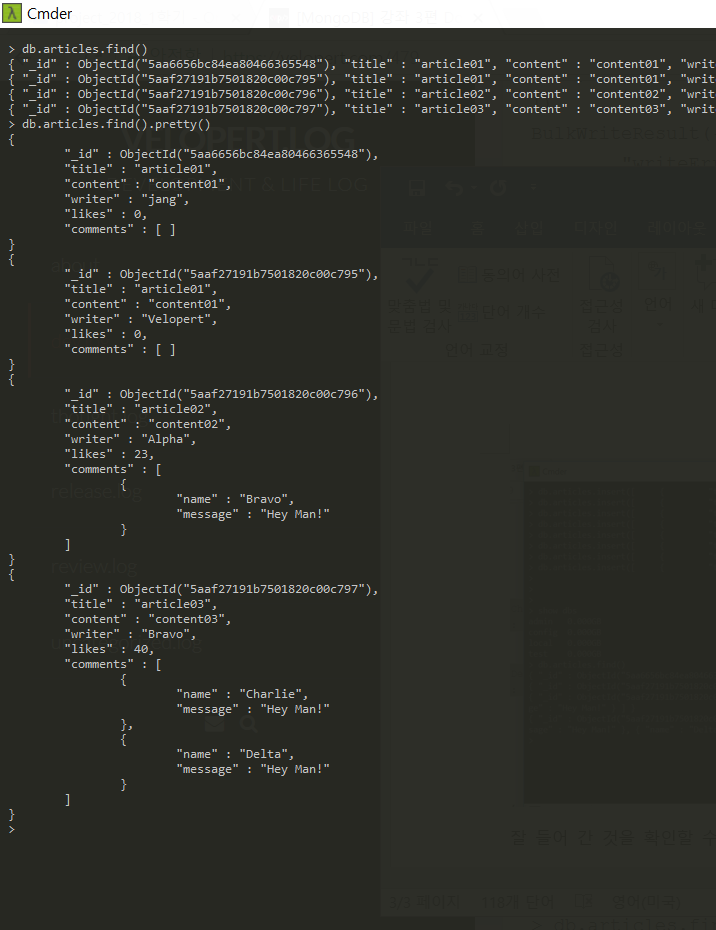
위 명령어로 articles collection에 document들을 삽입했다.

db.articles.find() 명령어로 정보들이 잘 들어갔는지 확인 할 수 있다



# 명령어

|  |  |
| --- | --- |
| 1 | db.articles.find() |



# 명령어

|  |  |
| --- | --- |
| 1 | db.articles.find().pretty() |

.

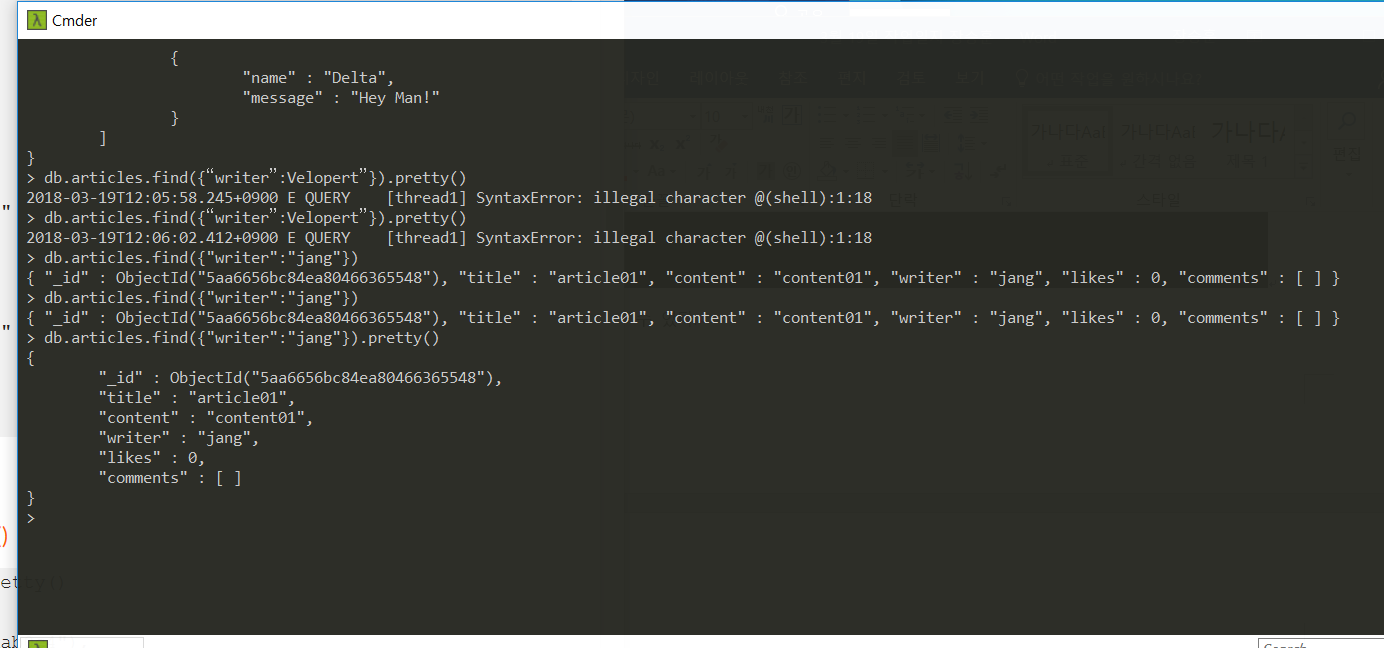
명령어 : db.articles.find().pretty()를 쓰면 조금 더 보기 좋게 볼 수 있다.

# Find 함수의 구조

find 함수는 아래와 같이 구성이 되어 있다.

|  |
| --- |
| > db.music.find({title : “나 항상 그대를”})  [ db.콜렉션\_이름.find({ Query 조건 }, projection(조회할 필드 설정)) ] |

이를 통해서 Document들을 선택을 해서 반환하는 객체를 Cursor로 불린다. 실제로 Relational Database에서 PL/SQL를 공부해 본 경험이 있다면 Cursor 함수를 잠깐 본 적이 있을 것이다. Select 문에서 조건을 지정을 하고 반환된 데이터 목록들을 담아서 PL/SQL 문 내부에서 쓰는 함수가 있었는데 MongoDB에서도 마찬가지인 셈이다. Cursor 객체가 반환이 된다면 추후에 이를 정렬화, 데이터 수의 제약 등을 둘 수 있는데 이에 대해서는 Update 문이 끝나고 구체적으로 들어가보도록 하자.



# 명령어

|  |  |
| --- | --- |
| 1 | db.articles.find({“writer”:jang”}) |

이제 db.articles.find({“writer”:jang”}) 이란 명령어로 작가가 장인 document를 찾을 수 있다.

