타이포라로 작성한 문서

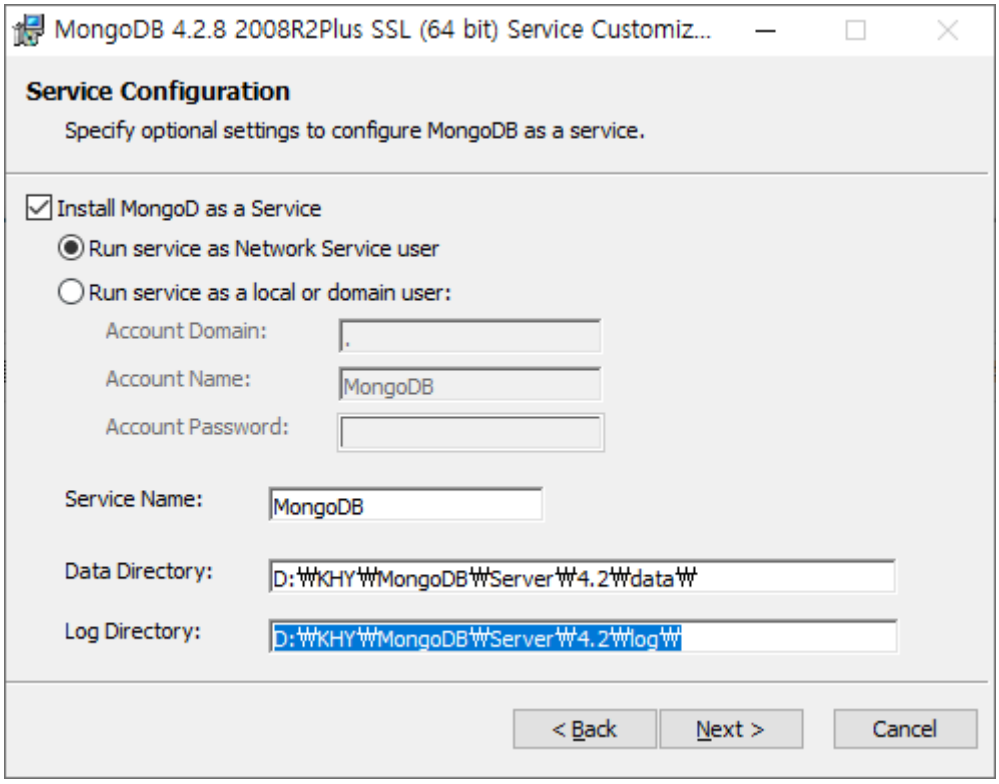
시험

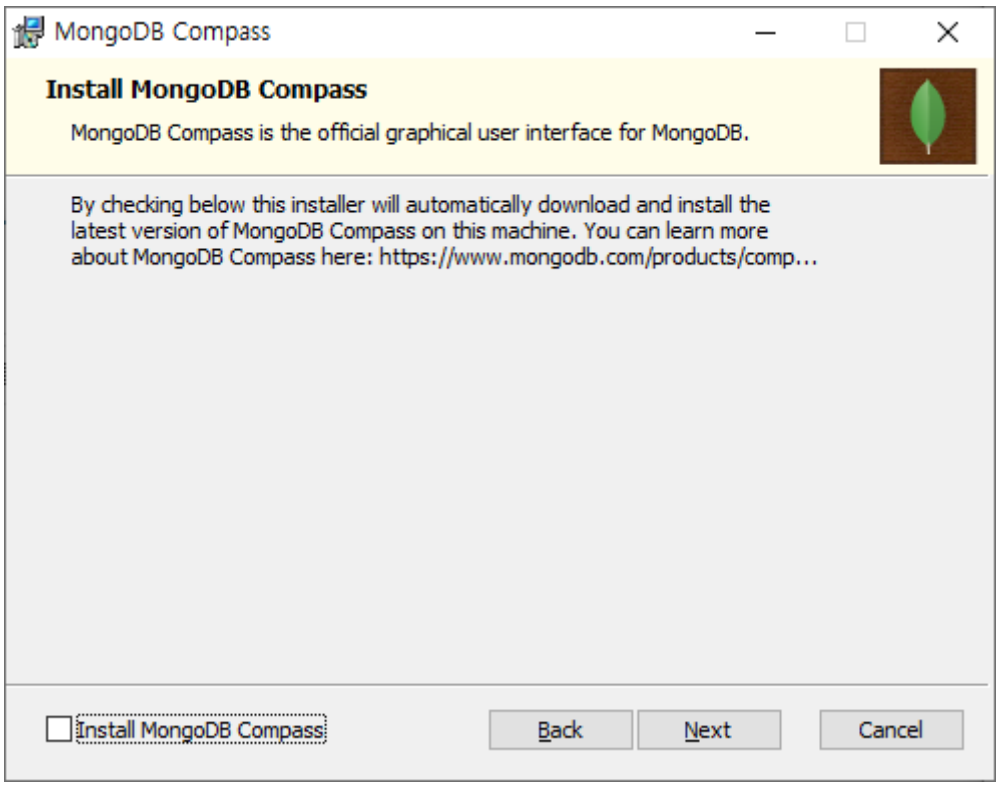
- 실습 + 이론 내용에서 나옴

MONGO DB

