

Document find & update 문

소프트웨어공학과 / 201332001 강인성 / hogu9401@gmail.com

1 find 문법	2~6
A. find 함수의 구조	2
B. 기본 조회와 비교 연산자	2~5
C. 조건 연산자	5
D. Projection	6
2 cursor 문법	6~9
A. sort 문법	7
B. limit 문법	7~8
C. skip 문법	8~9
3 update 문법	10~14
A. update 함수의 구조와 기본 사용법	10~12
B. update 추가 연산 사용법	12~14

1. find() 문법

A. find 함수의 구조

find 함수는 아래와 같이 구성이 되어 있다.

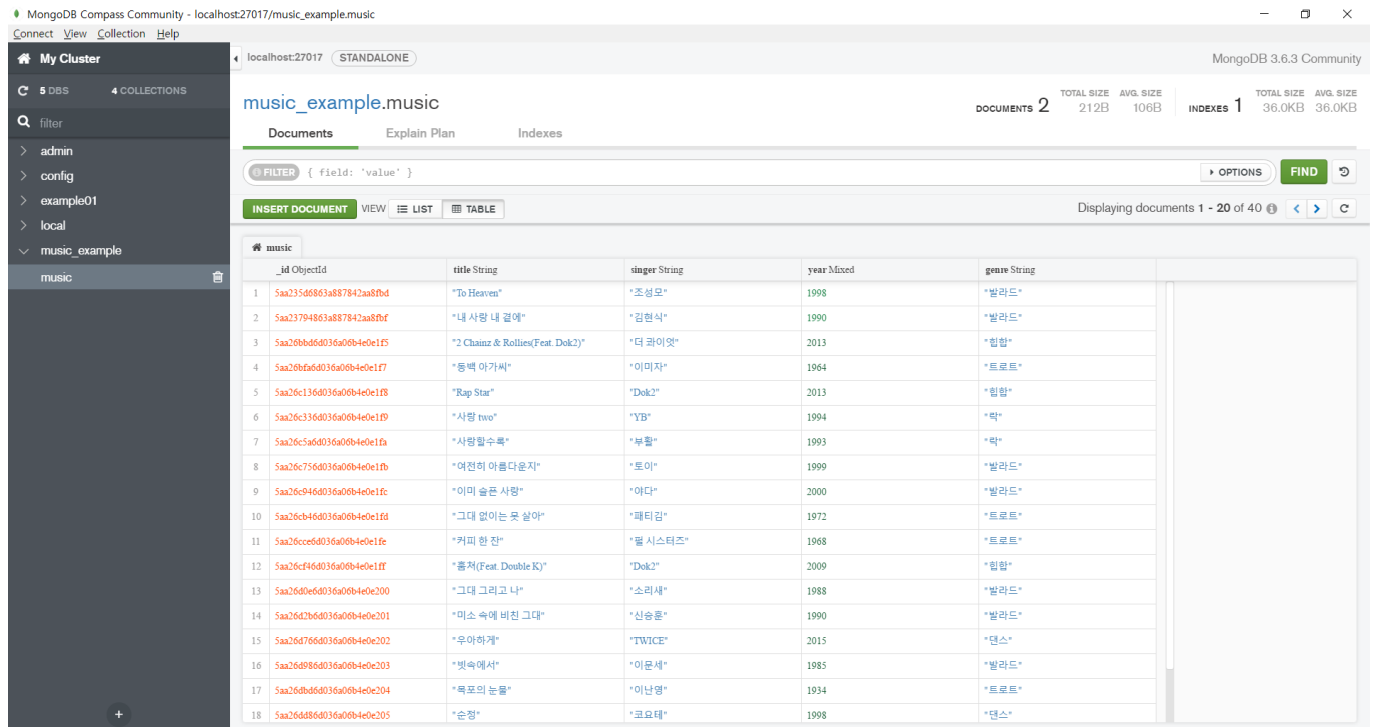
```
> db.music.find({title : "나 항상 그대를"})  
[ db.컬렉션_이름.find({ Query 조건 }, projection(조회할 필드 설정)) ]
```

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 마찬가지로인 셈이다. Cursor 객체가 반환이 된다면 추후에 이를 정렬화, 데이터 수의 제약 등을 둘 수 있는데 이에 대해서는 Update 문이 끝나고 구체적으로 들어가보도록 하자.

이전 MongoDB 기초 스터디 노트에서 find() 함수를 이용한다면 모든 데이터들을 조회하는 사실을 배웠다. 여기서 출력되는 순서는 _id의 값을 순서대로 출력이 되었다. 그렇지만 이번에는 find 함수에 Query 조건을 추가를 해서 일부 데이터들만 출력이 되도록 연습을 해보도록 하겠다.

B. 기본 조회와 비교 연산자

비교 연산자에 대하여 알아보기 위해 방금 music Collection에 mock-up data를 임시로 만들었다. 참고로 MongoDB Compass를 활용하면 쉽게 Document 들을 Insert할 수 있으니 실험용 mock-up data를 만드는데 어려움이 없을 것이다. 또한 이는 MongoDB를 통한 CRUD 실습을 위해 쓰이니 참고하길 바란다.



	_id ObjectId	title String	singer String	year Mixed	genre String
1	5aa235d6863a887842aa8fbd	"To Heaven"	"조성모"	1998	"발라드"
2	5aa23794863a887842aa8fbd	"내 사랑 내 곁에"	"김현식"	1990	"발라드"
3	5aa28b6d6d036a0b4e0e1f5	"2 Chanz & Rollies(Feat. Dok2)"	"더 콰이엇"	2013	"힙합"
4	5aa28b6d6d036a0b4e0e1f7	"동백 아가씨"	"이미지"	1964	"트로트"
5	5aa26c136d036a0b4e0e1f8	"Rap Star"	"Dok2"	2013	"힙합"
6	5aa26c336d036a0b4e0e1f9	"사랑 two"	"YB"	1994	"락"
7	5aa26c5ad036a0b4e0e1fa	"사랑할수록"	"부활"	1993	"락"
8	5aa26c756d036a0b4e0e1fb	"여전히 아름다운지"	"토이"	1999	"발라드"
9	5aa26c946d036a0b4e0e1fc	"이미 슬픈 사랑"	"애다"	2000	"발라드"
10	5aa26cb46d036a0b4e0e1fd	"그대 없이는 못 살아"	"패티김"	1972	"트로트"
11	5aa26ccfd036a0b4e0e1fe	"커피 한 잔"	"몰 시스템즈"	1968	"트로트"
12	5aa26cf46d036a0b4e0e1ff	"홀쳐(Feat. Double K)"	"Dok2"	2009	"힙합"
13	5aa26d0e6d036a0b4e0e200	"그대 그리고 나"	"소리새"	1988	"발라드"
14	5aa26d2b6d036a0b4e0e201	"미소 속에 비친 그대"	"신승훈"	1990	"발라드"
15	5aa26d766d036a0b4e0e202	"우아하게"	"TWICE"	2015	"댄스"
16	5aa26d986d036a0b4e0e203	"빗속에서"	"이문세"	1985	"발라드"
17	5aa26db6d036a0b4e0e204	"목포의 눈물"	"이난영"	1934	"트로트"
18	5aa26dd86d036a0b4e0e205	"순정"	"크로티"	1998	"댄스"