# 연산자

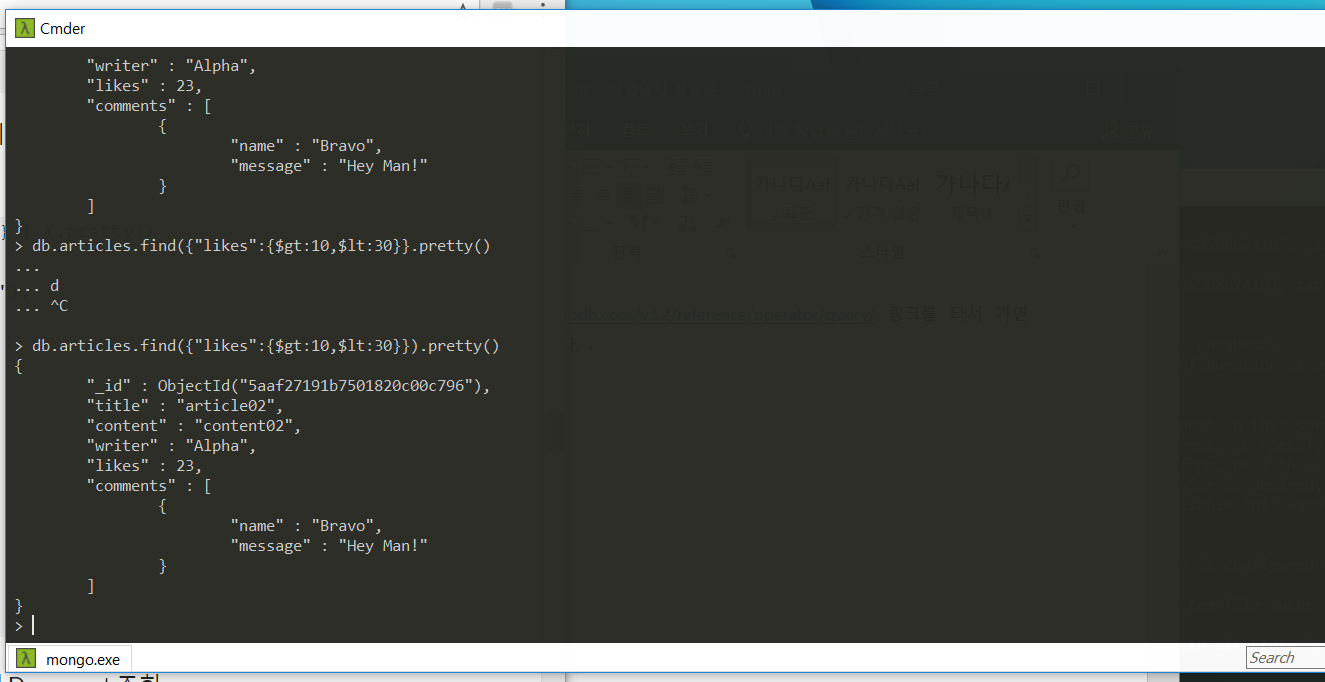
MongoDb도 mySql 처럼 조건에 연산자를 집어 넣을 수 있다.

몽고 디비에서는 연산자를 다르게 쓰는데 밑에 연산자표를 참고해서 조건문을 만들면 된다

|  |  |
| --- | --- |
| **operator** | **설명** |
| $eq | 주어진 데이터와 일치해야 함. 정작 찾을 데이터 옆에 쓰면 되어 극히 무용지물. |
| $gt | 주어진 데이터의 값보다 커야 한다.(초과) |
| $gte | 주어진 데이터의 값 이상이어야 한다. |
| $lt | 주어진 데이터의 값보다 작아야 한다.(미만) |
| $lte | 주어진 데이터의 값 이하이어야 한다. |
| $ne | 주어진 데이터와 달라야 한다. |
| $in | 주어진 배열 안에 속하는 값. RDBMS의 IN과 같다. |
| $nin | 주어빈 배열 안에 속하지 않는 값. RDBMS의 Not IN과 같다. |

더 자세한 정보는 <https://docs.mongodb.com/v3.2/reference/operator/query/> 링크를 타서 가면 몽고 디비 매뉴얼을 참조하길 바랍니다.