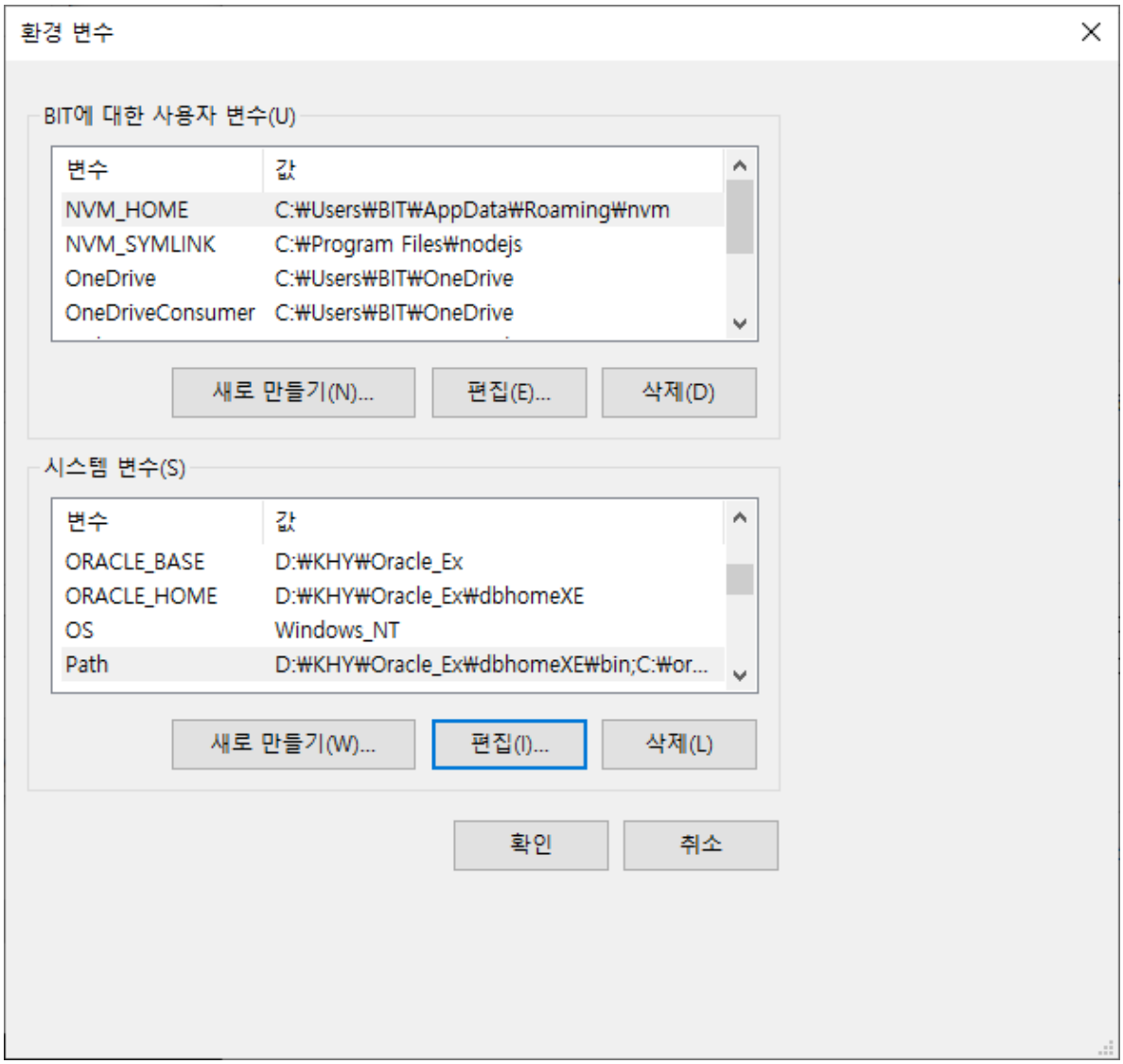
- Shading : 현재 사용하는 DBA 방법(특징을 이용할 때)
- HBASE : 하드 플랫폼용 DB
- Redis : 주로 쓰는 데이터를 메모리에 올리는 방식(리눅스 버전 밖에 없음)
- NoSQL