[그림] 임시 Mock-up data 목록들

(1) find() 함수를 통한 데이터를 깔끔하게 정리한 출력

이는 MongoDB Shell를 이용한 출력을 할 때에만 쓰지 이외에는 쓸 일이 없다. find 함수 맨 뒤에 pretty() 함수만 써 주면 다음과 같은 결과가 나온다.

```
> db.music.find().pretty()
```



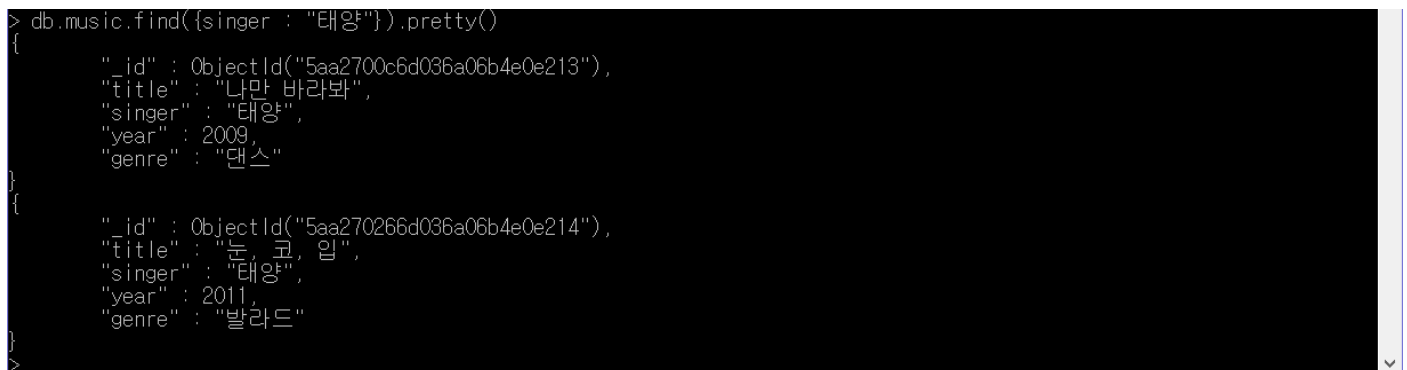
```
mongosh> db.music.find().pretty()
{
  "_id" : ObjectId("5aa235d6863a887842aa8fbd"),
  "title" : "To Heaven",
  "singer" : "조성모",
  "year" : 1998,
  "genre" : "발라드"
}
{
  "_id" : ObjectId("5aa23794863a887842aa8fbf"),
  "title" : "내 사랑 내 곁에",
  "singer" : "김현식",
  "year" : 1990,
  "genre" : "발라드"
}
{
  "_id" : ObjectId("5aa26bbd6d036a06b4e0e1f5"),
  "title" : "2 Chainz & Rollies(Feat. Dok2)",
  "singer" : "더 콰이엇",
  "year" : 2013,
  "genre" : "힙합"
}
{
  "_id" : ObjectId("5aa26bfa6d036a06b4e0e1f7"),
  "title" : "동백 아가씨",
  "singer" : "이미자",
  "year" : 1964,
  "genre" : "트로트"
}
```

[그림] pretty() 함수를 이용한 출력 결과

(2) 가수 이름이 태양인 데이터를 조회

데이터가 항상 일치할 때 찾는 Query 문은 그 데이터에 해당되는 field와 함께 작성하면 된다.

```
> db.music.find({singer : "태양"}).pretty()
[ db.컬렉션_이름.find({ 해당 필드 : 필드 데이터 ... })(pretty 함수는 선택) ]
```



```
mongosh> db.music.find({singer : "태양"}).pretty()
{
  "_id" : ObjectId("5aa2700c6d036a06b4e0e213"),
  "title" : "나만 바라봐",
  "singer" : "태양",
  "year" : 2009,
  "genre" : "댄스"
}
{
  "_id" : ObjectId("5aa270266d036a06b4e0e214"),
  "title" : "눈, 코, 입",
  "singer" : "태양",
  "year" : 2011,
  "genre" : "발라드"
}
```

[그림] 항상 일치한 데이터를 통한 조회 실행 결과

(3) 1990년대 노래 데이터를 조회

비교 연산자를 통하여 조회할 때에는 당연히 부등호를 이용해야 할 것이다. 부등호는 직접 입력하는 것이 아니라 Less Than, Greater Than 등의 영문을 줄여서 입력을 할 수 있다. 다음과 같이 작성을 해보면 아래 그림과 같은 결과가 나온다.

```
> db.music.find({year : {$gte : 1990, $lte : 1999}}).pretty()
[ db.컬렉션_이름.find({ 해당 필드 : { $부등호 : 데이터 ... } ... })(pretty 함수는 선택) ]
```

```
> db.music.find({year : {$gte : 1990, $lte : 1999}}).pretty()
{
  "_id" : ObjectId("5aa235d6863a887842aa8fbd"),
  "title" : "To Heaven",
  "singer" : "조성모",
  "year" : 1998,
  "genre" : "발라드"
}
{
  "_id" : ObjectId("5aa23794863a887842aa8fbf"),
  "title" : "내 사랑 내 곁에",
  "singer" : "김현식",
  "year" : 1990,
  "genre" : "발라드"
}
{
  "_id" : ObjectId("5aa26c336d036a06b4e0e1f9"),
  "title" : "사랑 two",
  "singer" : "YB",
  "year" : 1994,
  "genre" : "락"
}
```

