

User & Index

소프트웨어공학과 / 201332001 강인성 / hogu9401@gmail.com

| | |
|--------------------------|-------|
| 1 User 문법 | 2~6 |
| A. User 생성과 권한 | 2~4 |
| B. mongo 보안 서버 접속 | 5~7 |
| C. User 권한 관리 및 제거 | 7~10 |
| 2 Index 문법 | 10~15 |
| A. index 정의 | 10~11 |
| B. index 문법 | 11~15 |

1. User 문법

우리가 Relational Database에서 User를 잠깐 다뤘던 추억이 있을 것이다. 당연히 Relational Database에서는 설치와 동시에 root 계정을 만들어서 초기에 Relational Database를 관리할 수 있도록 유도를 해 주면서 User를 생성하여 각 계정에게 권한을 주거나 걷어 들일 수 있다(Grant / Revoke를 이용). 그렇지만 MongoDB, Redis를 설치하고 난 후에는 계정 초기에는 root 계정 개념이 아예 없기 때문에 3번 스터디 노트까지는 애초에 인증을 하지 않고 무작위로 작동을 하였기에 실제로 User를 통하여 MongoDB에 현존하는 Database들에 대한 권한을 관리를 해줘야 한다. 또한 Spring과 MongoDB를 연동시키기 위해서는 User에 대하여 당연히 정의를 해 줘야 하기 때문에 (물론 클러스터링 작업을 들어 갈 때에도 해당 된다.) 이번 스터디 노트에서는 Index보다 User 문법을 우선적으로 공부를 하는 방향으로 잡았다.

A. User 생성과 권한

우리가 Relational Database에서 쓰인 User와 MongoDB에서 쓰는 User의 개념이 색달라진다. 또한 Relational Database와 MongoDB의 보안성의 차이를 본다면 현저히 Relational Database가 월등한 편이지만 MongoDB에서는 NoSQL의 보안을 향상시키기 위해서는 다행히 User 개념이 존재한다. 초기에 MongoDB를 설치할 때에는 Admin User를 생성하지 않고 바로 넘어가서 mongod 명령어로 그냥 접속을 할 때에는 NoSQL Database의 보안성을 보장하지 않았었다. 그래서 이번 스터디 노트를 토대로 User를 생성해서 데이터베이스를 관리할 수 있는 문장들이 어떻게 작성을 하는가에 대하여 배워 두도록 하자.

(1) User 생성하기

MongoDB에서의 User와 Relational Database의 User의 차이는 바로 데이터베이스 사용을 조정하는 것이다. MongoDB에서 생성된 User(사용자 개념. 관리자와는 다르게 봐야 한다.)는 한 데이터베이스만 접근이 가능하고, Relational Database는 여러 Schema에 대해 무한히 접근 가능하다. 우선 mongo를 작성해서 MongoDB Shell를 실행해두도록 하자.

- 관리자 계정과 권한 추가하기

관리자 계정은 아래와 같은 문장을 이용해서 작성을 하면 된다. 여기서 관리자는 모든 데이터베이스에 대한 접근과 관리가 가능하다. 관리자는 간략하게 adminUser, 비밀번호는 test123으로 작성을 하겠다.

```
> use admin
db.createUser( {
  user : "adminUser",
  pwd : "test123",
  roles : [ "userAdminAnyDatabase", "dbAdminAnyDatabase", "readWriteAnyDatabase" ]
})
```

관리자 계정을 생성하게 되었다는 의미는 권한들을 자세히 살펴보면 모든 Database에 대한 관리와 읽고 쓰는 기능과 각 Database에 대한 권한을 가진 관리자들을 관리할 수 있다는 의미로 생각할 수 있다. 위와 같은 문장을 작성하고 난 후에 다음과 같은 결과가 나오면 성공적으로 추가가 되었다는 증거다.

```
선택 명령 프롬프트 - mongo
Successfully added user: {
  "user" : "adminUser",
  "roles" : [
    "userAdminAnyDatabase",
    "dbAdminAnyDatabase",
    "readWriteAnyDatabase"
  ]
}
> show users
{
  "_id" : "admin.adminUser",
  "user" : "adminUser",
  "db" : "admin",
  "roles" : [
    {
      "role" : "userAdminAnyDatabase",
      "db" : "admin"
    },
    {
      "role" : "dbAdminAnyDatabase",
      "db" : "admin"
    },
    {
      "role" : "readWriteAnyDatabase",
      "db" : "admin"
    }
  ]
}
```

[그림] 관리자 User를 생성한 결과. 위에서도 언급을 했지만 관리자 User와 일반 User의 차이는 어지간히 다르게 생각해야 한다.

- 일반 User 생성하기

일반 User는 admin Database를 이용하는 것이 아닌 우리가 생성한 Database 내부에서 사용을 해 줘야 한다. 그래서 music_test_01라는 Database를 생성해서 권한을 추가해보도록 하자. 참고로 music_test_01 Database는 MongoDB + Spring Data 접목에서도 이용할 예정이다.

```
> use music_test_01
db.createUser( {
  user : "kang",
  pwd : "test123",
  roles : [ "dbAdmin", "readWrite" ]
})
```

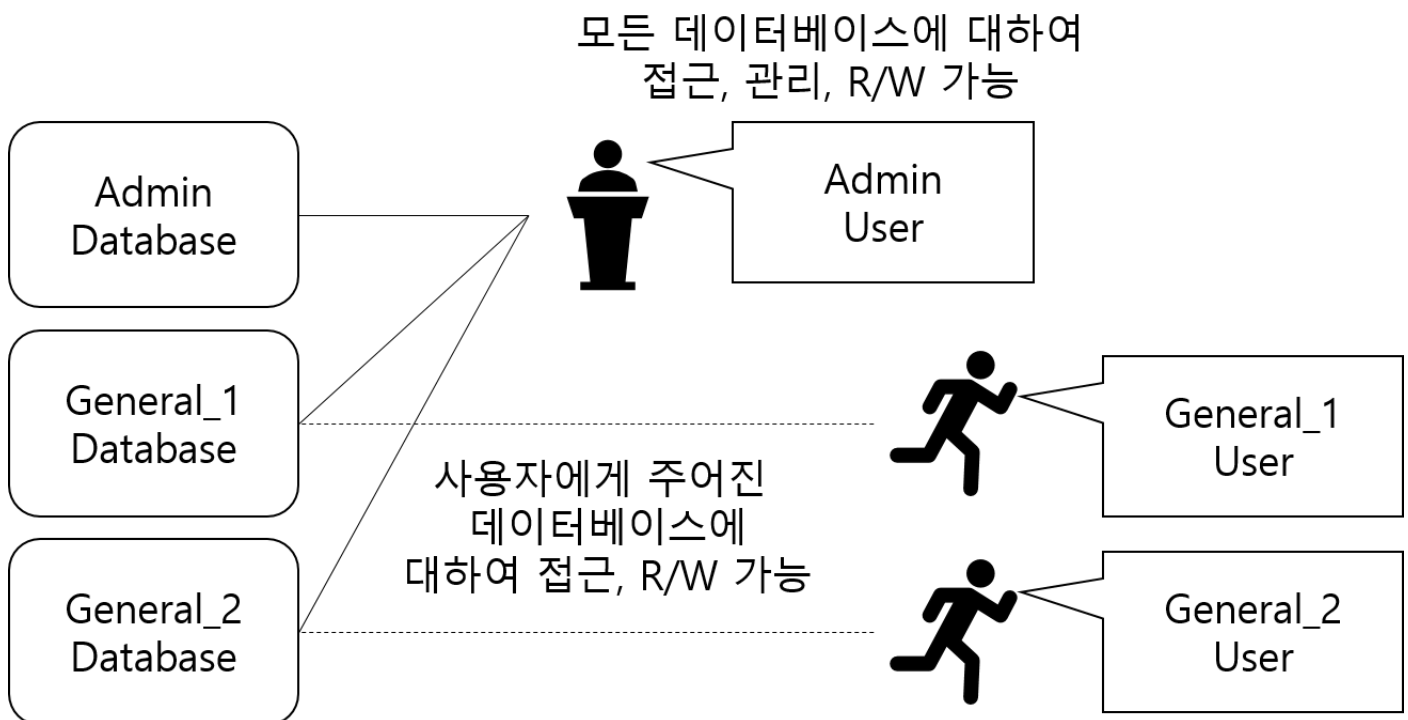
여기서 일반 User는 오로지 한 데이터베이스에 대하여 접근이 가능하도록 구성이 되어 있기 때문에 각 User를 이용해서 접근을 하게 된다면 한 데이터베이스만 읽고 쓰고, collection들을 관리할 수 있게 된다. 방금 위와 같은 문장을 작성하고 난 후에 실행 결과를 확인하게 되면 다음 페이지와 같은 결과를 얻을 수 있다.