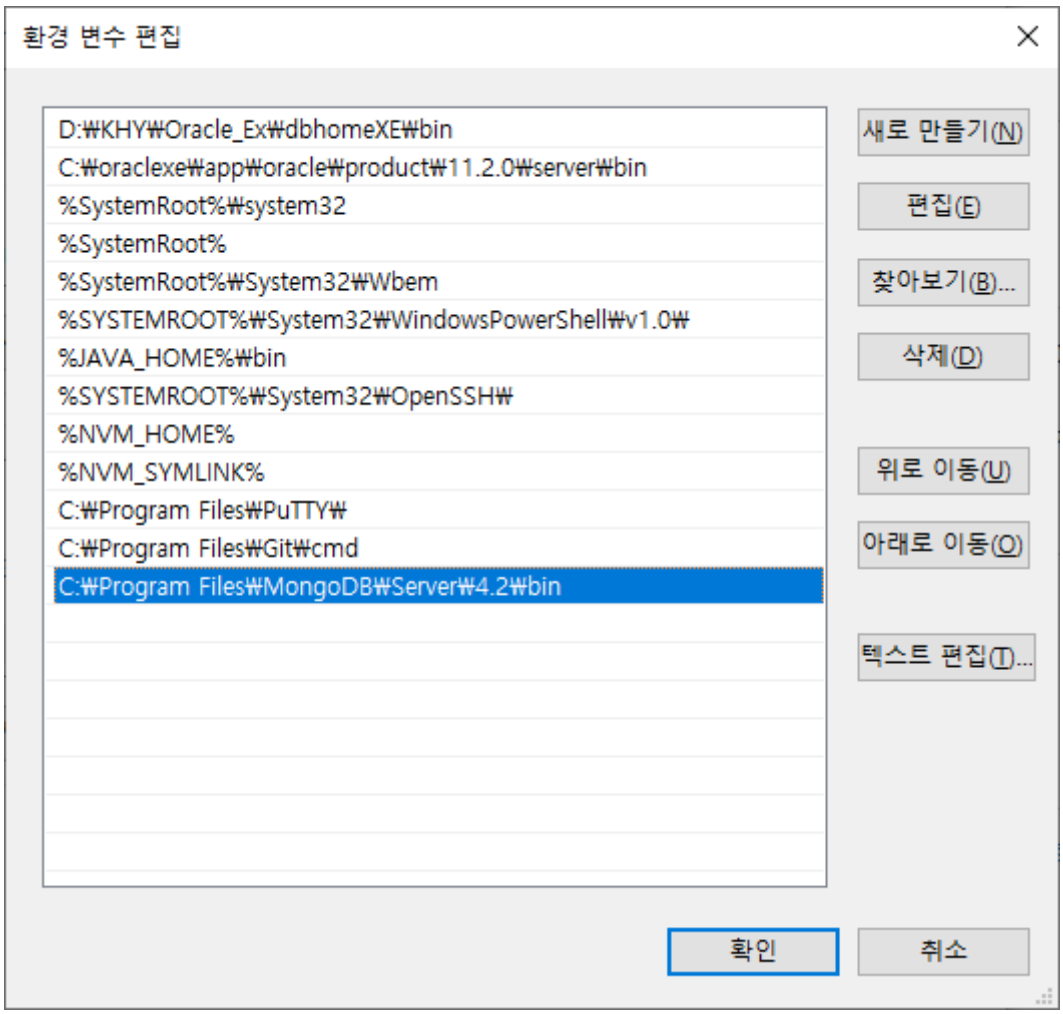
설치

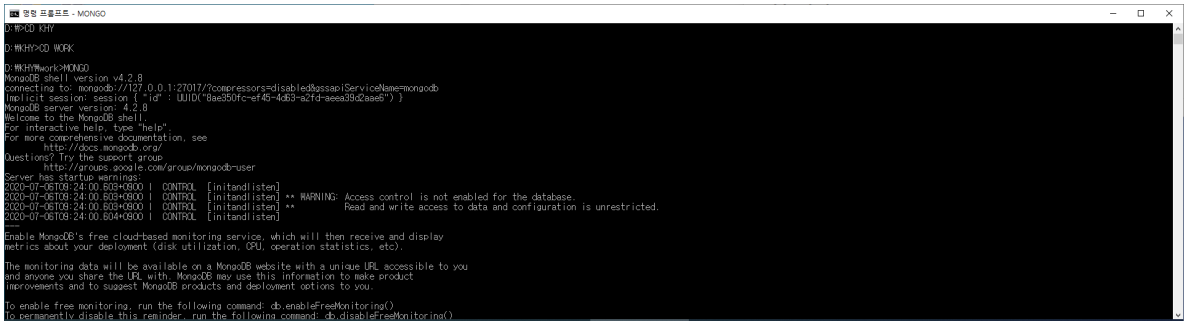




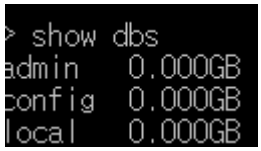
콤파스는 설치 시 오류가 생기는 기종들이 있어 지금은 뺌







> 해당 화면이 뜨지 않으면 터미널을 종료 후 다시 실행



사전지식

- 스키마가 없음
- 데이터를 도큐먼트 형식으로 저장(값 하나가 도큐먼트 하나)
- JSON 사용(타입 상관 없이 넣는 대로 저장가능)
- BSON이라고도 부름(Binary JSON)

용어

- document : row
- collection : table
- _id : primary key (수동 넣기 전엔 자동으로 생성)

1. Document

- {"키" : "벨류"}
- type-sensitive
- case-sensitive
- documents 내의 키/벨류 쌍은 순서를 가짐

2. Collections

- 다이나믹 스키마
- 서로 다른 shape을 가지는 documents는 하나의 collection에 저장될 수 있음
- any document can be put into any collection

3. Databases

- 컬렉션의 모임은 데이터베이스
- 하나의 몽고디비 인스턴스는 여러개의 데이터베이스 지원 가능
- 자체 퍼미션 존재, 각각의 디비에 별도의 파일로 저장됨(데이터 베이스를 파일 하나로 표현)
- 디비 이름은 대소문자 구분(특별한 경우 없는 한 소문자로 작성하는 것이 관례, 윈도우는 대소문자 구분을 안 하기 때문에 파일로 가져올 때 유의 해야함)

4. 네임스페이스 사용

- DB : cms
- collection : blog.post
- blog.posts collection의 네임스페이스는 cms.blog.posts임

5. 예약된 database 명

- admin
- local
- config

NoSQL(not only sql)

- 시퀄문 사용하지 않음
- 함수로만 이루어짐
- 앱(자바/파이썬 등) -> 드라이버(해당 언어) -> 몽고디비

실습

```
> post = {"title": "My Blog Post",
... "content": "Here's my blog post.",
... "date": new Date()}