[그림] 부등호를 활용한 결과

(4) 일부 가수들에 대한 데이터들 조회

일부 데이터를 가져올 때에는 IN 연산자를 요구한다. IN 연산자도 마찬가지로 데이터 별로 이용이 가능하다. 예를 들어 가수 이름이 Dok2, The Quiett인 노래를 가져오는 연습을 해보면 아래와 같다.

```
> db.music.find({singer : {$in : ["Dok2", "The Quiett"]}}).pretty()
[ db.컬렉션_이름.find({ 해당 필드 : { $in : [데이터1, ...] ... } ... })(pretty 함수는 선택) ]
```

```
> db.music.find({singer : {$in : ["Dok2", "The Quiett"]}}).pretty()
{
  "_id" : ObjectId("5aa26bbd6d036a06b4e0e1f5"),
  "title" : "2 Chainz & Rollies(Feat. Dok2)",
  "singer" : "The Quiett",
  "year" : 2013,
  "genre" : "힙합"
}
{
  "_id" : ObjectId("5aa26c136d036a06b4e0e1f8"),
  "title" : "Rap Star",
  "singer" : "Dok2",
  "year" : 2013,
  "genre" : "힙합"
}
{
  "_id" : ObjectId("5aa26cf46d036a06b4e0e1ff"),
  "title" : "훔쳐(Feat. Double K)",
  "singer" : "Dok2",
  "year" : 2009,
  "genre" : "힙합"
}
```

[그림] in 연산자를 이용한 실행 결과

(5) 비교 연산자 이용 가능한 목록

비교 연산자는 다음과 같은 종류들이 존재하며 알맞게 사용하여 작성하면 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. RDBMS 의 IN 과 같다.
\$nin	주어진 배열 안에 속하지 않는 값. RDBMS 의 Not IN 과 같다.

C. 조건 연산자

조건 연산자는 and, or, not, nor(원래는 xor인데 여기서는 nor로 쓴다.) 4가지로 나뉜다. 이를 이용하기 위해서 방금 비교 연산자처럼 \$and, \$or 등으로 작성해서 사용하면 되니 크게 설명하지 않고 넘어간다.

예를 들어 가수 Dok2가 부른 노래 중에서 2010년 이전의 노래 목록들을 출력하는 문장을 작성하면 아래와 같다.

```
> db.music.find({ $and : [ { singer : "Dok2"}, { year : { $lte : 2010 } } ] }).pretty()
[ db.컬렉션_이름.find({ 해당 필드 : { $조건 연산자 : [Field 조건들, ...] ... } ... })(pretty 함수는 선택) ]
```

참고로 조건 연산자를 함께 이용할 때에는 생각보다 괄호 주의를 해야 한다.

```
> db.music.find({$and : [{singer : "Dok2"}, {year : {$lte : 2010}}]}).pretty()
{
  "_id" : ObjectId("5aa26cf46d036a06b4e0e1ff"),
  "title" : "훙쳐(Feat. Double K)",
  "singer" : "Dok2",
  "year" : 2009,
  "genre" : "힙합"
}
{
  "_id" : ObjectId("5aa2728a6d036a06b4e0e217"),
  "title" : "On My Way(Feat. Zion.T)",
  "singer" : "Dok2",
  "year" : 2009,
  "genre" : "힙합"
}
```

[그림] 방금 전 조건 대로 실행 결과를 출력하면 위 그림과 같다.

그리고 조건 연산자 이외에도 정규식을 통한 비교인 \$regex 연산자, JavaScript에서 사용하는 함수를 입력하여 검색이 가능한 where 연산자, Document 내부에 존재하는 배열 값으로 검색을 하기 위한 \$elemMatch 연산자 등이 있는데 이에 대하여 이야기를 하면 장황해지기 때문에 나중에 이용할 기회가 있다면 추가 문서에 작성을 해 두겠다.

D. Projection

Relational Database에서는 Table에 있는 모든 데이터를 가져오는 것이 아닌 일부의 데이터를 가져올 때 Select 문에서 Column을 이용해서 조회를 했었다. 또한 REST API를 이용하게 된다면 어느 데이터에 대해서는 분명히 제한하게 되는 경우도 고려를 해야 한다. find 함수에서도 이를 반영을 해서 어느 데이터들을 보낼지에 대하여 작성을 할 수 있는 Query도 존재한다. 바로 Projection을 이용하면 된다.

예를 들어 music Collection에서 title와 singer만 나오게끔 하고 싶으면 아래와 같은 문장으로 작동하고 확인할 수 있다. 그렇지만 Projection을 쓰는 객체 내부에서 다른 Field들에 대해서 true를 표시를 안 하면 다행이 안 나오지만, _id에 대해서 표시를 안 하고 싶은 경우에는 다른 Field와는 다르게 앞에 _id : false를 작성하여 안 나오게끔 해야 한다.

```
> db.music.find({}, { _id : false, title : true, singer : true }).pretty()
[ db.컬렉션_이름.find( { Query 조건들 }, { 컬럼들 : 표시 여부(true / false) ... })(pretty 함수는 선택) ]
```



```
명령 프롬프트 - mongo
> db.music.find({}, { _id : false, title : true, singer : true }).pretty()
{"title": "To Heaven", "singer": "조성모"}
{"title": "내 사랑 내 곁에", "singer": "김현식"}
{"title": "2 Chainz & Rollies(Feat. Dok2)", "singer": "The Quiett"}
{"title": "동백 아가씨", "singer": "이미자"}
{"title": "Rap Star", "singer": "Dok2"}
{"title": "사랑 two", "singer": "YB"}
{"title": "사랑할수록", "singer": "부활"}
{"title": "여전히 아름다운지", "singer": "토이"}
{"title": "이미 슬픈 사랑", "singer": "야다"}
{"title": "그대 없이는 못 살아", "singer": "패티김"}
{"title": "커피 한 잔", "singer": "폴 시스터즈"}
{"title": "훔쳐(Feat. Double K)", "singer": "Dok2"}
{"title": "그대 그리고 나", "singer": "소리새"}
{"title": "미소 속에 비친 그대", "singer": "신승훈"}
{"title": "우아하게", "singer": "TWICE"}
{"title": "빗속에서", "singer": "이문세"}
{"title": "속포의 눈물", "singer": "이난영"}
{"title": "순정", "singer": "코요테"}
{"title": "Profile(Feat. The Quiett & Dok2)", "singer": "Beenzino"}
{"title": "니가 싫어하는 노래(Feat. Jay Park)", "singer": "Dok2"}
Type "it" for more
>
```

[그림] 위의 문장을 mongo 서버에서 돌린 결과.

2. 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.