이제 likes가 10이상 30이하인 document를 찾아보자



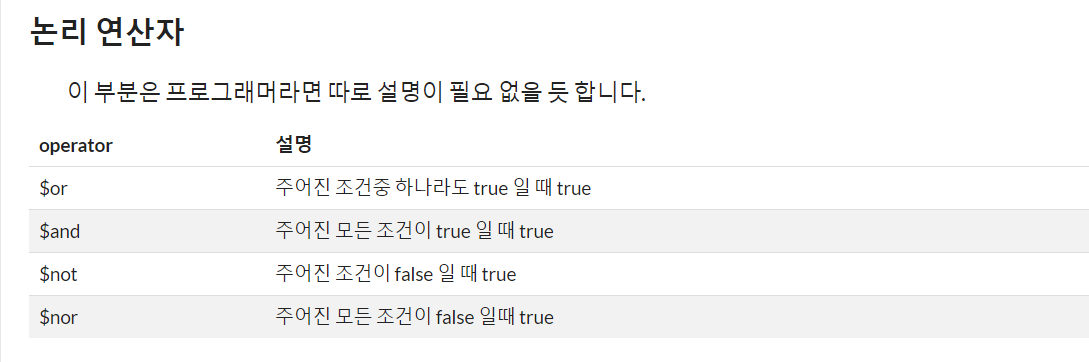
# 명령어

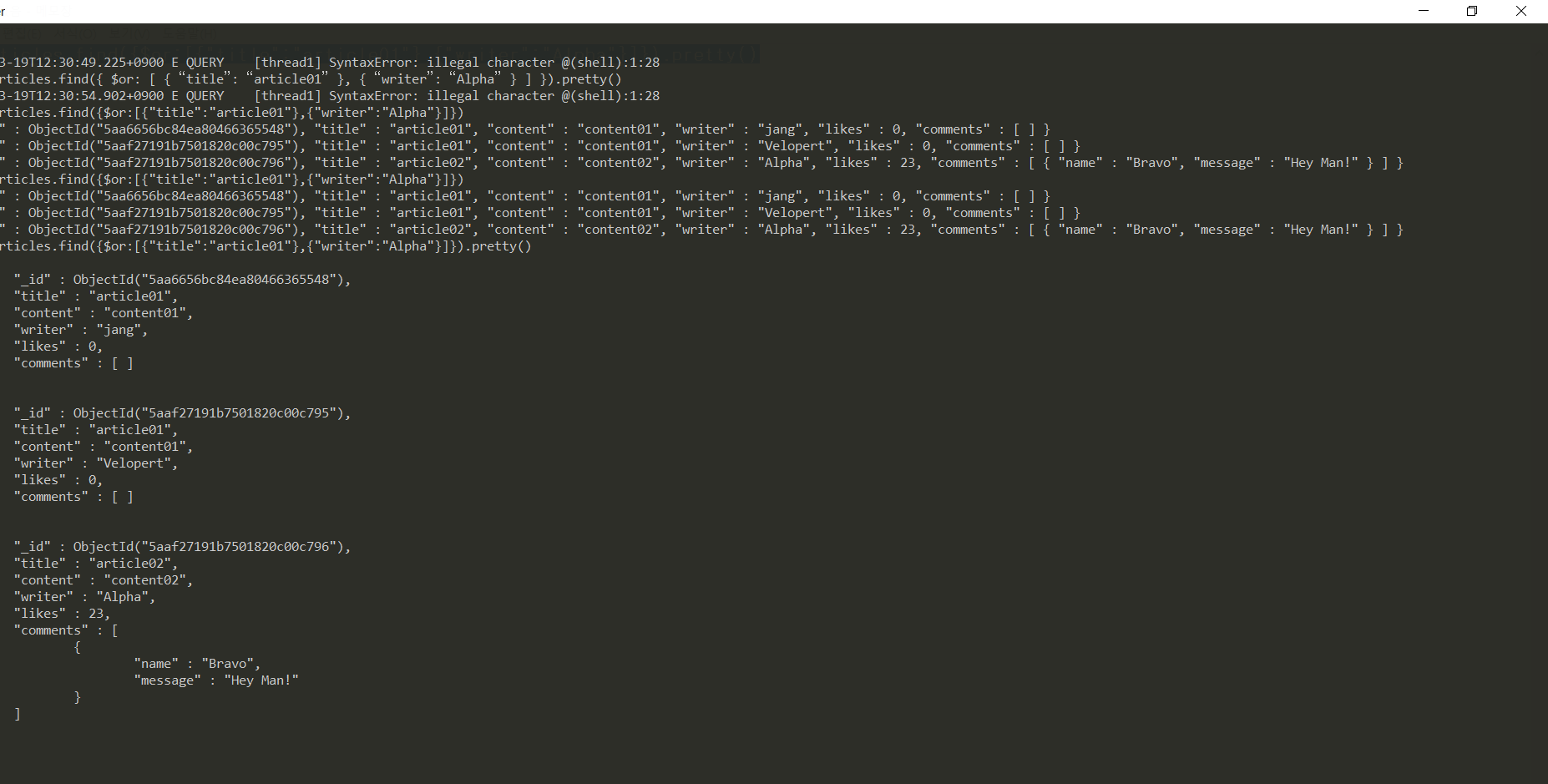
|  |  |
| --- | --- |
| 1 | db.articles.find({“likes”:($gt:10,$lt:30)}).pretty() |

명령어 : db.articles.find({“likes”:($gt:10,$lt:30)}).pretty() 로 likes가 10이상 30이하인 document를 찾을 수 있다.

# 논리연산자

다음은 논리연산자인데 디비와는 다르게 맨 앞에 선언해서 쓸 떄 조금 헷갈린다. 그리고 복사 붙여넣기를 할 경우 메모장에서 복사 붙여넣기를 하기 바란다. 기존 디비처럼 띄어쓰기가 잘 못 되면 에러가 나므로 메모장에서 띄어쓰기도 잘 잡아놓고 붙여넣기를 바랍니다.



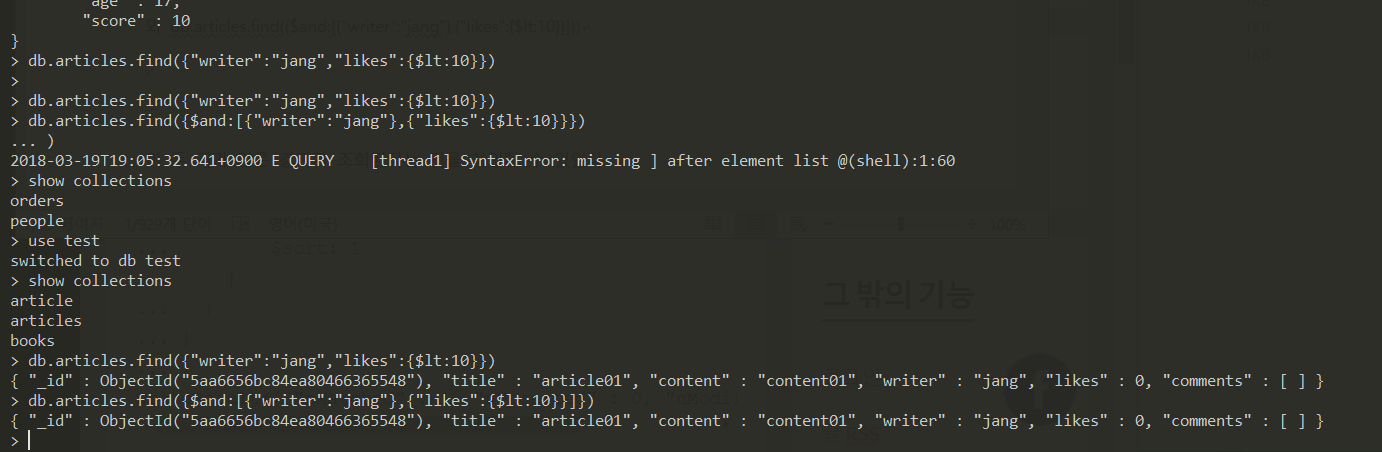


# 명령어

|  |  |
| --- | --- |
| 1 | db.articles.find({$or:[{"title":"article01"},{"writer":"Alpha"}]}).pretty() |

명령어 : db.articles.find({$or:[{"title":"article01"},{"writer":"Alpha"}]}).pretty()

를 입력하면 title이 article01이거나 writer가 Alpha인 document를 찾아준다.



# 명령어

|  |  |  |
| --- | --- | --- |
| 1 | db.articles.find({$and:[{"writer":"jang"},{"likes":{$lt:10}}]}) | db.articles.find().pretty() |

명령어 db.articles.find({"writer":"jang","likes":{$lt:10}})

와 db.articles.find({$and:[{"writer":"jang"},{"likes":{$lt:10}}]})

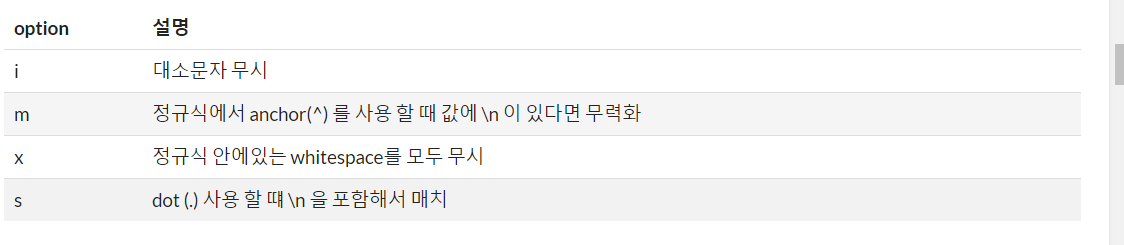
이 두 개가 같은 결과가 조회된다는 것을 확인할 수 있다.