➔ 다음 페이지

```
선택 명령 프롬프트 - mongo
> db.createUser({
... user : "kang",
... pwd : "nz973g!!",
... roles : ["dbAdmin", "readWrite"]
... })
Successfully added user: { "user" : "kang", "roles" : [ "dbAdmin", "readWrite" ] }
> show users
{
  "_id" : "music_test_01.kang",
  "user" : "kang",
  "db" : "music_test_01",
  "roles" : [
    {
      "role" : "dbAdmin",
      "db" : "music_test_01"
    },
    {
      "role" : "readWrite",
      "db" : "music_test_01"
    }
  ]
}
```

[그림] 각기 다른 데이터베이스에 사용자를 추가하여 권한을 준 결과. 참고로 show users는 현재 데이터베이스에 권한이 주어진 사용자들을 볼 수 있는 문장이다.

우리가 관리자 User와 일반 User로 나뉘어서 User를 등록하는 예제를 작성해봤다. 이를 토대로 정리를 한다면 아래의 그림과 같이 정리가 가능하다.

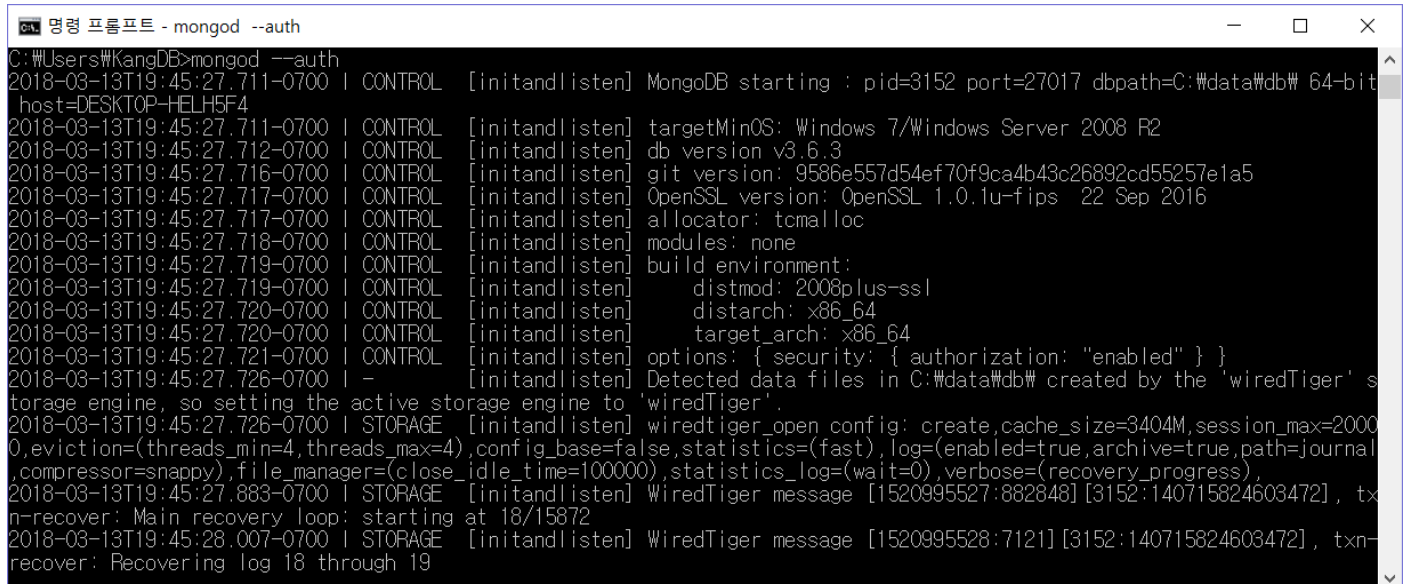


[그림] Admin User & 일반 User 그림으로 작성. 여기서 R/W는 Read And Write로 Document에 대한 CRUD가 가능하다고 이해하면 된다.

B. mongo 보안 서버 접속

우리가 지난 시간 때 까지는 mongod 명령어를 이용해서 서버를 작동했는데 이번에는 권한을 제어할 수 있는 연습을 해 보기 위하여 --auth 를 뒤에 작성해서 접속을 시켜보도록 하자.