➔ Next Page

A. sort 문법

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.

```
> db.music.find().sort({ singer : 1 }).pretty()
[ db.컬렉션_이름.find(Query, Projection).sort({ Field 이름 : 1 / -1 ... })(pretty 함수는 선택) ]
```

A screenshot of a MongoDB command prompt window titled '명령 프롬프트 - mongo'. The prompt shows a query: > db.music.find().sort({ singer : 1 }). The output is a list of JSON documents sorted by the 'singer' field in ascending order. The first document is {"_id": ObjectId("5aa26e076d036a06b4e0e206"), "title": "Profile(Feat. The Quiett & Dok2)", "singer": "Beenzino", "year": 2012, "genre": "힙합"}. The last document shown is {"_id": ObjectId("5aa26fcd6d036a06b4e0e211"), "title": "컵쟁이", "singer": "버즈", "year": 2006, "genre": "발라드"}. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

[그림] 위의 문장을 실행한 결과. 1로 입력을 하면 가수 이름을 순서대로 영문자 a부터 한글 순으로 출력이 되는 결과를 볼 수 있다.

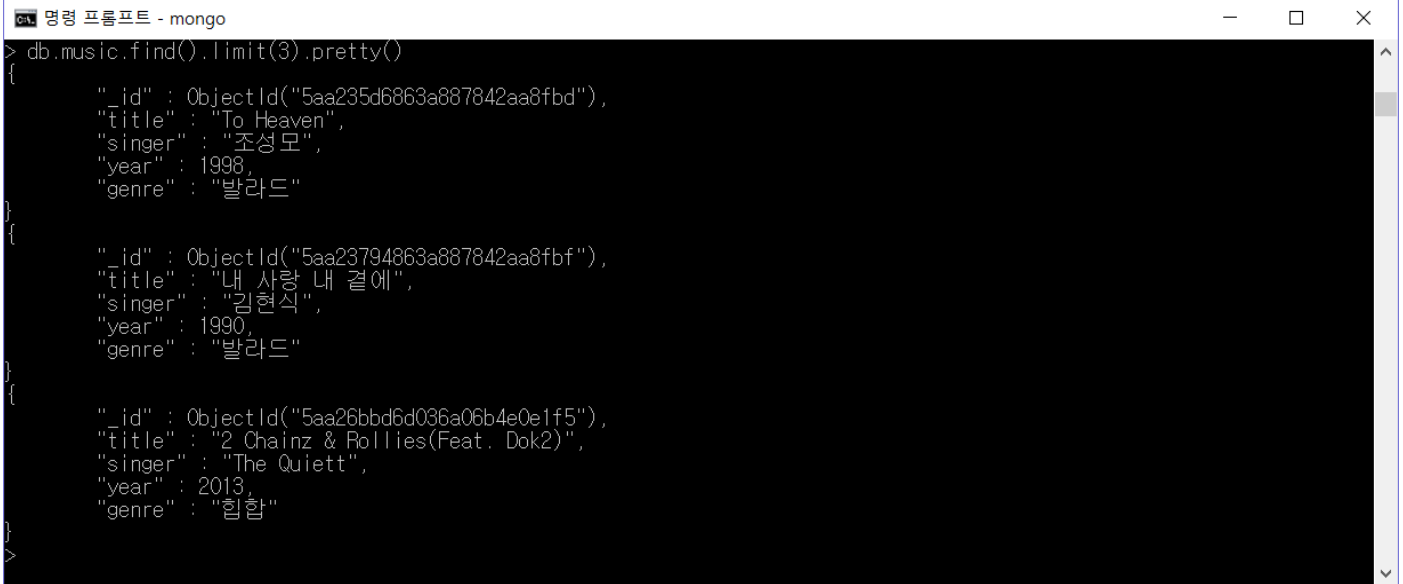
참고로 조회 순서를 오름차순(숫자(0-9), 알파벳(A-Z), 한글(ㄱ-ㅎ))으로 하고 싶은 경우는 이에 해당되는 Field를 1로 작성하고, 내림차순(한글(ㅎ-ㄱ), 알파벳(Z-A), 숫자(9-0))을 원한다면 -1로 작성을 하면 된다. 또한 여러 Field들에 대하여 설정을 할 수 있다.

B. limit 문법

limit는 Relational Database에서도 Record들의 수를 제한할 때 쓴 함수도 있긴 하지만 RDBMS의 환경에 따라 제공 여부를 따져서 이용을 해야 한다. 그렇지만 다행히 MongoDB에서는 조회 결과에 따른 Document들의 수에 대하여 제한을 둘 수 있다. 예를 들어 등록된 음악들 목록 중에서 3개만 가져오는 문장을 다음 페이지와 같이 작성할 수 있다.

➔ Next Page

```
> db.music.find().limit(3).pretty()
[ db.컬렉션_이름.find(Query, Projection).limit(제한_개수)(pretty 함수는 선택) ]
```



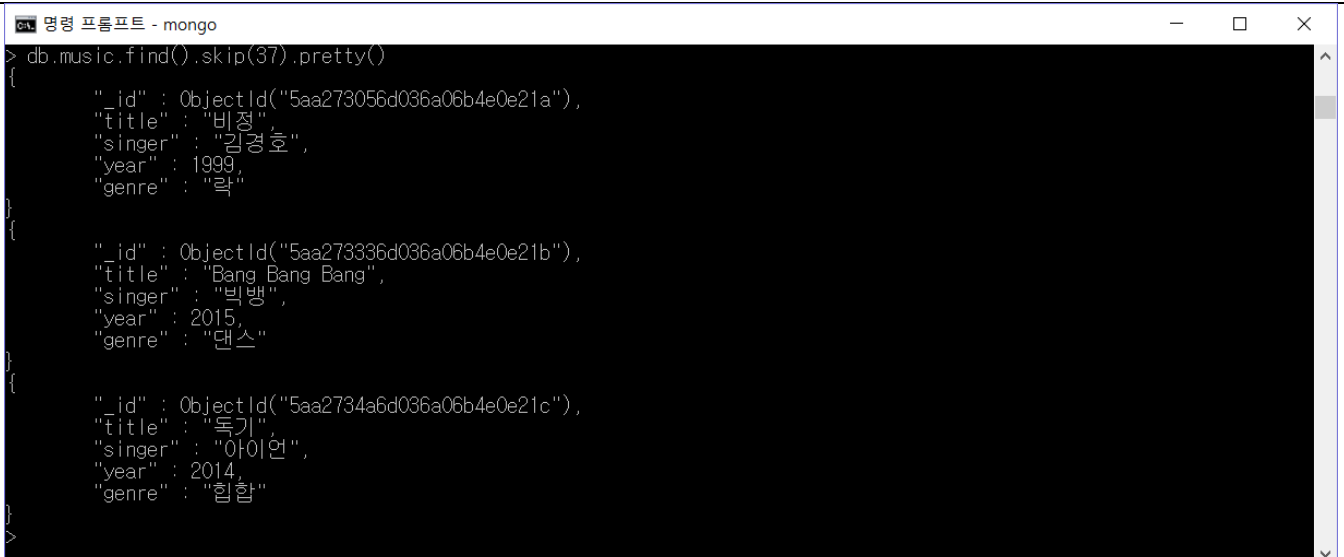
```
명령 프롬프트 - mongo
> db.music.find().limit(3).pretty()
{
  "_id" : ObjectId("5aa235d6863a887842aa8fbd"),
  "title" : "To Heaven",
  "singer" : "조성모",
  "year" : 1998,
  "genre" : "발라드"
}
{
  "_id" : ObjectId("5aa23794863a887842aa8fbf"),
  "title" : "내 사랑 내 곁에",
  "singer" : "김현식",
  "year" : 1990,
  "genre" : "발라드"
}
{
  "_id" : ObjectId("5aa26bbd6d036a06b4e0e1f5"),
  "title" : "2 Chainz & Rollies(Feat. Dok2)",
  "singer" : "The Quiett",
  "year" : 2013,
  "genre" : "힙합"
}
>
```