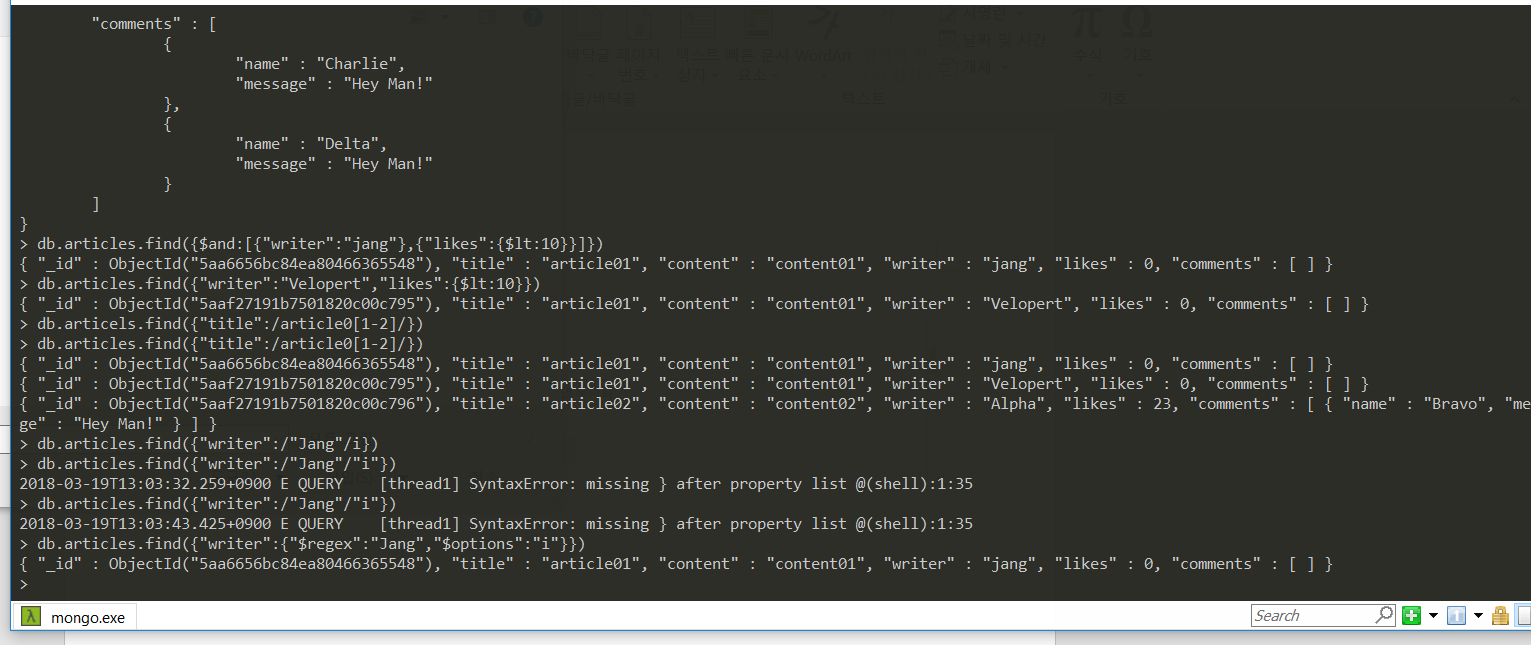
# 정규식

문자열을 찾을 때, 완전히 동일한 문자열만 찾는 것이 아니고, 어떤 패턴과 일치하는 문자열을 전부 다 찾을 수 있으면 편리할 것이다.

찾을 문자열의 패턴을 지정하는 표현식(expression)이 정규식(regular expression) 이다

MongoDb도 정규식을 지원한다.





# 명령어

|  |  |
| --- | --- |
| 1 | db.articles.find({"writer":{"$regex":"Jang","$options":"i"}}) |

정규식 i를 이용해서

명령어 : db.articles.find({"writer":{"$regex":"Jang","$options":"i"}})

입력하면 $regex 라는 연산자에 Jang을 입력해도 옵션 i를 통해 대소문자 구분 없이 조회가 가능하다

# 명령어

|  |  |
| --- | --- |
| 1 | db.articles.find({"title":/article0[1-2]/}) |

명령어 : db.articles.find({"title":/article0[1-2]/})

이렇게 입력하면 title에 article01,02가 다 조회되는 것을 알 수 있다.

5.Cursor

# Cursor 문법

cursor 객체는 find() 함수에 Query와 Projection을 이용해서 조회를 완료한 Document의 목록들을 정리를 한 개념으로 인지할 수 있다. 그렇지만 Document 목록들을 조회할 때에 가장 중요한 Sorting(정렬), Limit(Document 수의 제한), Skip(일부 Document에 대하여 그 index부터 반환을 함)에 대한 조건을 추가할 필요가 있다.(예를 들어 Pagination에서 페이지 별로 조회나 마지막 Document를 찾아서 조회하는 경우 etc.) 이럴 때에 쓰는 문법들이 바로 cursor 문법인데 어떻게 이용이 되는지 알아보도록 하자.

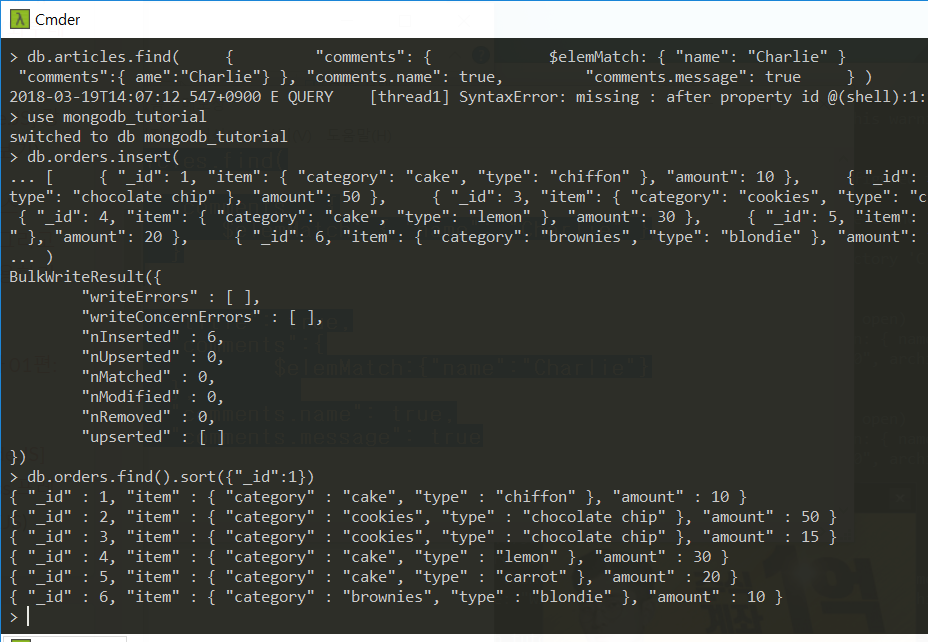
# 명령어

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | 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 }  ]) |

를 이용해서 Cursor 문법을 확인해 보겠습니다.