{
        "title" : "My Blog Post",
        "content" : "Here's my blog post.",
        "date" : ISODate("2020-07-06T01:26:38.080Z")
}
```

```
> db.blog.insertOne(post)
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f027e37508ccb96b5700774")
}
```

> 중괄호를 ()안에 넣어도 됨

> 스크립트 쉘이라고 부름

```
> db.blog.find()
{ "_id" : ObjectId("5f027e37508ccb96b5700774"), "title" : "My Blog Post", "content" : "Here's my blog post.", "date" : ISODate("2020-07-06T01:26:38
.080Z") }
>
```

> 컬렉션이 존재하지 않으면 자동으로 생성하여 넣음

```
> db.blog.findOne()
{
        "_id" : ObjectId("5f027e37508ccb96b5700774"),
        "title" : "My Blog Post",
        "content" : "Here's my blog post.",
        "date" : ISODate("2020-07-06T01:26:38.080Z")
}
>
```

> 데이터를 하나만 볼 때

```
> post.comments = []
[ ]
```

> 빈 배열 생성

```
> db.blog.update({"title": "My Blog Post"}, post)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

```
> db.blog.findOne()
{
        "_id" : ObjectId("5f027e37508ccb96b5700774"),
        "title" : "My Blog Post",
        "content" : "Here's my blog post.",
        "date" : ISODate("2020-07-06T01:26:38.080Z"),
        "comments" : [ ]
}
>
```