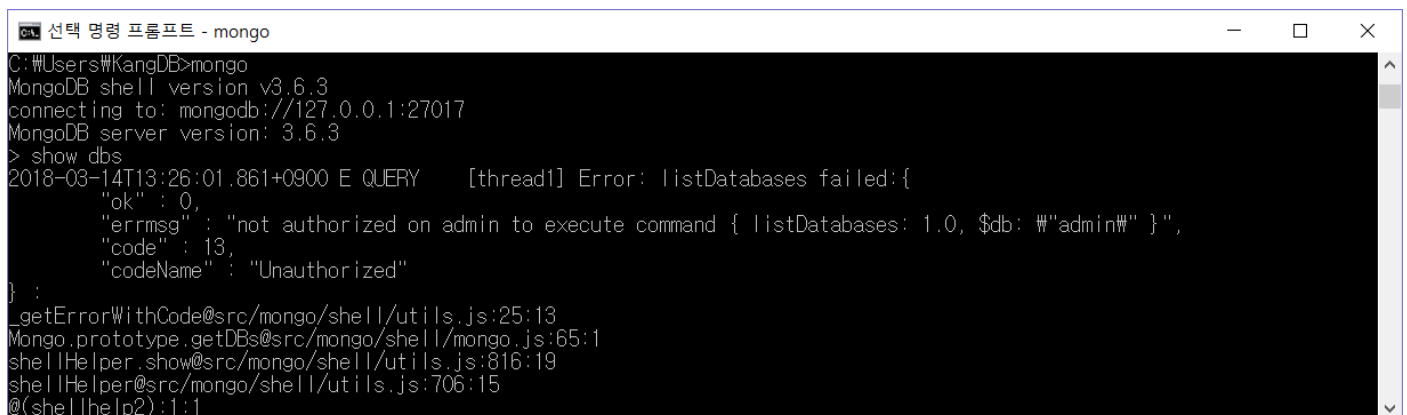
방금 우리가 실행했던 모든 화면들을 ctrl + C를 눌러서 종료 시키고 mongod 명령어 대신 mongod --auth를 이용해서 다시 실행해보도록 하자.



```
명령 프롬프트 - mongod --auth
C:\Users\KangDB>mongod --auth
2018-03-13T19:45:27.711-0700 | CONTROL | [initandlisten] MongoDB starting : pid=3152 port=27017 dbpath=C:\data\db\ 64-bit
host=DESKTOP-HELH5F4
2018-03-13T19:45:27.711-0700 | CONTROL | [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-03-13T19:45:27.712-0700 | CONTROL | [initandlisten] db version v3.6.3
2018-03-13T19:45:27.716-0700 | CONTROL | [initandlisten] git version: 9586e557d54ef70f9ca4b43c26892cd55257e1a5
2018-03-13T19:45:27.717-0700 | CONTROL | [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips 22 Sep 2016
2018-03-13T19:45:27.717-0700 | CONTROL | [initandlisten] allocator: tcmalloc
2018-03-13T19:45:27.718-0700 | CONTROL | [initandlisten] modules: none
2018-03-13T19:45:27.719-0700 | CONTROL | [initandlisten] build environment:
2018-03-13T19:45:27.719-0700 | CONTROL | [initandlisten] distmod: 2008plus-ssl
2018-03-13T19:45:27.720-0700 | CONTROL | [initandlisten] distarch: x86_64
2018-03-13T19:45:27.720-0700 | CONTROL | [initandlisten] target_arch: x86_64
2018-03-13T19:45:27.721-0700 | CONTROL | [initandlisten] options: { security: { authorization: "enabled" } }
2018-03-13T19:45:27.726-0700 | - | [initandlisten] Detected data files in C:\data\db\ created by the 'wiredTiger' s
storage engine, so setting the active storage engine to 'wiredTiger'.
2018-03-13T19:45:27.726-0700 | STORAGE | [initandlisten] wiredtiger_open config: create,cache_size=3404M,session_max=2000
0,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal
,compressor=snappy),file_manager=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress),
2018-03-13T19:45:27.883-0700 | STORAGE | [initandlisten] WiredTiger message [1520995527:882848] [3152:140715824603472], tx
n-recover: Main recovery loop: starting at 18/15872
2018-03-13T19:45:28.007-0700 | STORAGE | [initandlisten] WiredTiger message [1520995528:7121] [3152:140715824603472], txn
recover: Recovering log 18 through 19
```

[그림] mongod --auth 명령어를 통한 실행 결과. 일반적으로 작동하는 mongod와는 차이가 없어 보이지만 실제로 mongo 명령어를 통해 Shell를 실행하게 되면 입장이 달라지게 된다.

이처럼 실행을 같이 해 둔 상태로 mongo Shell를 실행하고 작동을 해 보도록 하겠다.



```
선택 명령 프롬프트 - mongo
C:\Users\KangDB>mongo
MongoDB shell version v3.6.3
connecting to: mongod://127.0.0.1:27017
MongoDB server version: 3.6.3
> show dbs
2018-03-14T13:26:01.861+0900 E QUERY [thread1] Error: listDatabases failed: {
  "ok" : 0,
  "errmsg" : "not authorized on admin to execute command { listDatabases: 1.0, $db: \"admin\" }",
  "code" : 13,
  "codeName" : "Unauthorized"
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:65:1
shellHelper.show@src/mongo/shell/utils.js:816:19
shellHelper@src/mongo/shell/utils.js:706:15
@(shellHelp2):1:1
```

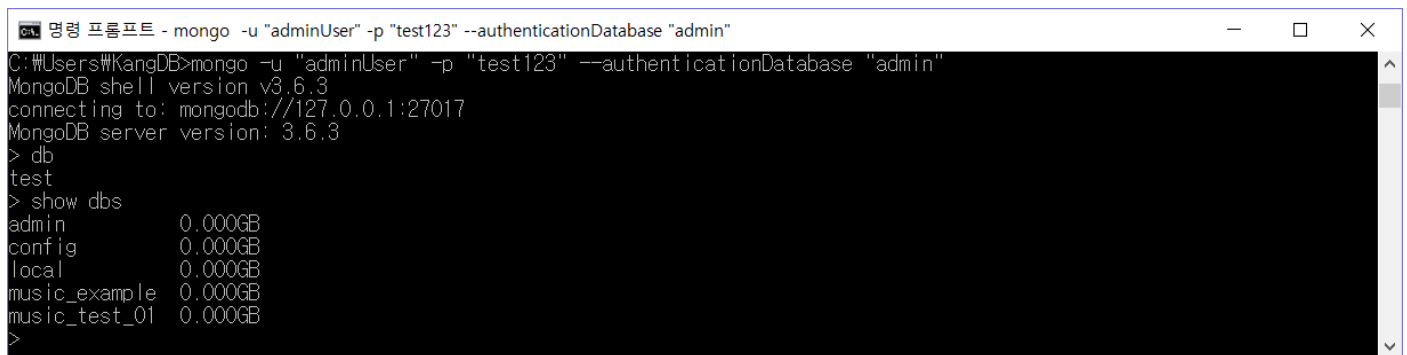
[그림] mongo 명령어를 실행하고 난 후에 보안 모드로 초기 접속 시 show dbs 명령어를 작성한 결과. 계정 확인이 되지 않았다는 에러를 출력하게 된다.

mongod --auth를 이용해서 mongo Shell를 실행하게 된다면 분명히 누군가의 계정을 확인해야 할 수단이 있어야 한다. 그래서 초반에 show dbs 문장을 이용해서 간략하게 데이터베이스 목록들을 확인하게 된다면 이러한 결과가 나오게 된다. 그래서 show dbs 문장을 실행하기 위해서는 admin(관리자) 계정으로 로그인을 하고 난 후에 실행을 해야 한다. 이는 아래와 같이 로그인해서 작성할 수 있다.

```
> use admin
db.auth("adminUser", "test123")
```

참고로 admin Database에는 관리자 User들이 들어가게 되어서 모든 Database에 대한 권한을 관리할 수 있다. 또한 관리자 User를 통해서 현재 Database의 목록들을 확인할 수 있다. 그렇지만 일반 User로 로그인을 한다면 하나의 데이터베이스만 이용이 가능하기 때문에 데이터베이스의 목록을 확인할 수 있는 방법이 없다. 또한 admin 테이블을 찍지 않고 mongo 명령어를 이용해서 로그인을 할 수 있는 방법도 있다. 아래 문장과 같이 작성하도록 하자.