# Sort

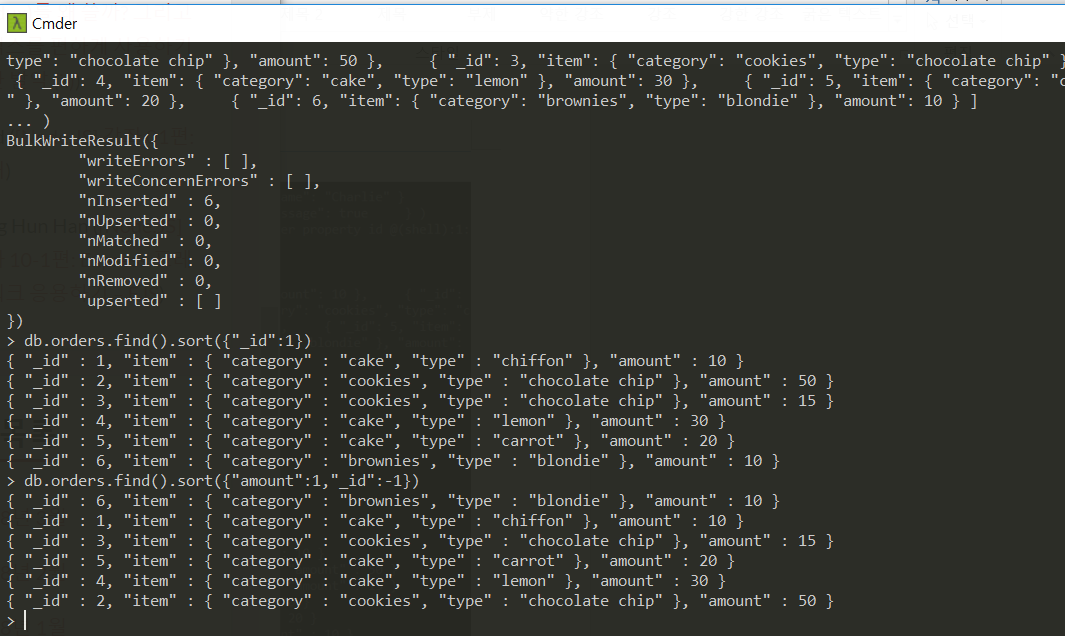
sort는 말 그대로 Document 조회 결과인 cursor에 대해 정렬을 할 때 쓰는 문법이다. Relation Database에서는 Order By와 같은 문맥이다. 예를 들어 가수 이름 순서대로 조회할 때 아래와 같이 작성을 해서 결과를 출력할 수 있다.



# 명령어

|  |  |
| --- | --- |
| 1 | db.orders.find().sort({"\_id":1}) |

명령어 : db.orders.find().sort({"\_id":1}) 로 \_id 값의 오름차순 정렬



# 명령어

|  |  |
| --- | --- |
| 1 | db.orders.find().sort({"amount":1,"\_id":-1}) |

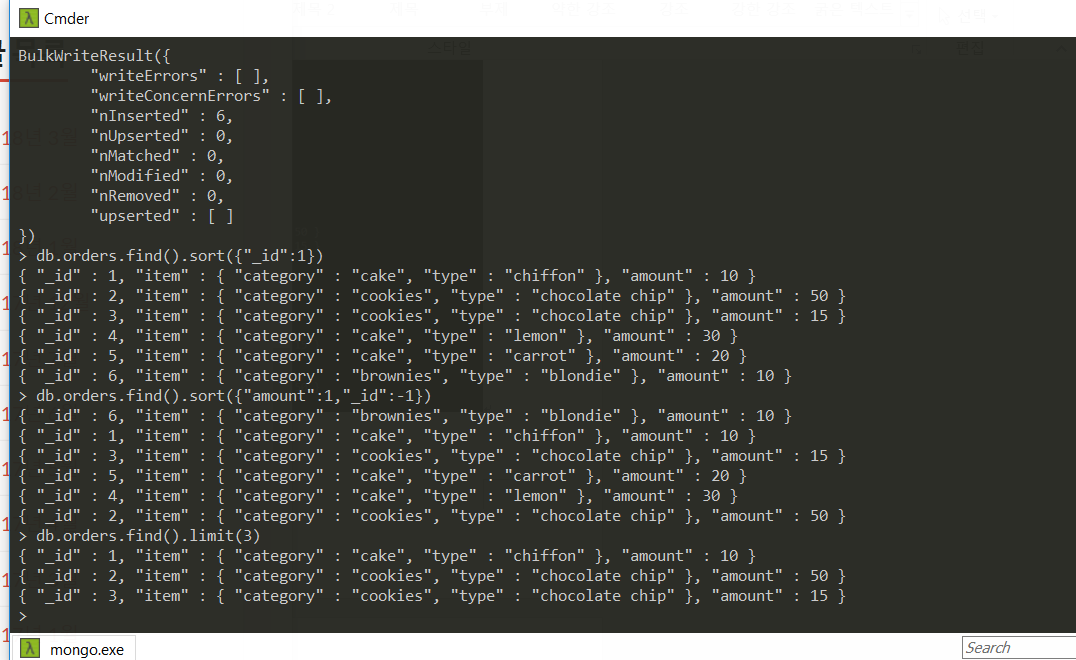
명령어 : db.orders.find().sort({"amount":1,"\_id":-1}) 로

Amount는 오름차순 \_id는 내림차순으로 정렬 sort(“name”:1) 이 오름차순

sort(“name”:-1) 내림차순

# Limit

limit는 Relational Database에서도 Record들의 수를 제한할 때 쓴 함수도 있긴 하지만 RDBMS의 환경에 따라 제공 여부를 따져서 이용을 해야 한다. 그렇지만 다행이 MongoDB에서는 조회 결과에 따른 Document들의 수에 대하여 제한을 둘 수 있다.



# 명령어

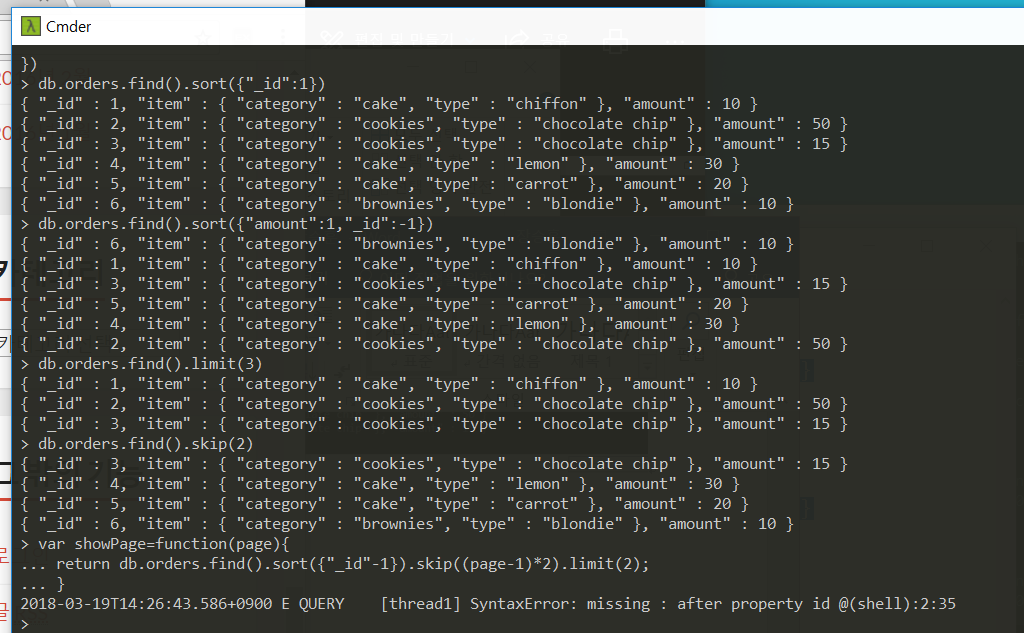
|  |  |
| --- | --- |
| 1 | db.orders.find().limit(3) |