[그림] 위의 문장을 실행한 결과. limit 함수는 Document 수를 제한하는 기능이기 때문에 쉽게 이해할 수 있으니 자세한 설명은 생략하겠다.

C. skip 문법

skip은 우리가 MySQL에서 썼던 LIMIT 함수에 대해 생각을 해 볼 수 있는데 예를 들어 SQL 질의 마지막에 LIMIT(0, 15)를 쓴다면 처음 Record(0번째 Record)부터 15개를 가져오는 역할을 한다. MongoDB에서도 마찬가지로 어느 번째 Document를 출력할지에 대한 질의를 작성할 수 있다. 예를 들어 37번째 Document부터 가져오는 질의는 아래와 같이 작성할 수 있다.

```
> db.music.find().skip(37).pretty()
[ db.컬렉션_이름.find(Query, Projection).skip(시작_Document_번지)(pretty 함수는 선택) ]
```



```
명령 프롬프트 - mongo
> db.music.find().skip(37).pretty()
{
  "_id" : ObjectId("5aa273056d036a06b4e0e21a"),
  "title" : "비정",
  "singer" : "김경호",
  "year" : 1999,
  "genre" : "락"
}
{
  "_id" : ObjectId("5aa273336d036a06b4e0e21b"),
  "title" : "Bang Bang Bang",
  "singer" : "빅뱅",
  "year" : 2015,
  "genre" : "댄스"
}
{
  "_id" : ObjectId("5aa2734a6d036a06b4e0e21c"),
  "title" : "독기",
  "singer" : "아이언",
  "year" : 2014,
  "genre" : "힙합"
}
>
```

[그림] 실행을 한 결과 38번째로 등록한 노래 비정부부터 출력이 된다. 번지는 추가한 Document의 순서 중 1를 빼서 작성을 해야 한다.

이제 limit, skip 함수를 통하여 Document 목록의 수를 제한하는 기능을 알았으니 이젠 페이지 별로 출력을 할 수 있는 논리를 구상할 수 있다. 과거에 Pagination을 공부를 한 경험이 있다면 이 논리를 구상하는데 어렵지 않게 작성할 수 있을 것이다. 함수는 JavaScript를 이용해서 작성을 하였다. 정렬 기준은 노래 제목 오름차순으로 작성을 하겠다.

```
function myPage(page, size){
  if(size<0 || page<=0) return null;
  return db.music.find().sort({title : 1}).skip((page-1)*size).limit(size);
}
> db.music.find().skip(37).pretty()
[ db.컬렉션_이름.find(Query, Projection).skip(시작_Document_번지)(pretty 함수는 선택) ]
```

```
> function myPage(page, size){
... if(size<0 || page<=0) return null;
... return db.music.find().sort({title : 1}).skip((page-1)*size).limit(size);
... }
> myPage(2, 5)
{ "_id" : ObjectId("5aa2728a6d036a06b4e0e217"), "title" : "On My Way(Feat. Zion.T)", "singer" : "Dok2", "year" : 2009, "genre" : "힙합" }
{ "_id" : ObjectId("5aa26e076d036a06b4e0e206"), "title" : "Profile(Feat. The Quiett & Dok2)", "singer" : "Beenzino", "year" : 2012, "genre" : "힙합" }
{ "_id" : ObjectId("5aa26c136d036a06b4e0e1f8"), "title" : "Rap Star", "singer" : "Dok2", "year" : 2013, "genre" : "힙합" }
{ "_id" : ObjectId("5aa235d6863a887842aa8fbd"), "title" : "To Heaven", "singer" : "조성모", "year" : 1998, "genre" : "발라드" }
{ "_id" : ObjectId("5aa26fad6d036a06b4e0e210"), "title" : "가시", "singer" : "버즈", "year" : 2005, "genre" : "발라드" }
> myPage(1, 3)
{ "_id" : ObjectId("5aa270396d036a06b4e0e215"), "title" : "111%", "singer" : "Dok2", "year" : "2015", "genre" : "힙합" }
{ "_id" : ObjectId("5aa26bbd6d036a06b4e0e1f5"), "title" : "2 Chainz & Rollies(Feat. Dok2)", "singer" : "The Quiett", "year" : 2013, "genre" : "힙합" }
{ "_id" : ObjectId("5aa273336d036a06b4e0e21b"), "title" : "Bang Bang Bang", "singer" : "빅뱅", "year" : 2015, "genre" : "댄스" }
> myPage(0, 3)
null
> myPage(1, -3)
null
```

	_id	title	singer	year	genre
1	Saa270396d036a06b4e0e215	"111%"	"Dok2"	"2015"	"힙합"
2	Saa26bbd6d036a06b4e0e1f5	"2 Chainz & Rollies(Feat. Dok2)"	"The Quiett"	2013	"힙합"
3	Saa273336d036a06b4e0e21b	"Bang Bang Bang"	"빅뱅"	2015	"댄스"
4	Saa26f3c6d036a06b4e0e20e	"Born Hater(Feat. Beenzino, 버벌진트, M	"에픽하이"	2015	"힙합"
5	Saa26f7b6d036a06b4e0e20f	"Endless"	"플라워"	2004	"발라드"
6	Saa2728a6d036a06b4e0e217	"On My Way(Feat. Zion.T)"	"Dok2"	2009	"힙합"
7	Saa26e076d036a06b4e0e206	"Profile(Feat. The Quiett & Dok2)"	"Beenzino"	2012	"힙합"
8	Saa26c136d036a06b4e0e1f8	"Rap Star"	"Dok2"	2013	"힙합"
9	Saa235d6863a887842aa8fbd	"To Heaven"	"조성모"	1998	"발라드"
10	Saa26fad6d036a06b4e0e210	"가시"	"버즈"	2005	"발라드"

[그림] 실행 결과. 빨간 윤곽선은 myPage(2, 5)의 결과, 파란 윤곽선은 myPage(1, 3)의 결과.