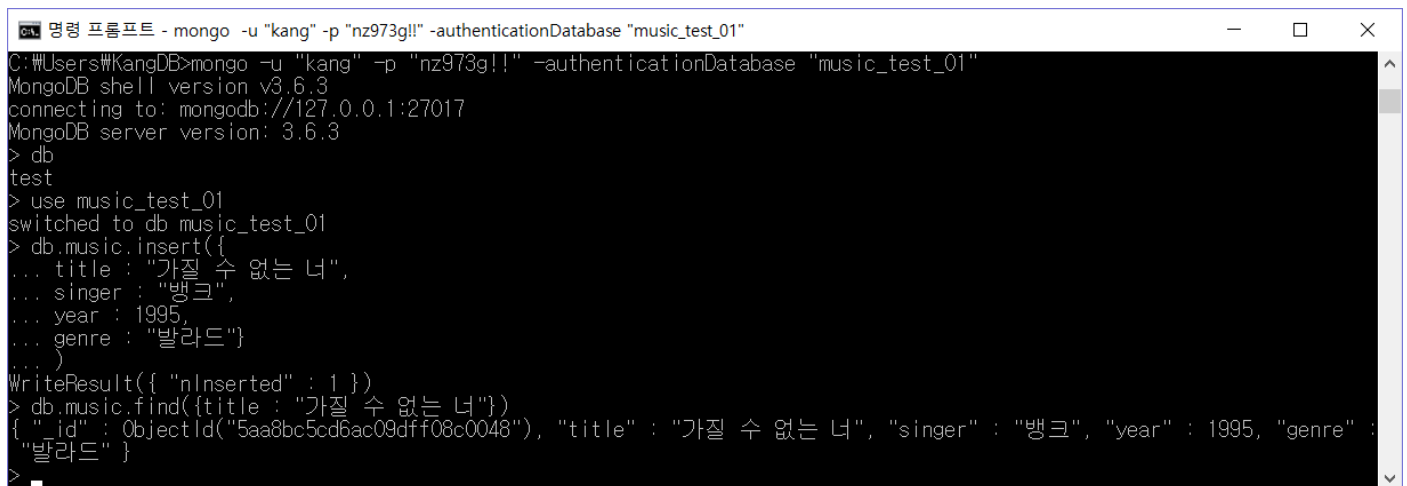
```
> mongo -u "adminUser" -p "test123" --authenticationDatabase "admin"
[ mongo -u <username> -p <password> -authenticationDatabase <database> ]
```



```
명령 프롬프트 - mongo -u "adminUser" -p "test123" --authenticationDatabase "admin"
C:\Users\KangDB>mongo -u "adminUser" -p "test123" --authenticationDatabase "admin"
MongoDB shell version v3.6.3
connecting to: mongod://127.0.0.1:27017
MongoDB server version: 3.6.3
> db
test
> show dbs
admin            0.000GB
config           0.000GB
local            0.000GB
music_example    0.000GB
music_test_01    0.000GB
>
```

[그림] 위의 문장을 토대로 실행을 한 결과. 당연히 show dbs 명령어를 입력하면 결과가 올바르게 나온다.

물론 관리자 User 뿐만 아니라 일반 User도 이처럼 접속을 할 수 있다. 다만 일반 User는 권한이 주어진 Database만 조작이 가능할 뿐이다.



```
명령 프롬프트 - mongo -u "kang" -p "nz973g!!" -authenticationDatabase "music_test_01"
C:\Users\KangDB>mongo -u "kang" -p "nz973g!!" -authenticationDatabase "music_test_01"
MongoDB shell version v3.6.3
connecting to: mongod://127.0.0.1:27017
MongoDB server version: 3.6.3
> db
test
> use music_test_01
switched to db music_test_01
> db.music.insert({
... title : "가질 수 없는 너",
... singer : "뱅크",
... year : 1995,
... genre : "발라드"
... })
WriteResult({ "nInserted" : 1 })
> db.music.find({title : "가질 수 없는 너"})
{ "_id" : ObjectId("5aa8bc5cd6ac09dff08c0048"), "title" : "가질 수 없는 너", "singer" : "뱅크", "year" : 1995, "genre" : "발라드" }
>
```

[그림] 일반 User로도 접속이 가능하다. 또한 readWrite 권한을 가졌으면 당연히 Document를 수정할 수 있고, 조회도 가능하다.

그리고 사용자가 로그인 성공이 완료되면 다시 Logout을 해 줘야 하는데 Ctrl + C 버튼을 이용해서 mongo Shell를 종료하는 방법도 있지만 db.logout() 함수를 이용해서 종료를 시켜주고 다른 계정으로 로그인 하는 경우 해당되는 Database를 가리키고 난 후에 db.auth(<username>, <password>)를 이용해서 로그인이 가능하다.

C. User 권한 관리 및 제거

MongoDB에서 쓰는 User의 개념을 이제 점차 이해했을 것이다. 이번에는 각 User들에 대해서 권한을 조정하는 연습을 해 보도록 하겠다. 우선 각 User들에게 줄 수 있는 권한들은 아래와 같이 나뉠 수 있다.

(1) MongoDB에서 현재 제공하고 있는 권한 목록들

- Document에 대해 쓰는 권한

read : Document들을 오로지 읽을 수만 있는 권한만 주어짐.

readWrite : Document들을 읽을 수 있는 동시에 삽입, 삭제, 수정이 가능하다.

- 하나의 Database에 대해 쓰는 권한

dbAdmin : 현재 데이터베이스에 대한 Schema 작업과 Index 생성과 통계 관련 작업을 실행 할 수 있다. 그렇지만 타 User들에게 권한을 줄 수 있지는 않다. 그런데 이 작업만 주었다고 하더라도 Document들을 읽고 쓸 수는 없는 일이다.

dbOwner : 이는 데이터베이스에 대한 모든 관리를 할 수 있도록 하는 권한이다. 이 권한은 dbAdmin, readWrite와 결합한 개념과 마찬가지로인 셈이다.

userAdmin : 각기 다른 일반 User들에게 권한을 주거나 걷어 들일 수 있도록 한다.

- 모든 Database에 대해 쓰는 권한

여기에 있는 모든 권한은 local, config Database에 대해서는 해당되지 않는다.

readAnyDatabase : 말 그대로 모든 Database에 있는 Document들을 읽을 수만 있는 권한이다.

readWriteAnyDatabase : 모든 Database에 있는 Document들을 읽고 쓸 수 있는 권한이다.

userAdminAnyDatabase : 모든 Database에 현존하는 User들에 대해서 권한을 주고 뺏을 수 있는 권한이다.

dbAdminAnyDatabase : 모든 Database에 대해 Index를 생성하거나 통계 관련 작업을 할 수 있는 권한이다.

- Clustering 관리에 대한 권한

참고로 NoSQL Clustering 작업에 대해서는 06번 스터디 노트에서 다룰 예정이다. 이러한 권한들이 존재한다는 것만 인지하고 넘어가자.

clusterAdmin : 클러스터 관리를 할 수 있는 User 중 최고봉. 이는 clusterManager, clusterMonitor, hostManager에 대한 권한을 줄 수 있고 데이터베이스를 삭제하는(dropDatabase()) 권한도 주어진다.

clusterManager : 클러스터에서 실시간으로 확인되는 모니터링을 관리할 수 있고 local, config 데이터베이스에

접근이 가능하다.

clusterMonitor : 모니터링을 관리할 수 있는 수준은 아니고 어떻게 돌아가는지에 대해 읽어 들일 수만 있다.

hostManager : 모니터링과 서버를 공급할 수 있도록 하는 권한이다.