명령어: db.orders.find().limit(3)

로 찾는 개수 3개로 제한

# Skip

skip은 우리가 MySQL에서 썼던 LIMIT 함수에 대해 생각을 해 볼 수 있는데 예를 들어 SQL 질의 마지막에LIMIT(0, 15)를 쓴다면 처음 Record(0번째 Record)부터 15개를 가져오는 역할을 한다. MongoDB에서도 마찬가지로 어느 번째 Document를 출력할지에 대한 질의를 작성할 수 있다.



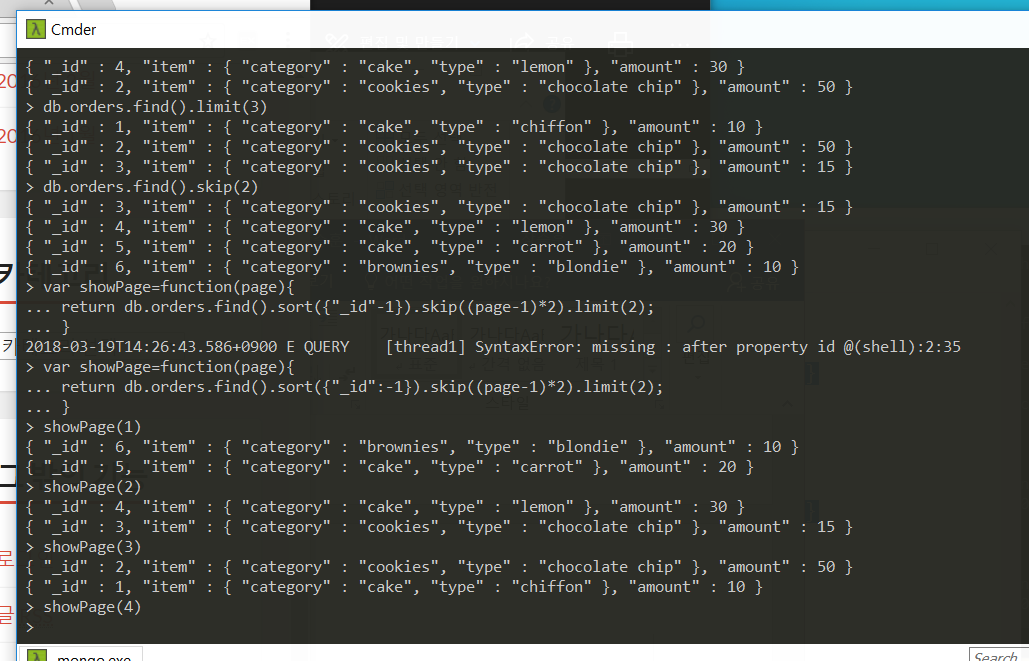
# 명령어

|  |  |
| --- | --- |
| 1 | db.orders.find().skip(2) |

찾는 목록에서 2개의 데이터를 생략하고 보여줌

# ShowPage

showPage라는 메소드를 선언한다 그래서 \_id 내림차순으로 정리하고 showPage 메소드에서 넣은 (값 -1 )\* 2 만큼 스킵해주고 2개의 document만 보여준다



# 명령어

|  |  |
| --- | --- |
| 1  2  3  4 | var showPage=function(page){  ... return db.orders.find().sort({"\_id":-1}).skip((page-1)\*2).limit(2);  ... } |

그래서 showPage(1)하면

> showPage(1)

{ "\_id" : 6, "item" : { "category" : "brownies", "type" : "blondie" }, "amount" : 10 }

{ "\_id" : 5, "item" : { "category" : "cake", "type" : "carrot" }, "amount" : 20 }

showPage(2) 하면

> showPage(2)

{ "\_id" : 4, "item" : { "category" : "cake", "type" : "lemon" }, "amount" : 30 }

{ "\_id" : 3, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 15 }

showPage(3) 하면

> showPage(3)

{ "\_id" : 2, "item" : { "category" : "cookies", "type" : "chocolate chip" }, "amount" : 50 }

{ "\_id" : 1, "item" : { "category" : "cake", "type" : "chiffon" }, "amount" : 10 }

ShowPage(4)부터는 보여주는 document가 없게 된다.

> showPage(4)

전체 document가 6개인데 6개를 스킵해서 보여줄 document가 없어진다.

6.Update

# Update 문법

Relational Database에서도 Update 문이 존재하는데 크게 나뉘어서 보면 Update 테이블\_이름 SET 설정 값 WHERE 조건 이렇게 나뉜다. 이처럼 MongoDB에서도 Update 문법이 이와 거의 유사하게 구성되어 있어서 공부하는데 크게 어렵지 않지만 Query를 작성할 때 복잡하게 느낄 것이다. 어떻게 구성이 되어 있는지 살펴보도록 한다.

# 명령어

|  |  |
| --- | --- |
| 1  2  3  4  5 | db.people.insert([  { name: "Abet", age: 19 }, { name: "Betty", age: 20 }, { name: "Charlie", age: 23, skills: [ "mongodb", "nodejs"] }, { name: "David", age: 23, score: 20 }  ]) |

Update 문법을 확인하기 위해 People collection에 name과 age를 가진 document들을 insert 한다

# Update 함수의 구조

update 함수는 일단 작성하기 앞서 어떻게 구성되어 있는지 살펴보도록 하겠다. 우리가 Relational Database에서 흔히 쓴 Update 문과는 그나마 유사한 편이지만, MongoDB에서 쓰는 update 문은 일반 Update 문에서 추가된 요소들이 존재한다.

|  |
| --- |
| db.콜렉션\_이름.update(  { Query 질의 },  { RDB Update의 SET에 해당되는 데이터 조율 },  {  upsert : true / false,  multi : true / false,  writeConcern : { Document }  }  } |