3. 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.

A. 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.컬렉션_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.

우선 update의 기본적인 문장들에 대해서 작성을 해서 이용을 해보자. 간단한 사례로 가수 이름이 Dok2인 Document를 도끼로 변경을 하는 문장은 아래와 같다.

```
db.music.update(  
  { singer : "Dok2" },  
  { $set : {singer : "도끼"} }  
);
```

이처럼 작성을 하고 난 후에 가수가 Dok2인 Document들에 대하여 도끼로 변경이 되는게 정상이 아닌가 싶은 생각을 한 사람도 있겠지만 다음 페이지에서 나온 스크린 샷 결과를 보면 아마 당황할 것이다.

```
> db.music.update(
... {singer: "Dok2"},
... {$set: {singer: "도끼"}}
... );
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.music.find()
{"_id" : ObjectId("5aa235d6863a887842aa8fbd"), "title" : "To Heaven", "singer" : "조성모", "year" : 1998, "genre" : ObjectId("5aa4aa7c84b9492d38752d9e") }
{"_id" : ObjectId("5aa23794863a887842aa8fbd"), "title" : "내 사랑 내 곁에", "singer" : "김현식", "year" : 1990, "genre" : "발라드" }
{"_id" : ObjectId("5aa26bbd6d036a06b4e0e1f5"), "title" : "2 Chainz & Rollies(Feat. Dok2)", "singer" : "The Quiett", "year" : 2013, "genre" : "힙합" }
{"_id" : ObjectId("5aa26b7af6d036a06b4e0e1f7"), "title" : "동백 아가씨", "singer" : "이미자", "year" : 1964, "genre" : "트로트" }
{"_id" : ObjectId("5aa26c136d036a06b4e0e1f8"), "title" : "Rap Star", "singer" : "도끼", "year" : 2013, "genre" : "힙합" }
{"_id" : ObjectId("5aa26c336d036a06b4e0e1f9"), "title" : "사랑 two", "singer" : "VB", "year" : 1994, "genre" : "락" }
{"_id" : ObjectId("5aa26c5a6d036a06b4e0e1fa"), "title" : "사랑할수록", "singer" : "부활", "year" : 1993, "genre" : "락" }
{"_id" : ObjectId("5aa26c756d036a06b4e0e1fb"), "title" : "여전히 아름다운지", "singer" : "토이", "year" : 1999, "genre" : "발라드" }
{"_id" : ObjectId("5aa26c946d036a06b4e0e1fc"), "title" : "이미 슬픈 사랑", "singer" : "야다", "year" : 2000, "genre" : "발라드" }
{"_id" : ObjectId("5aa26cb46d036a06b4e0e1fd"), "title" : "그대 없이는 못 살아", "singer" : "패티김", "year" : 1972, "genre" : "트로트" }
{"_id" : ObjectId("5aa26cce6d036a06b4e0e1fe"), "title" : "커피 한 잔", "singer" : "필 시스템즈", "year" : 1968, "genre" : "트로트" }
{"_id" : ObjectId("5aa26cf46d036a06b4e0e1ff"), "title" : "춤쳐(Feat. Double K)", "singer" : "Dok2", "year" : 2009, "genre" : "힙합" }
{"_id" : ObjectId("5aa26d0e6d036a06b4e0e200"), "title" : "그대 그리고 나", "singer" : "소리새", "year" : 1988, "genre" : "발라드" }
{"_id" : ObjectId("5aa26d2b6d036a06b4e0e201"), "title" : "미소 속에 비친 그대", "singer" : "신승훈", "year" : 1990, "genre" : "발라드" }
{"_id" : ObjectId("5aa26d766d036a06b4e0e202"), "title" : "우아하게", "singer" : "TWICE", "year" : 2015, "genre" : "댄스" }
{"_id" : ObjectId("5aa26d986d036a06b4e0e203"), "title" : "빈속에서", "singer" : "이문세", "year" : 1985, "genre" : "발라드" }
{"_id" : ObjectId("5aa26dbd6d036a06b4e0e204"), "title" : "목표의 눈물", "singer" : "이난영", "year" : 1934, "genre" : "트로트" }
{"_id" : ObjectId("5aa26dd86d036a06b4e0e205"), "title" : "순정", "singer" : "코요테", "year" : 1998, "genre" : "댄스" }
{"_id" : ObjectId("5aa26e076d036a06b4e0e206"), "title" : "Profile(Feat. The Quiett & Dok2)", "singer" : "Beenzino", "year" : 2012, "genre" : "힙합" }
{"_id" : ObjectId("5aa26e426d036a06b4e0e207"), "title" : "니가 싫어하는 노래(Feat. Jay Park)", "singer" : "Dok2", "year" : 2017, "genre" : "힙합" }
```

[그림] update() 함수를 진행하고 find() 함수를 작동한 결과. 가수가 Dok2인 노래는 맨 처음에 등록한 Rap Star에
만 반영이 되었음을 확인할 수 있다.