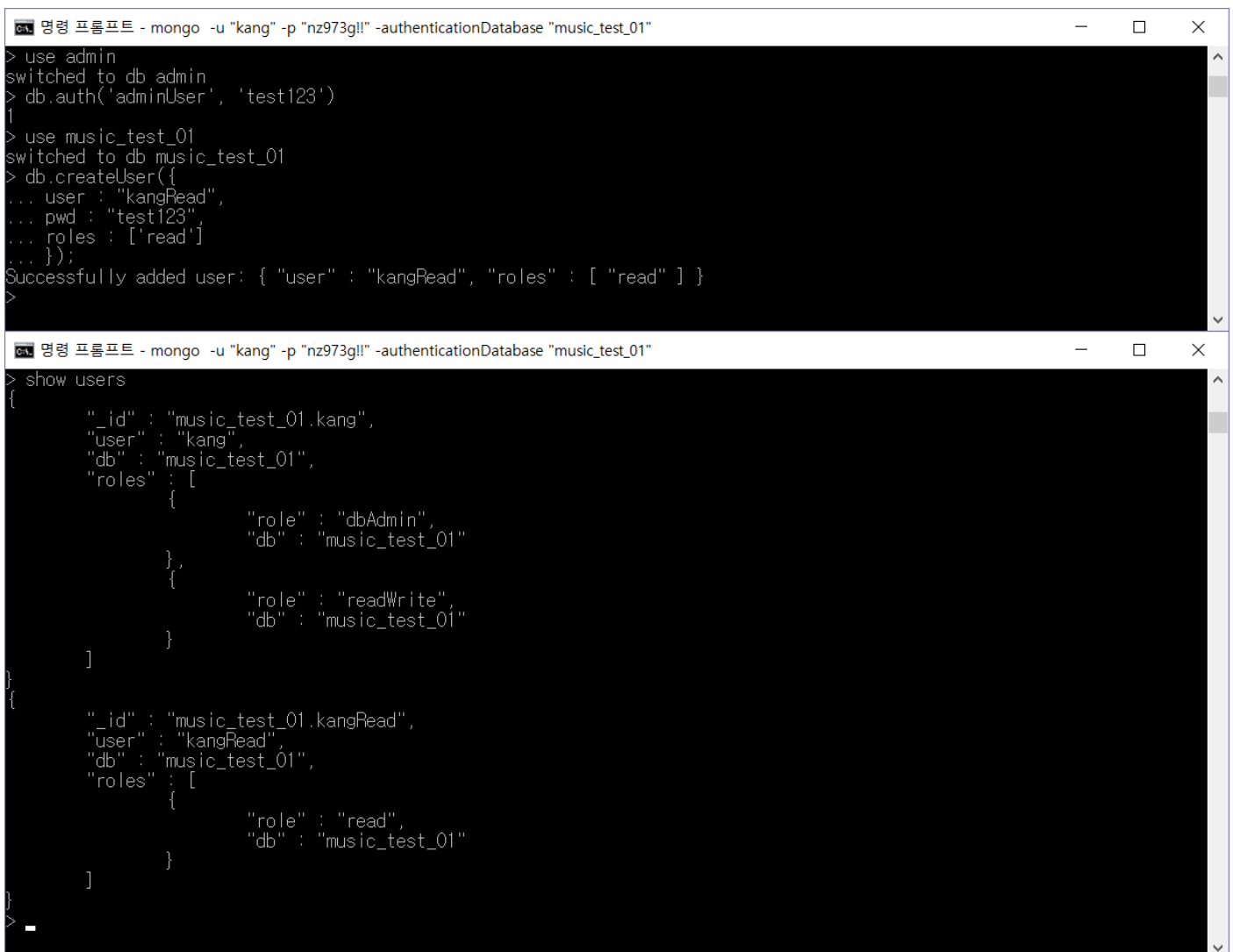
- 백업과 복원 관련 권한

backup : 이는 현재 Database에 적재된 데이터들에 대해 mongodump를 이용해서 백업을 시켜줄 수 있는 권한이다.

restore : 이는 이전에 작성했던 Database의 시점을 설정하여 mongorestore를 토대로 가져와서 복원을 할 수 있는 권한이다.

(2) User 권한 관리

우선 권한을 관리할 수 있는 User인 adminUser를 이용해서 접속을 해 보고 music_test_01 데이터베이스에 kangRead User를 새로 추가하겠다.



```
명령 프롬프트 - mongo -u "kang" -p "nz973g!!" -authenticationDatabase "music_test_01"
> use admin
switched to db admin
> db.auth('adminUser', 'test123')
1
> use music_test_01
switched to db music_test_01
> db.createUser({
... user : "kangRead",
... pwd : "test123",
... roles : ['read']
... });
Successfully added user: { "user" : "kangRead", "roles" : [ "read" ] }
>

명령 프롬프트 - mongo -u "kang" -p "nz973g!!" -authenticationDatabase "music_test_01"
> show users
{
  "_id" : "music_test_01.kang",
  "user" : "kang",
  "db" : "music_test_01",
  "roles" : [
    {
      "role" : "dbAdmin",
      "db" : "music_test_01"
    },
    {
      "role" : "readWrite",
      "db" : "music_test_01"
    }
  ]
}
{
  "_id" : "music_test_01.kangRead",
  "user" : "kangRead",
  "db" : "music_test_01",
  "roles" : [
    {
      "role" : "read",
      "db" : "music_test_01"
    }
  ]
}
>
```

[그림] kangRead User를 새롭게 추가하였지만 권한은 Document 목록들을 읽어 들이기만 가능하도록 설정하였다.

그러면 현재 adminUser를 이용해서 kangRead에게도 readWrite 권한을 줄 수 있는 문장은 아래와 같이 작성할 수 있다.

```
> db.updateUser("kangRead", {roles : ['readWrite']})
```



```
> db.updateUser("kangRead", {roles : ['readWrite']})
> show users
{
  "_id" : "music_test_01.kang",
  "user" : "kang",
  "db" : "music_test_01",
  "roles" : [
    {
      "role" : "dbAdmin",
      "db" : "music_test_01"
    },
    {
      "role" : "readWrite",
      "db" : "music_test_01"
    }
  ]
}
{
  "_id" : "music_test_01.kangRead",
  "user" : "kangRead",
  "db" : "music_test_01",
  "roles" : [
    {
      "role" : "readWrite",
      "db" : "music_test_01"
    }
  ]
}
```

[그림] kangRead User에게 새로운 권한을 주고 난 후에 나온 문장. 이제 kangRead User는 Document들을 추가, 삭제, 수정할 수 있다.

다만 권한 부여에 대한 수정은 오로지 관리자 User나 이 데이터베이스를 총괄하는 User만 수정이 가능하다. 권한에 대해 Grant, Revoke를 하는 문장은 간략하게 아래와 같은 문장으로 작성을 해서 쓰면 된다.

db.grantRolesToUser([사용자], [권한 목록들]), db.revokeRolesToUser([사용자], [권한 목록들])

그렇지만 각 사용자가 비밀번호를 변경하는 경우를 생각해봐야 할 때가 있다. 비밀번호나 사용자 정보를 변경한다는 권한은 존재하지 않을 뿐더러 모든 사용자들에게 제공이 되어 있기 때문에 현재 사용자에서 로그아웃을 하고 kangRead로 로그인을 해서 비밀번호를 변경하는 문장을 작성할 수 있는데 이는 아래의 화면을 참고해서 작성해서 쓰면 되겠다.