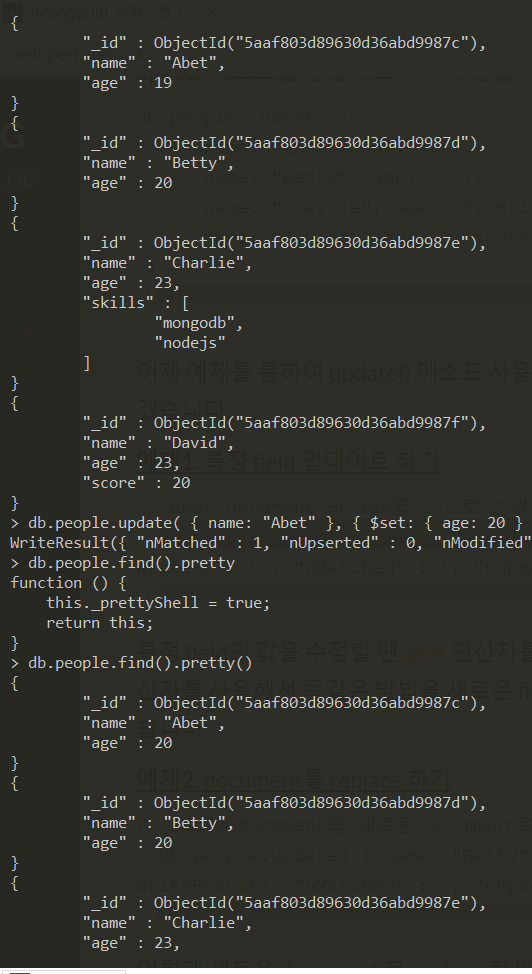
새로 추가 된 개념은 upsert, multi, writeConcern으로 볼 수 있는데 이 세 가지 옵션에 대한 정의는 아래와 같다.

upsert : Query로 조회한 Document가 존재하지 않으면 아예 새로운 Document를 추가한다. 기본 값은 false.

multi : Query로 조회한 Document들 중에서 맨 초기에 추가한 Document만 Update를 진행하면 false, 여러 Document들을 수정을 원한다면 true. 기본 값은 false로 되어 있어서 예를 들어 수치를 이용한 질의 갱신을 하는 경우에는 달랑 한 Document만 갱신되어 익숙하지 않을 것이다.

writeConcern : 이는 Document 갱신 시 성공적으로 동작을 하였는지에 대하여 인지하기 위해 쓰는 변수로 볼 수 있는데 여기서는 크게 이용하지 않고 넘어가도록 하겠다

# Update



# 명령어

|  |  |
| --- | --- |
| 1 | db.people.update( { name: "Abet" }, { $set: { age: 20 } } ) |

db.people.update( { name: "Abet" }, { $set: { age: 20 } } ) 으로 name Abet인 사람의 age를 20으로 바꾼다

Abet의 age가 20으로 바뀐 것을 알 수 있다.



# 명령어

|  |  |
| --- | --- |
| 1 | db.people.update( { name: "Betty" }, { "name": "Betty 2nd", age: 1 }) |

굳이 $set 연산자를 쓰지 않고도 document를 update 할 수 있다.

Name이 Betty의 document를 name은 betty2nd 나이는 1 인 document로 update 했다 .이렇게 하더라도 \_id는 바뀌지 않는 것을 알 수 있다.



# 명령어

|  |  |
| --- | --- |
| 1 | db.people.update( { name: "David" }, { $unset: { score: 1 } } ) |

특정 field를 제거 하기

name이 david의 score filed를 지워버렸다. Unset 연산자를 통해서 score : 1을 넣었다 여기서 1은 true라는 뜻이다.



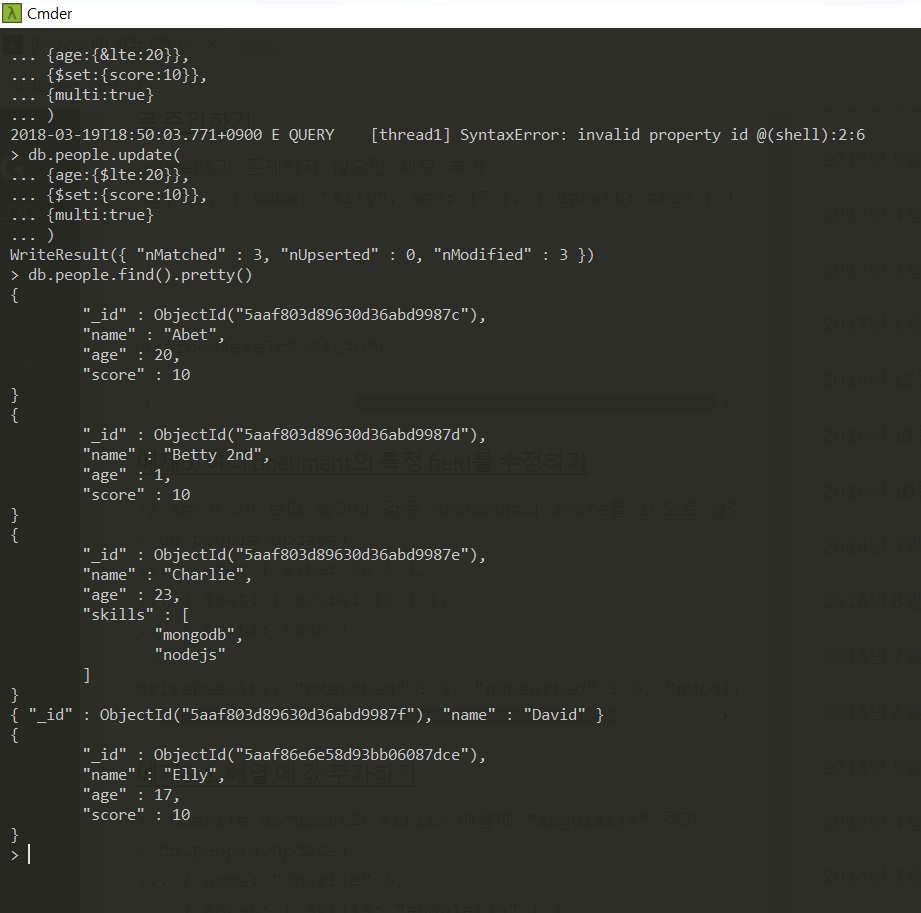
# 명령어

|  |  |
| --- | --- |
| 1 | db.people.update( { name: "Elly" }, { name: "Elly", age: 17 }, { upsert: true } ) |

Update 할 document가 없으면 create 하는 명령어

명령어: db.people.update( { name: "Elly" }, { name: "Elly", age: 17 }, { upsert: true } )

Name이 elly인 사람이 없으면 새로 추가해준다



# 명령어

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | db.people.update(  ... {age:{$lte:20}},  ... {$set:{score:10}},  ... {multi:true}  ... ) |

Age가 20 보다 낮거나 같은 document의 score 10으로 설정

저기서 multi : true는 다중성을 부여한 것이다.