```
> db.blog.remove({"title": "My Blog Post"})
WriteResult({ "nRemoved" : 1 })
> db.blog.findOne()
null
>
```

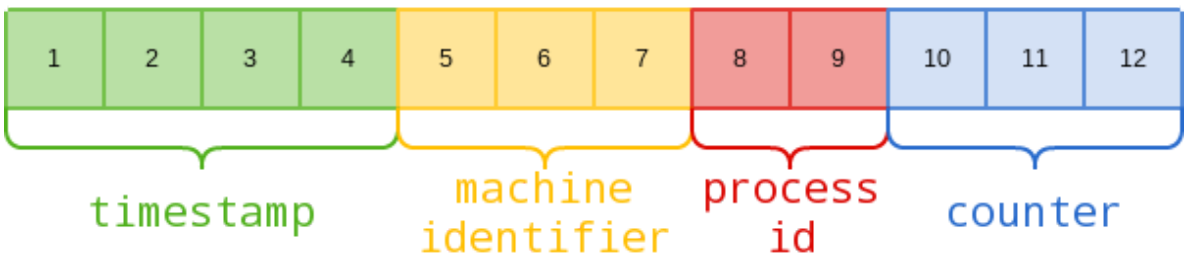
> remove로 삭제(유일한 키면 무엇이든 삭제 가능)

- 이렇듯 함수 언어들을 사용함

몽고디비 주요 데이터타입

1. null
2. boolean
3. number
4. string
5. date
6. regular expression
7. array : 배열의 값에 타입 종류 상관 없이 넣을 수 있음
8. embedded document : 도큐먼트 안 값 자체에 도큐먼트 작성 가능
9. object id
10. binary data
11. code : 자바 스크립트 코드 작성가능

ObjectID 구조



- objectid는 "_id"의 기본타입
- 서로 다른 장치를 넘나들면서 전체적으로 유일한 키 생성하는 방법
- 기본키를 위한 전통적인 자동생성 값을

- 머신코드: 서버코드
- 타임스탬: 실제 정보 들어가는 코드
- 프로세스 아이디: 몽고디비 아이디
- 카운터: 증가값(타임 머신 프로세스 아이디가 동일할 경우 증가하는 식)

```
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        sh.help()                    sharding helpers
        rs.help()                    replica set helpers
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce

        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
        show profile                 show most recent system.profile entries with time >= 1ms
        show logs                    show the accessible logger names
        show log [name]              prints out the last segment of log in memory, 'global' is default
        use <db_name>                set current database
        db.foo.find()                list objects in collection foo
        db.foo.find( { a : 1 } )     list objects in foo where a == 1
        it                           result of the last line evaluated; use to further iterate
        DBQuery.shellBatchSize = x   set default number of items to display on shell
        exit                         quit the mongo shell
>
```

> 함수를 괄호 빼고 타이핑 시 스크립트 언어 설명이 나옴

- API (applycation prognaming interface): 해당 언어를 위해 만들어 놓은 인터페이스 묶음(?)

CRUD

```
> db.movies.insertOne({"title": "Stand by Me"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f028a3d508ccb96b5700775")
}
```

```
> db.movies.drop()
true
```

> drop()으로 movies 컬렉션까지 삭제

```
> db.movies.insertMany([{"title": "Ghostbusters"},
... {"title": "E.T."},
... {"title": "Blade Runner"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f028b71508ccb96b5700777"),
                ObjectId("5f028b71508ccb96b5700778"),
                ObjectId("5f028b71508ccb96b5700779")
        ]
}
```

> 여러 값이 들어갈 것이므로 []으로 배열 표시

```
> db.movies.find()
{ "_id" : ObjectId("5f028b71508ccb96b5700778"), "title" : "E.T." }
{ "_id" : ObjectId("5f028b71508ccb96b5700779"), "title" : "Blade Runner" }
{ "_id" : ObjectId("5f028bbf508ccb96b570077a"), "title" : "Ghostbusters" }
>
```

> 고스트버스터즈의 경우 타이핑 미스로 인해 지운 후 새로 넣은 모습

Insert 종류

- Ordered Insert :

  InsertMany의 두 번째 파라미터로 true(기본값)을 주면, 제공되는 데이터 순서대로 삽입됨. 이 경우 에러를 만나면 그 후 데이터는 삽입되지 않음

```
> db.movies.insertMany([
... {"_id": 0, "title": "Top Gun"},
... {"_id": 1, "title": "Back to the Future"},
... {"_id": 1, "title": "Gremlins"},
... {"_id": 2, "title": "Aliens"}])
```

> 의도적으로 그램린의 id 중복 시킴