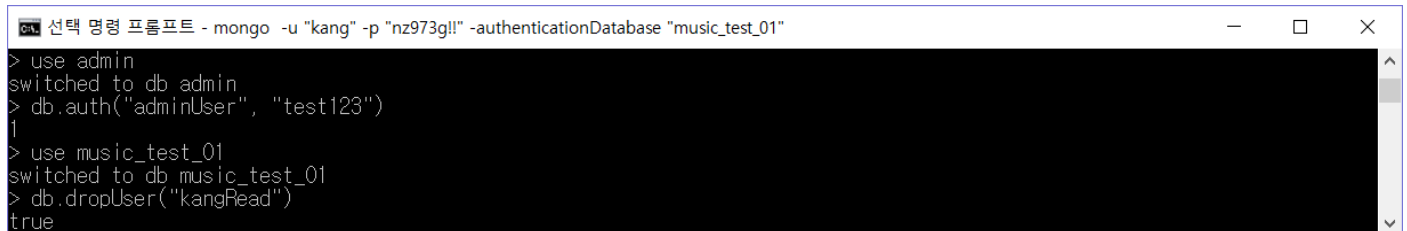
```
> db.auth("kangRead", "test123")
1
> db
music_test_01
> db.updateUser("kangRead", {pwd : "test1234"});
> db.logout()
{ "ok" : 1 }
> db.auth("kangRead", "test123");
Error: Authentication failed.
0
> db.auth("kangRead", "test1234")
1
>
```

[그림] 로그인 후 비밀번호를 바꾸고 다시 로그인을 한 결과. 비밀번호가 정상적으로 수정되었다.

(3) 사용자 삭제

사용자를 삭제할 수 있는 권한도 마찬가지로 관리자 User만이 삭제할 수 있다. 사용자 삭제 문장은 의외로 간단하다. 다만 사용이 가능한 User는 userAdmin 권한이 주어진 관리자 User만 가능하다. 방금 전에 생성된 kangRead 사용자에게 대해서 삭제를 하는 문장은 다음과 같다.

```
> db.dropUser("kangRead")  
[ db.dropUser(<username>) ]
```



```
선택 명령 프롬프트 - mongo -u "kang" -p "nz973g!!" -authenticationDatabase "music_test_01"  
> use admin  
switched to db admin  
> db.auth("adminUser", "test123")  
1  
> use music_test_01  
switched to db music_test_01  
> db.dropUser("kangRead")  
true
```

[그림] 방금 문장을 토대로 실행한 결과. 정상적으로 kangRead User가 삭제되었다.

2. Index 문법

방금 전까지 MongoDB에서 User를 어떻게 생성하고 삭제하는지에 대해 익혔다. 그러면 이번에는 dbAdmin 권한이 주어진 User들에 대해서 Index를 생성하고 삭제하는 방안에 대해서 공부를 해 보도록 하겠다.

A. Index의 정의

Relational Database에서도 Index를 이용해서 Record 별 데이터들을 정리를 하는 사실은 이미 알고 있는 사실이다. 그렇지만 MongoDB에서 쓰이는 Index는 무엇일까? Relational Database와 마찬가지로 탐색 속도를 향상시키는 목적으로 구성되어 B-Tree를 이용해서 구현을 한 개념으로 이해할 수 있다. B-Tree 알고리즘을 모른다면 복습을 하고 돌아오길 바란다. MongoDB에서는 Database 내부에 있는 Collection들을 이용해서 Document들을 조회를 하는데 Collection들이 점차 많이 늘어나게 된다면 Document들의 검색 속도 향상을 시키는 방법을 구상해야 할 때가 찾아 온다. 예를 들어 음악이라고 치면 노래 제목을 이용해서 탐색을 하는 경우가 대다수이고, 가수 이름을 통해서 검색하는 경우도 종종 있다.

어느 데이터베이스 개념을 다룰지라도 Index 개념은 꼭 인지를 하고 넘어가야 한다. 비록 작은 Index라 생각하겠지만 언젠가 조회를 하는데 있어서 하드웨어 부분에서 성능을 향상시키지 못 하더라도 데이터베이스에서의 검색 효율을 그나마 높여주도록 유도할 수 있기 때문이다.

(1) MongoDB Index의 종류

Index의 종류는 크게 7가지로 나뉘게 된다. Relational Database에서 다뤘던 Index에서는 오로지 B-Tree를 기반으로 Index를 생성을 하였지만(더욱 많아지는 데이터들에 대해서는 Bitmap Index까지 고려할 수도 있지만...), Hashed Index가 새롭게 추가되어 기존의 B-Tree Index의 단점을 향상시키는 역할도 한다.

- `_id` Index : 우리가 Collection을 최초로 만들 때 자동적으로 `_id` Field(RDB에서 Primary와 같은 격)을 이용해서 Index를 생성시켜 준다. 당연히 `_id`의 값들은 Unique 제약 조건과 Not Null 제약 조건을 지니기 때문에 데이터를 새로 추가하거나 삭제할 때 지장이 없게 된다.
- Single Field Index : `_id` Index를 이용하지 않는 대신에 Collection 내부에 있는 다른 Field 1개를 이용해서 검색 속도를 향상 시켜주는 역할을 한다. 예를 들어 우리가 작성한 music Collection에서는 과반수로 음악 제목을 이용해서 검색을 하는 경우 음악 제목에 대해 정렬 작업을 진행한 후(정렬 작업은 오름차순, 내림차순 둘 중 하나)에 검색어를 따라서 탐색을 할 수 있도록 유도를 해 준다.
- Compound Field Index : 이는 Field를 여러 개를 이용해서 검색 속도를 향상 시켜주는 역할을 한다. 예를 들어 music Collection에서 음악 제목이 동일한 경우 가수 이름까지 검색을 하는 경우에 각 음악 제목과 가수 이름에 대한 정렬 작업을 진행하고 사용자가 작성한 Query에 대하여 탐색을 하도록 유도한다.
- Multikey Index : 우리가 MongoDB에서도 배열이 데이터 타입에 추가가 가능하다는 사실은 알고 있는데 여기서는 기초를 다지기 위해서 사례를 아직 작성하지 않았다. 이에 대해서는 추후에 프로젝트를 진행하면서 필요하면 그 때 정리해서 요약 작업을 할 예정이다. 배열로 구성된 Field에 대한 Index를 작성할 때 활용을 한다.
- Geospatial Index : 우리가 지도에서 좌표를 받아와서 데이터를 저장하거나 마인크래프트처럼 위치 기반으로 좌표 데이터를 주고 받는 게임을 할 때 등등 좌표나 수치 변화(dx, dy 등을 이용한 계산 처리)에 대한 데이터를 효율적으로 탐색하도록 유도하는 Index이다.
- Text Index : 짧은 String 문장을 다루는 것이 아닌 게시글, 뉴스, 타임라인 등 긴 글에 대하여 검색 효율을 높여주는 Index이다.
- Hashed Index : 위에서 설명한 6개에서는 B-Tree 자료구조를 이용하는 반면에 말 그대로 Hash 자료구조를 사용해서 검색 효율을 B-Tree보다 향상을 시켜주는 역할을 하지만 알고리즘에서 이미 배웠듯이 Hash는 정렬을 하지 않고 오로지 데이터에 대한 Hash 값으로 탐색 속도만 향상시키는 역할만 해 준다.