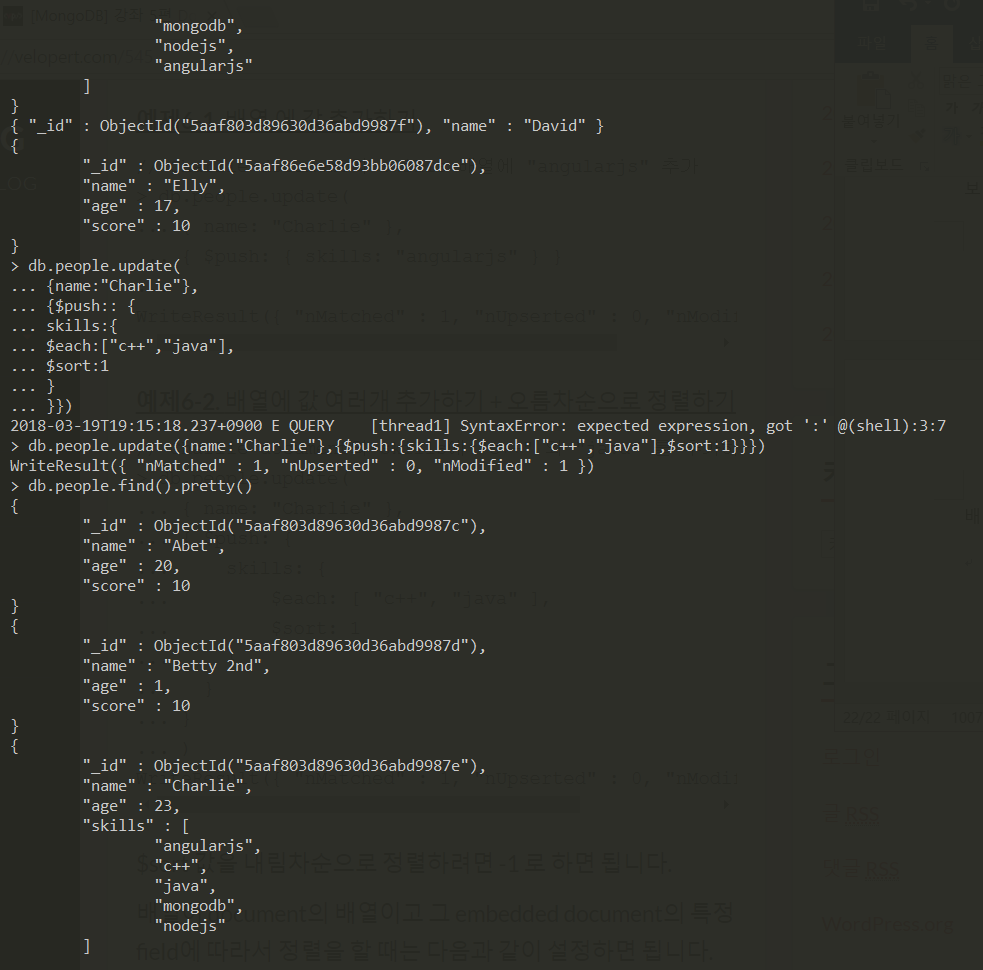


# 명령어

|  |  |
| --- | --- |
| 1  2  3  4  5 | db.people.update(  ... {name:"Charlie"},  ... {$push:{skills:"angularjs"}}  ... ) |

배열에 값 추가하기 document 안에 있는 배열인 skills에 값을 추가할 것이다 .

보면 name이 Charlie인 document의 skills배열에 angularjs가 추가 된 것을 알 수 있다 .



# 명령어

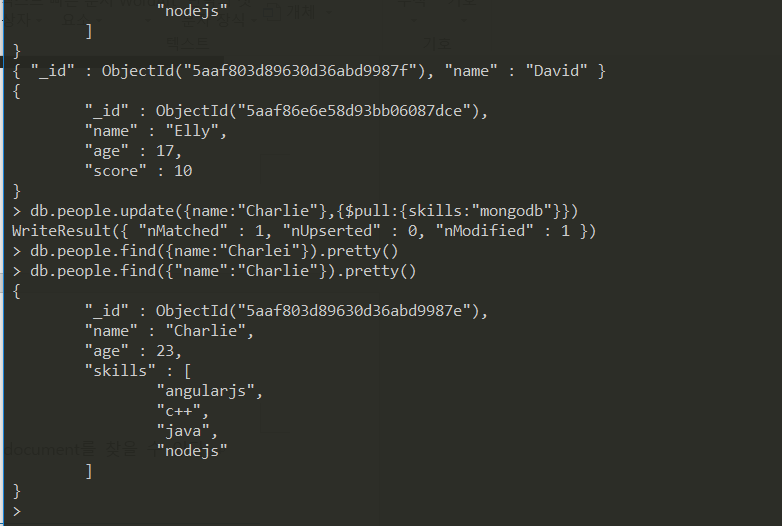
|  |  |
| --- | --- |
| 1 | db.people.update({name:"Charlie"},{$push:{skills:{$each:["c++","java"],$sort:1}}}) |

배열에 값 여러 개 추가하기 + 오름차순으로 정렬하기

명령어 : db.people.update({name:"Charlie"},{$push:{skills:{$each:["c++","java"],$sort:1}}})

를 하면 name이 charlie에 skills 배열에 c++ 그리고 java를 추가하고 오름차순으로 정렬을 해주는 명령어이다 -1이면 내림차순 정렬 1이면 오름차순 정렬

Name이 Charlie인 skills 배열에 c++ java가 추가된 것을 볼 수 있다.

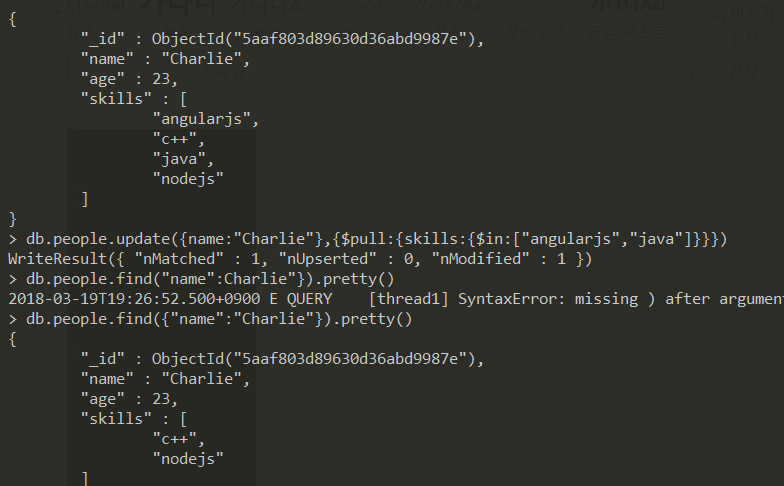


# 명령어

|  |  |
| --- | --- |
| 1 | db.people.update({name:"Charlie"},{$pull:{skills:"mongodb"}}) |

다음은 배열에 있는 값 제거하기

보면 skills에서 mongodb가 사라진 것을 알 수 있다 .



# 명령어

|  |  |
| --- | --- |
| 1 | db.people.update({name:"Charlie"},{$pull:{skills:{$in:["angularjs","java"]}}}) |

추가처럼 여러 개 삭제하기

skills의 angularjs와 java가 삭제된 것을 볼 수 있다

기초는 여기서 마무리하고 다음 장에서 User에 대해 설명하고 Spring 연동으로 넘어가도록 하겠습니다.