```
2020-07-06T11:35:02.180+0900 E  QUERY    [js] uncaught exception: BulkWriteError({
        "writeErrors" : [
                {
                        "index" : 2,
                        "code" : 11000,
                        "errmsg" : "E11000 duplicate key error collection: test.movies index: _id_ dup key: { _id: 1.0 }",
                        "op" : {
                                "_id" : 1,
                                "title" : "Gremlins"
                        }
                }
        ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
}) :
```

```
BulkWriteError({
        "writeErrors" : [
                {
                        "index" : 2,
                        "code" : 11000,
                        "errmsg" : "E11000 duplicate key error collection: test.movies index: _id_ dup key: { _id: 1.0 }",
                        "op" : {
                                "_id" : 1,
                                "title" : "Gremlins"
                        }
                }
        ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
BulkWriteError@src/mongo/shell/bulk_api.js:367:48
BulkWriteResult/this.toError@src/mongo/shell/bulk_api.js:332:24
Bulk/this.execute@src/mongo/shell/bulk_api.js:1186:23
DBCollection.prototype.insertMany@src/mongo/shell/crud_api.js:326:5
@(shell):1:1
```

```
> db.movies.find()
{ "_id" : 0, "title" : "Top Gun" }
{ "_id" : 1, "title" : "Back to the Future" }
>
```

> 에러 이후의 값들은 들어가지 않음

- Unordered Insert :

  InsertMany의 두 번째 파라미터로 false를 주면, 몽고디비는 성능 향상을 위해 삽입되는 데이터를 내부적으로 재정렬. 삽입 도중 에러 발생한 도큐먼트만 제외하고 나머지 도큐멘트들은 정상적으로 삽입

```
> db.movies.insertMany([
... {"_id": 3, "title": "Sixteen Candles"},
... {"_id": 4, "title": "The Terminator"},
... {"_id": 4, "title": "The Princess Bride"},
... {"_id": 5, "title": "Scarface"}],
... {"ordered": false})
```

```
2020-07-06T11:46:13.220+0900 E  QUERY    [js] uncaught exception: BulkWriteError({
        "writeErrors" : [
                {
                        "index" : 2,
                        "code" : 11000,
                        "errmsg" : "E11000 duplicate key error collection: test.movies index: _id_ dup key: { _id: 4.0 }",
                        "op" : {
                                "_id" : 4,
                                "title" : "The Princess Bride"
                        }
                }
        ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
}) :
```

```
BulkWriteError({
        "writeErrors" : [
                {
                        "index" : 2,
                        "code" : 11000,
                        "errmsg" : "E11000 duplicate key error collection: test.movies index: _id_ dup key: { _id: 4.0 }",
                        "op" : {
                                "_id" : 4,
                                "title" : "The Princess Bride"
                        }
                }
        ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
BulkWriteError@src/mongo/shell/bulk_api.js:367:48
BulkWriteResult/this.toError@src/mongo/shell/bulk_api.js:332:24
Bulk/this.execute@src/mongo/shell/bulk_api.js:1186:23
DBCollection.prototype.insertMany@src/mongo/shell/crud_api.js:326:5
@(shell):1:1
```

```
> db.movies.find()
{ "_id" : 0, "title" : "Top Gun" }
{ "_id" : 1, "title" : "Back to the Future" }
{ "_id" : 3, "title" : "Sixteen Candles" }
{ "_id" : 4, "title" : "The Terminator" }
{ "_id" : 5, "title" : "Scarface" }
>
```

> 프린세스 브라이드를 제외한 나머지 값들이 들어간 모습

```
> db.movies.deleteOne({"_id": 4})
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

```
> db.movies.find()
{ "_id" : 0, "title" : "Top Gun" }
{ "_id" : 1, "title" : "Back to the Future" }
{ "_id" : 3, "title" : "Sixteen Candles" }
{ "_id" : 5, "title" : "Scarface" }
>
```

> 하나 삭제 후 출력 결과(_id는 기본키)