Relational Database에서 Update 문을 일반적으로 작성을 하고 난 뒤에 조회를 한다면 방금 우리가 작성한
Collection이 Table이었다면 각 Record에서는 가수가 Dok2인 노래들이 전부 도끼로 바뀌어 있는 것이 관례이다.
하지만 여기서는 그렇지 않다. 방금 전에 마지막에 추가하는 옵션 중에 multi 옵션이 있는데 이를 통하여 대체를
해 주면 아래와 같은 결과를 볼 수 있다.

```
db.music.update(
  { singer: "Dok2" },
  { $set: {singer: "도끼"} },
  { multi: true }
);
```

```
> db.music.update(
... {singer: "Dok2"},
... {$set: {singer: "도끼"}},
... {multi: true}
... );
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })
> db.music.find()
{"_id" : ObjectId("5aa235d6863a887842aa8fbd"), "title" : "To Heaven", "singer" : "조성모", "year" : 1998, "genre" : ObjectId("5aa4aa7c84b9492d38752d9e") }
{"_id" : ObjectId("5aa23794863a887842aa8fbd"), "title" : "내 사랑 내 곁에", "singer" : "김현식", "year" : 1990, "genre" : "발라드" }
{"_id" : ObjectId("5aa26bbd6d036a06b4e0e1f5"), "title" : "2 Chainz & Rollies(Feat. Dok2)", "singer" : "The Quiett", "year" : 2013, "genre" : "힙합" }
{"_id" : ObjectId("5aa26b7af6d036a06b4e0e1f7"), "title" : "동백 아가씨", "singer" : "이미자", "year" : 1964, "genre" : "트로트" }
{"_id" : ObjectId("5aa26c136d036a06b4e0e1f8"), "title" : "Rap Star", "singer" : "도끼", "year" : 2013, "genre" : "힙합" }
{"_id" : ObjectId("5aa26c336d036a06b4e0e1f9"), "title" : "사랑 two", "singer" : "VB", "year" : 1994, "genre" : "락" }
{"_id" : ObjectId("5aa26c5a6d036a06b4e0e1fa"), "title" : "사랑할수록", "singer" : "부활", "year" : 1993, "genre" : "락" }
{"_id" : ObjectId("5aa26c756d036a06b4e0e1fb"), "title" : "여전히 아름다운지", "singer" : "토이", "year" : 1999, "genre" : "발라드" }
{"_id" : ObjectId("5aa26c946d036a06b4e0e1fc"), "title" : "이미 슬픈 사랑", "singer" : "야다", "year" : 2000, "genre" : "발라드" }
{"_id" : ObjectId("5aa26cb46d036a06b4e0e1fd"), "title" : "그대 없이는 못 살아", "singer" : "패티김", "year" : 1972, "genre" : "트로트" }
{"_id" : ObjectId("5aa26cce6d036a06b4e0e1fe"), "title" : "커피 한 잔", "singer" : "필 시스템즈", "year" : 1968, "genre" : "트로트" }
{"_id" : ObjectId("5aa26cf46d036a06b4e0e1ff"), "title" : "춤쳐(Feat. Double K)", "singer" : "도끼", "year" : 2009, "genre" : "힙합" }
{"_id" : ObjectId("5aa26d0e6d036a06b4e0e200"), "title" : "그대 그리고 나", "singer" : "소리새", "year" : 1988, "genre" : "발라드" }
{"_id" : ObjectId("5aa26d2b6d036a06b4e0e201"), "title" : "미소 속에 비친 그대", "singer" : "신승훈", "year" : 1990, "genre" : "발라드" }
{"_id" : ObjectId("5aa26d766d036a06b4e0e202"), "title" : "우아하게", "singer" : "TWICE", "year" : 2015, "genre" : "댄스" }
{"_id" : ObjectId("5aa26d986d036a06b4e0e203"), "title" : "빈속에서", "singer" : "이문세", "year" : 1985, "genre" : "발라드" }
{"_id" : ObjectId("5aa26dbd6d036a06b4e0e204"), "title" : "목표의 눈물", "singer" : "이난영", "year" : 1934, "genre" : "트로트" }
{"_id" : ObjectId("5aa26dd86d036a06b4e0e205"), "title" : "순정", "singer" : "코요테", "year" : 1998, "genre" : "댄스" }
{"_id" : ObjectId("5aa26e076d036a06b4e0e206"), "title" : "Profile(Feat. The Quiett & Dok2)", "singer" : "Beenzino", "year" : 2012, "genre" : "힙합" }
{"_id" : ObjectId("5aa26e426d036a06b4e0e207"), "title" : "니가 싫어하는 노래(Feat. Jay Park)", "singer" : "도끼", "year" : 2017, "genre" : "힙합" }
Type "it" for more
{"_id" : ObjectId("5aa26e6f6d036a06b4e0e208"), "title" : "어디에도", "singer" : "M.C. THE MAX", "year" : 2015, "genre" : "발라드" }
{"_id" : ObjectId("5aa26e936d036a06b4e0e209"), "title" : "거리에서", "singer" : "성시경", "year" : 2006, "genre" : "발라드" }
{"_id" : ObjectId("5aa26ead6d036a06b4e0e20a"), "title" : "킬리만자로의 표범", "singer" : "조용필", "year" : 1985, "genre" : "발라드" }
{"_id" : ObjectId("5aa26eecc6d036a06b4e0e20c"), "title" : "타향살이", "singer" : "고복수", "year" : 1934, "genre" : "트로트" }
{"_id" : ObjectId("5aa26f266d036a06b4e0e20d"), "title" : "비행기", "singer" : "거북이", "year" : 2006, "genre" : "댄스" }
{"_id" : ObjectId("5aa26f3c6d036a06b4e0e20e"), "title" : "Born Hater(Feat. Beenzino, 버벌진트, MINO, BOBBY, B.I.)", "singer" : "에픽하이", "year" : 2015, "genre" : "힙합" }
{"_id" : ObjectId("5aa26f7b6d036a06b4e0e20f"), "title" : "Endless", "singer" : "플라워", "year" : 2004, "genre" : "발라드" }
{"_id" : ObjectId("5aa26f9d6d036a06b4e0e210"), "title" : "가시", "singer" : "버즈", "year" : 2005, "genre" : "발라드" }
{"_id" : ObjectId("5aa26fc6d036a06b4e0e211"), "title" : "검정치마", "singer" : "버즈", "year" : 2006, "genre" : "발라드" }
{"_id" : ObjectId("5aa26ff36d036a06b4e0e212"), "title" : "나 항상 그대를", "singer" : "이선희", "year" : 1987, "genre" : "발라드" }
{"_id" : ObjectId("5aa2700c6d036a06b4e0e213"), "title" : "나만 바라봐", "singer" : "태양", "year" : 2009, "genre" : "댄스" }
{"_id" : ObjectId("5aa2702b6d036a06b4e0e214"), "title" : "눈, 코, 입", "singer" : "태양", "year" : 2011, "genre" : "발라드" }
{"_id" : ObjectId("5aa270396d036a06b4e0e215"), "title" : "11%", "singer" : "도끼", "year" : 2015, "genre" : "힙합" }
{"_id" : ObjectId("5aa270736d036a06b4e0e216"), "title" : "난 알아요", "singer" : "서태지와 아이들", "year" : 1992, "genre" : "댄스" }
{"_id" : ObjectId("5aa2728a6d036a06b4e0e217"), "title" : "On My Way(Feat. Zion.T)", "singer" : "도끼", "year" : 2009, "genre" : "힙합" }
{"_id" : ObjectId("5aa272b06d036a06b4e0e218"), "title" : "연결고리(Feat. MC Meta)", "singer" : "Illionaire Records", "year" : 2014, "genre" : "힙합" }
{"_id" : ObjectId("5aa272e56d036a06b4e0e219"), "title" : "너를 위해", "singer" : "임재범", "year" : 2000, "genre" : "발라드" }
{"_id" : ObjectId("5aa273056d036a06b4e0e21a"), "title" : "비정", "singer" : "김경호", "year" : 1999, "genre" : "락" }
{"_id" : ObjectId("5aa273336d036a06b4e0e21b"), "title" : "Bang Bang Bang", "singer" : "빅뱅", "year" : 2015, "genre" : "댄스" }
{"_id" : ObjectId("5aa2734a6d036a06b4e0e21c"), "title" : "독기", "singer" : "아이언", "year" : 2014, "genre" : "힙합" }
```