B. Index 문법

Index 문법은 Relational Database와 마찬가지로 단순한 편이다. 물론 Index를 생성하기 알맞은 조건을 복습하자면 검색하는 Field들이 많은 경우, Field들 중에서 변경이 되긴 하지만 빈번한 경우 등에 적합하지만 너무 많은 Index를 생성하는 방안은 Index의 용량 낭비와 더욱 복잡해지는 B-Tree의 구조가 나올 수도 있는 부적절한 상황이 올 수 있기 때문에 Index가 어떤 면에서는 만병 통치약이 아니라는 뜻이다. 여기서는 MongoDB에서 Index를 생성하는 문장을 다뤄보는 연습을 해 보도록 하겠다. 위에서 다룬 개념들 중에서 Single Field Index, Compound Field Index 2개에 대해서만 다뤄보도록 하겠다. `_id` Index는 자동적으로 생성되는 Index이니 굳이 설명하지 않고 넘어가겠다.

➔ Next Page

(1) Single Field Index 생성

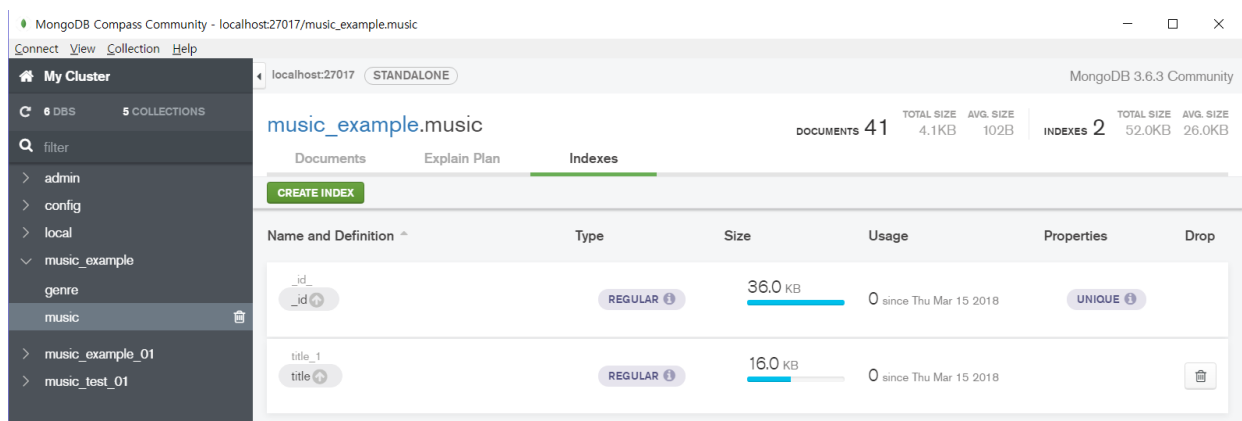
우리가 현재까지 작성했던 music_example를 기반으로 Index를 생성하는 문장을 작성을 해보도록 하겠다. 단일 Field에 대한 Index를 생성하는 문장은 아래와 같이 작성할 수 있다.

```
> db.컬렉션_이름.createIndex({ Field_이름 : 1 / -1 }) // 1은 오름차순, -1는 내림차순이다.
```



```
명령 프롬프트 - mongo
> use music_example
switched to db music_example
> db.music.createIndex({title : 1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

[그림] music_example 내부에 있는 collection인 music에 대한 Index를 형성한 결과

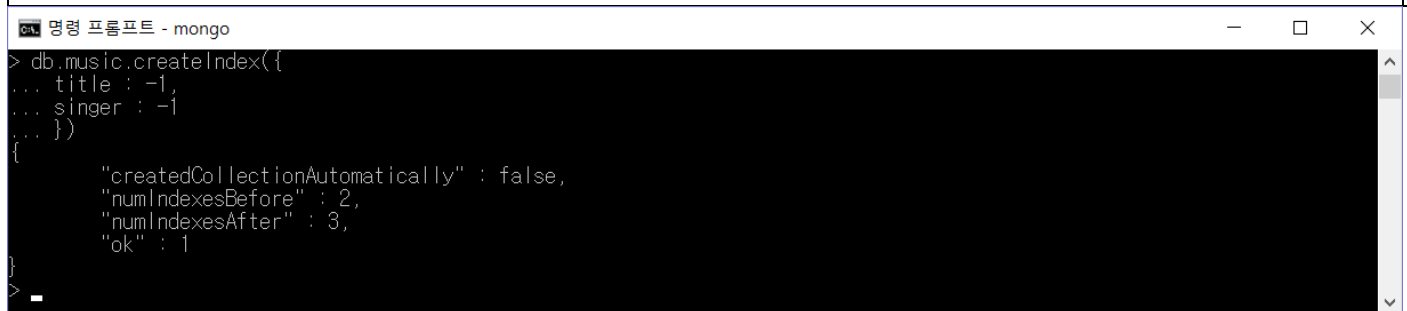


[그림] music_example의 music Collection 내부에 생성된 title 관련 Index를 생성한 결과

(2) Compound Field Index

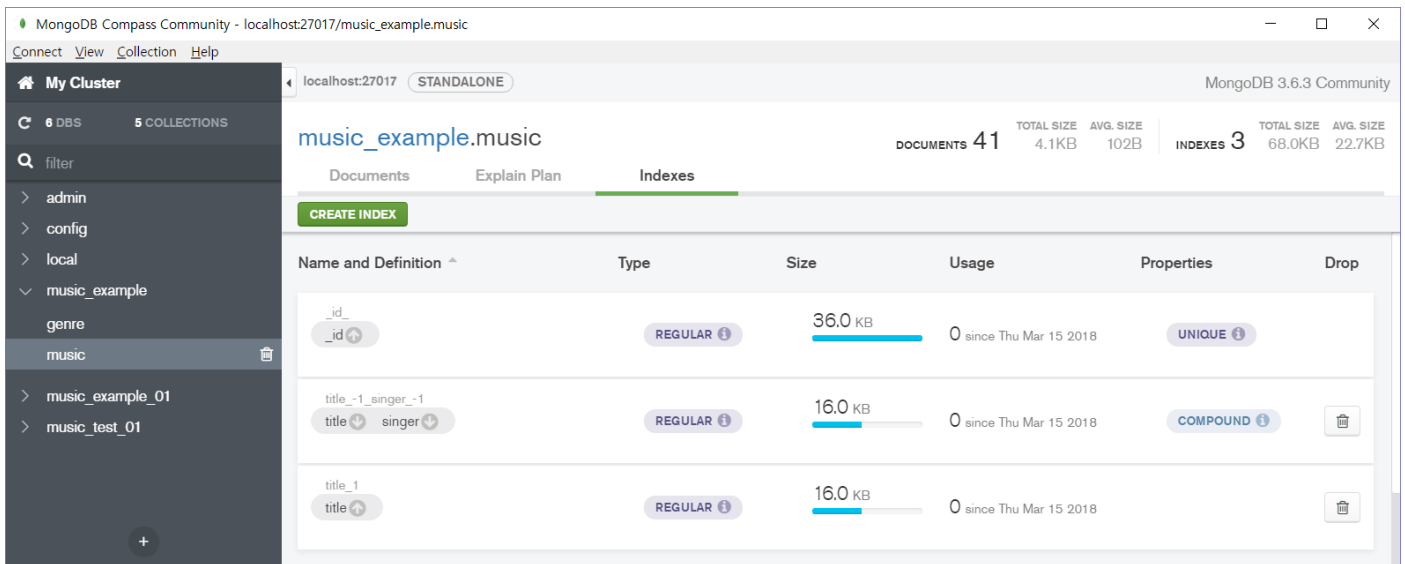
이번에는 음악 제목과 가수 이름을 기준으로 한 Index를 생성하는 문장을 작성해보도록 하겠다. 우리가 작성한 Music Collection에 있는 Field들은 제목, 가수, 연도, 장르 4가지로 나뉘 수 있는데 음악이 새로 만들어지는 경우에는 제목과 장르는 변경이 되지 않을 것이다. 그렇지만 가수 이름은 예명이 변경되는 경우(예명 변경은 그나마 드물다.)를 고려할 때를 제외하고는 Index를 생성하는데 지장은 없다. 그래서 이번에는 음반 제목과 가수 이름을 각각 내림차순을 이용한 탐색에 효율을 높여보도록 하겠다.