> deleteMany를 써도 되긴 하지만 1개의 데이터만 지울 경우 실수 방지를 위해 One을 씀(조건을 많이 작성해도 한 가지만 지움)

```
> db.movies.insertMany([
... {"_id": 0, "title": "Top Gun", "year": 1986},
... {"_id": 1, "title": "Back to the Future", "year": 1985},
... {"_id": 3, "title": "Sixteen Candles", "year": 1984},
... {"_id": 4, "title": "The Terminator", "year": 1984},
... {"_id": 5, "title": "Scarface", "year": 1983}])
{ "acknowledged" : true, "insertedIds" : [ 0, 1, 3, 4, 5 ] }
>
```

```
> db.movies.find()
{ "_id" : 0, "title" : "Top Gun", "year" : 1986 }
{ "_id" : 1, "title" : "Back to the Future", "year" : 1985 }
{ "_id" : 3, "title" : "Sixteen Candles", "year" : 1984 }
{ "_id" : 4, "title" : "The Terminator", "year" : 1984 }
{ "_id" : 5, "title" : "Scarface", "year" : 1983 }
>
```

```
> db.movies.deleteMany({"year": 1984})
{ "acknowledged" : true, "deletedCount" : 2 }
```

```
> db.movies.find()
{ "_id" : 0, "title" : "Top Gun", "year" : 1986 }
{ "_id" : 1, "title" : "Back to the Future", "year" : 1985 }
{ "_id" : 5, "title" : "Scarface", "year" : 1983 }
>
```

> deleteOne으로 중복되는 값을 지우면 가장 위(처음 들어온) 것을 지움

```
> db.movies.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 3 }
> db.movies.find()
>
```

> 조건을 주지 않으면 전부 지움

```
> show collections
blog
movies
>
```

> show collection으로 흔적은 찾을 수 있으나 값은 전부 NULL로 되어있음

> 전체를 날릴 땐 drop이 더 나음

UPDATE

- UpdateOne, updateMany로 나뉨
- 첫 번째 인자로 필터를 받으며 두 번째 인자로 변경될 도큐먼트들의 내용을 받음
- replaceOne은 첫 번째 인자로 필터를 받으며 두 번째 인자로 대치할 도큐먼트 내용을 ㅏ받음
- 2개 이상의 update가 특정 도큐먼트에 동시 전달 될 땐 먼저 서버에 도착한 update 내용이 반영되고 후에 도착한 내용이 뒤이어 적용(마지막 전달 된 내용이 최종 반영)

replace

```
> db.users.insertOne({"name": "joe", "friends": 32, "enemies": 2})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f02a7e2508ccb96b570077b")
}
```

```
> var joe = db.users.findOne({"name": "joe"})
```

> find()로 하면 var가 배열 타입으로 잡혀서 문제가 생김

```
> joe.relationships = {"friends": joe.friends, "enemies": joe.enemies}
{ "friends" : 32, "enemies" : 2 }
```

```
> joe.username = joe.name
joe
```

```
> delete joe.friends
true
```

```
> delete joe.enemies
true
```

```
> delete joe.name
true
```

```
true
> db.users.replaceOne({"name": "joe"}, joe)
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

```
> db.users.find()
{ "_id" : ObjectId("5f02a7e2508ccb96b570077b"), "username" : "joe", "relationships" : { "friends" : 32, "enemies" : 2 } }
```

- replaceOne의 주의사항

```
> db.people.insertMany([ {"name": "joe", "age": 65}, {"name": "joe", "age": 20}, {"name": "joe", "age": 49}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f02acbc508ccb96b570077c"),
                ObjectId("5f02acbc508ccb96b570077d"),
                ObjectId("5f02acbc508ccb96b570077e")
        ]
}
>
```

```
> db.people.find()
{ "_id" : ObjectId("5f02acbc508ccb96b570077c"), "name" : "joe", "age" : 65 }
{ "_id" : ObjectId("5f02acbc508ccb96b570077d"), "name" : "joe", "age" : 20 }
{ "_id" : ObjectId("5f02acbc508ccb96b570077e"), "name" : "joe", "age" : 49 }
```

```
> joe = db.people.findOne({"name": "joe", "age": 20})
{ "_id" : ObjectId("5f02acbc508ccb96b570077d"), "name" : "joe", "age" : 20 }
>
```

```
> joe.age++
20
> joe.age
21
>
```

```
> db.people.replaceOne({"name": "joe"}, joe)
2020-07-06T14:05:01.719+0900 E  QUERY    [js] WriteError({
        "index" : 0,
        "code" : 66,
        "errmsg" : "After applying the update, the (immutable) field '_id' was found to have been altered to _id: ObjectId('5f02acbc508ccb96b57007
d')",
        "op" : {
                "q" : {
                        "name" : "joe"
                },
                "u" : {
                        "_id" : ObjectId("5f02acbc508ccb96b570077d"),
                        "name" : "joe",
                        "age" : 21
                },
                "multi" : false,
                "upsert" : false
        }
}) :
```