위의 그림은 multi를 적용하고 난 후에 실행을 한 결과로 볼 수 있다. 이처럼 갱신을 하고자 하는 Document들의 질의를 입력할 때 여러 Document들이 해당되는 경우에는 multi로 조정을 해서 작성을 할 필요가 있다.

B. update 추가 연산 사용법

(1) Document 내부에 있는 Field들을 싹 다 날리고 새로 작성하기

방금 전에 \$set 연산에 대해서 잠깐 다뤄봤다. 그렇지만 우리가 NoSQL의 특성에는 일관성이 느슨하다는 사실에 대해 알고 있을 것이다. 그래서 각 Document에 Field를 새로 추가를 할 수 있는 문장들에 대해서 다뤄 보는데 예를 들어 노래 제목이 커피 한 잔인 곡에 대하여 year, genre를 빼고 title, singer, writer만 Document에 저장하는 문장을 작성을 해 보면 아래와 같이 작성을 할 수 있다.

```
> db.music.update(
  { title : "커피 한 잔" },
  { title : "커피 한 잔", singer : "필 시스터즈", writer : "신중현" }
);
[db.컬렉션_이름.update({ Query 질의 }, { new Document Fields });]
```

```
> db.music.update(
... {title : "커피 한 잔"},
... {title : "커피 한 잔", singer : "필 시스터즈", writer : "신중현"}
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.music.find({title : "커피 한 잔"}).pretty()
{
  "_id" : ObjectId("5aa26cce6d036a06b4e0e1fe"),
  "title" : "커피 한 잔",
  "singer" : "필 시스터즈",
  "writer" : "신중현"
}
```

[그림] update 문에서 \$set 연산자를 쓰지 않으면 새로운 Field가 형성하게 되어 아예 새로운 Document로 저장된다. 간혹 Document 내부에 있는 Field 내용만 수정을 하다가 이런 실수를 저지르지 않도록 유의하자.

(2) Document 내부에 있는 Field 없애기

이번에는 \$unset 연산자를 이용해서 필요 없는 Field를 아예 없애는 연습을 해보도록 하자. 주의해야 할 사항이 MongoDB는 NoSQL이기 때문에 Document 내부의 Field를 없앤다는 의미는 즉 그 Document에서 그 Field를 사용하지 않겠다는 의미로 받아 들어야 되고 Collection를 기반으로 뭉친 Field를 없앤다는 의미로 받아 들이면 안 된다. 예를 들어 그대 없이는 못 살아 라는 노래의 연도가 불분명해서 year Field를 없애는 문장을 작성하는 경우는 아래와 같이 작성할 수 있다.

```
> db.music.update(
  { title : "그대 없이는 못 살아" },
  { $unset : { year : true } }
);
[db.컬렉션_이름.update( { Query 질의 }, { $unset : { 제할 Field 이름 : true / false } ... } );]
```

```
명령 프롬프트 - mongo
> db.music.update(
... {title : "그대 없이는 못 살아"},
... {$unset : {year : 1}}
... );
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.music.find({title : "그대 없이는 못 살아"}).pretty()
{
  "_id" : ObjectId("5aa26cb46d036a06b4e0e1fd"),
  "title" : "그대 없이는 못 살아",
  "singer" : "패티김",
  "genre" : "트로트"
}
```

[그림] unset 연산자를 이용하여 제목이 그대 없이는 못 살아 에 해당 되는 Document의 Field를 제하고 find() 함수로 조회한 결과. 필자는 true가 아니라 1로 작성을 하였지만 JavaScript 기반 문서에서도 1이 true인 걸로 받아들여 상관은 없다.

(3) 질의 조회가 null일 때 새로운 Document 생성하는 경우

Document를 새로 update할 때 질의 조회가 null인 경우, 즉 조회 결과가 없는 대신에 새로운 Document를 쓸 때에는 아래와 같은 문장으로 새로운 Document를 생성할 수 있다.

```
> db.music.update(
  { title : "SIMON DOMINIC" },
  { title : "SIMON DOMINIC", singer : "Simon D.", year : 2015, genre : "힙합" },
  { upsert : true }
);
[db.컬렉션_이름.update( { Query 질의 }, { 새로 추가할 Document Fields & Values }, { upsert : true } );]

명령 프롬프트 - mongo
> db.music.update(
... { title : "SIMON DOMINIC" },
... { title : "SIMON DOMINIC", singer : "Simon D.", year : 2015, genre : "힙합" },
... { upsert : true }
... );
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("5aa6300e091641cf7abb58a0")
})
> db.music.find({ title : "SIMON DOMINIC" });
{ "_id" : ObjectId("5aa6300e091641cf7abb58a0"), "title" : "SIMON DOMINIC", "singer" : "Simon D.", "year" : 2015, "genre" : "힙합" }
```

[그림] 위의 질의를 실행한 결과. _id는 자동으로 생성되어 저장 된다.

[출처]

<https://www.tutorialspoint.com/mongodb/index.htm> - MongoDB Tutorial

<https://velopert.com/479> - MongoDB find() 함수

<https://velopert.com/516> - MongoDB cursor를 이용한 함수

<https://velopert.com/545> - MongoDB update 함수