```
> db.컬렉션_이름.createIndex({ Field_이름 : 1 / -1, ... }) // 1은 오름차순, -1는 내림차순이다.
```



```
명령 프롬프트 - mongo
> db.music.createIndex({
... title : -1,
... singer : -1
... })
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
```

[그림] music Collection에 Compound Field Index를 생성한 결과



[그림] 음악 제목과 가수 이름을 각각 내림차순으로 한 Index를 생성한 결과. Field의 수가 여러 개가 될 수 있을 지라도 Index의 용량은 오로지 16KB가 된다.

(3) Index에 속성 추가하기

우리가 Relational Database에서도 속성을 추가를 해주면서 Index를 생성을 했었다. 예를 들어 Where 문을 작성하거나 유일한 값(Distinct)에 대하여 정리를 해서 썼던 기억이 날 것이다. 그래서 여기서는 Index에 대해서 속성을 추가하는 문장을 일부 작성을 해 보도록 하겠다. MongoDB에서도 마찬가지로 Unique, Query를 통한 부분 Index를 생성할 수 있다. 아래의 문장을 참고해보도록 하자.

- Unique 속성 추가하기

어느 데이터베이스를 접근을 하더라도 Unique 제약 조건은 항상 존재하기 마련이다. MongoDB에서도 Unique 제약 조건을 추가할 수 있긴 하지만 Index를 통해 설정을 해야 한다. 예를 들어 음악 제목을 이용해서 Index를 생성하는 문장은 아래와 같이 작성을 할 수 있다. 그리고 방금 전에 만든 음악 제목 Index를 삭제하고 진행을 해야 생성이 가능하다.

```

명령 프롬프트 - mongo
> use music_example
switched to db music_example
> db.music.createIndex({title : 1}, {unique : true});
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}

```

[그림] music_example Database에 있는 music Collection에 title Index를 새로 추가하였음. 이를 추가할 시에는 title은 언제나 유일할 수 밖에 없다.

```
선택 명령 프롬프트 - mongo
> db.music.insert({
... title : "To Heaven",
... singer : "조성모",
... year : 1998,
... genre : "발라드"}
... )
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: music_example.music index: title_1 dup key: { : #\"To
Heaven#\" }"
  }
})
```

[그림] music Collection 내부에서 노래 제목이 To Heaven인 Document를 추가하는 경우에 방금 전에 생성된 Index로 인하여 추가를 하지 못하게 된다.

그렇지만 실제로 노래 제목이 같은 가수도 존재할 수가 있다. 예를 들어 현재 music Collection에 저장된 성시경의 거리에서도 있고, 김광석의 거리에서도 존재할 수도 있다 이런 이야기이다. 이를 방지하기 위해 방금 전에 생성한 title Index와 compound Index를 모두 삭제하고 난 후에 아래와 같은 Index로 대체하도록 하겠다.

```
명령 프롬프트 - mongo
> db.music.createIndex(
... {title : 1, singer : 1},
... {unique : true}
... );
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.music.insert({
... title : '거리에서',
... singer : '김광석',
... year : 1990,
... genre : '발라드'
... })
WriteResult({ "nInserted" : 1 })
>
```

[그림] music Collection에 음악 제목과 가수 이름을 토대로 한 Index와 unique를 설정을 하면 노래 제목이 같은 가수가 존재하더라도 추가하는데 지장이 없어지게 된다.

- Partial Index(부분 Index)

또한 Index를 추가하는 경우에는 특정 수치를 기반으로 한 Index도 존재할 수 있다. 예를 들어 1990년대 음악들에 대해서 Query를 생성하여 부분적으로 검색을 효율적으로 할 수 있는 Index를 생성하면 아래와 같이 이용할 수 있다.

```
명령 프롬프트 - mongo
> db.music.createIndex(
... {title : -1},
... {partialFilterExpression : {year : { $gte : 1990, $lte : 1999}}}
... );
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
>
```

방금 전 사진처럼 Index를 생성을 한다면 1990년대 노래($1990 \leq \text{year} \leq 1999$)에 대해서 노래 제목을 내림차순으로 탐색을 할 때 쓰는 Index로 이용을 하게 된다.

- TTL Index

Index를 생성할 때 Date 타입, Date 배열 타입에 대해서 적용을 할 때 이용을 하는 Index로 볼 수 있는데 이 Index를 이용한다면 몇 시간이 지나고 난 후에 Document를 만료를 시킬 수 있는 기능으로 볼 수 있다. Music Collection에는 Date 타입이 존재하지 않지만, 여기서는 간단한 사례로 JWT Token을 받아올 때에 Token의 유효시간이 만료 된다면 없어질 때 쓰거나 사용자에게 제공하는 Alert 기능 등에 좋은 Index로 이해할 수 있다.

```
> db.notifications.createIndex( { "notifiedDate": 1 }, { expireAfterSeconds: 3600 } )
db.컬렉션_이름.createIndex({ Field 구성. Date Field를 기반으로 }, { expireAfterSeconds : [ n ]})
n은 초 단위로 볼 수 있다.
```

(4) Index 조회와 삭제하기

Index 조회는 아래와 같은 명령어를 이용해서 조회하면 된다. Indexs가 아니라 Indexes이다.

```
> db.music.getIndexes();
db.컬렉션_이름.getIndexes();
```

Index 삭제는 아래와 같이 방금 전에 추가했던 Index를 토대로 삭제가 가능하다. 다만 _id Index는 Collection을 제거하지 않는 이상 삭제가 불가능하다는 점을 명심해야 한다.

```
> db.music.dropIndex({ title : -1 }, {partialFilterExpression : { year : { $gte : 1990, $lte : 1999}}});
db.컬렉션_이름.dropIndex({ Field 속성들 }, { 기타 속성들 })
```

[출처]

1. User 문법

- A. <http://blog.freezner.com/archives/1040> - 관리자 계정 생성 개념
- B. <http://html5around.com/wordpress/tutorials/mongodb-%EC%9D%B8%EC%A6%9D%EC%84%A4%EC%A0%95%ED%95%98%EA%B8%B0/> - MongoDB User 인증 접속 설명
- C. <https://docs.mongodb.com/manual/core/security-built-in-roles/> - MongoDB 권한에 대한 설명

2. Index 문법

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

<https://velopert.com/560> - MongoDB Index 문법