```
WriteError({
        "index" : 0,
        "code" : 66,
        "errmsg" : "After applying the update, the (immutable) field '_id' was found to have been altered to _id: ObjectId('5f02acbc508ccb96b57007
d')",
        "op" : {
                "q" : {
                        "name" : "joe"
                },
                "u" : {
                        "_id" : ObjectId("5f02acbc508ccb96b570077d"),
                        "name" : "joe",
                        "age" : 21
                },
                "multi" : false,
                "upsert" : false
        }
})
WriteError@src/mongo/shell/bulk_api.js:458:48
mergeBatchResults@src/mongo/shell/bulk_api.js:855:49
executeBatch@src/mongo/shell/bulk_api.js:919:13
Bulk/this.execute@src/mongo/shell/bulk_api.js:1163:21
DBCollection.prototype.replaceOne@src/mongo/shell/crud_api.js:510:17
@(shell):1:1
```

> field id was found... : id 중복(키 중복)으로 인해 대체가 불가하다(3be로만 전체를 바뀌려고 해서)

```
> db.people.replaceOne({"_id" : ObjectId("5f02acbc508ccb96b570077d")}, joe)
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.people.find()
{ "_id" : ObjectId("5f02acbc508ccb96b570077c"), "name" : "joe", "age" : 65 }
{ "_id" : ObjectId("5f02acbc508ccb96b570077d"), "name" : "joe", "age" : 21 }
{ "_id" : ObjectId("5f02acbc508ccb96b570077e"), "name" : "joe", "age" : 49 }
>
```

> 위 방식으로 직접 지정

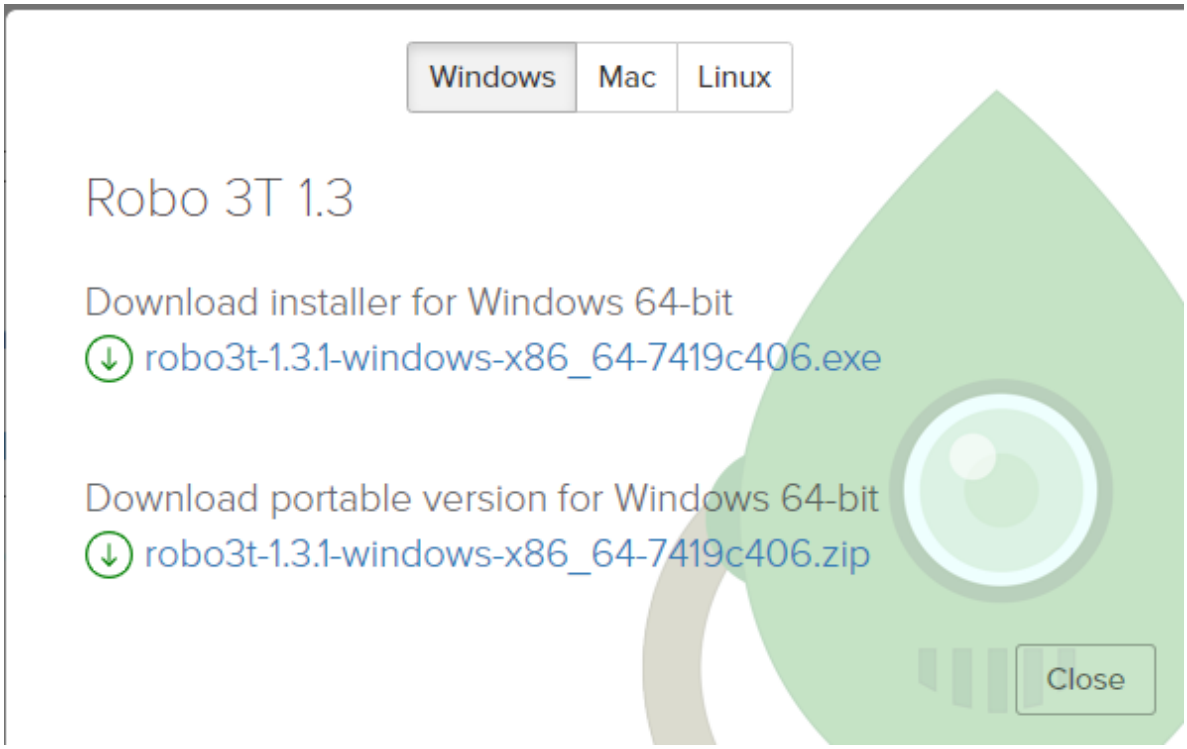
> 터미널에서 복사 : 엔터 / 붙여넣기: 마우스 우측

Update 연산자 사용

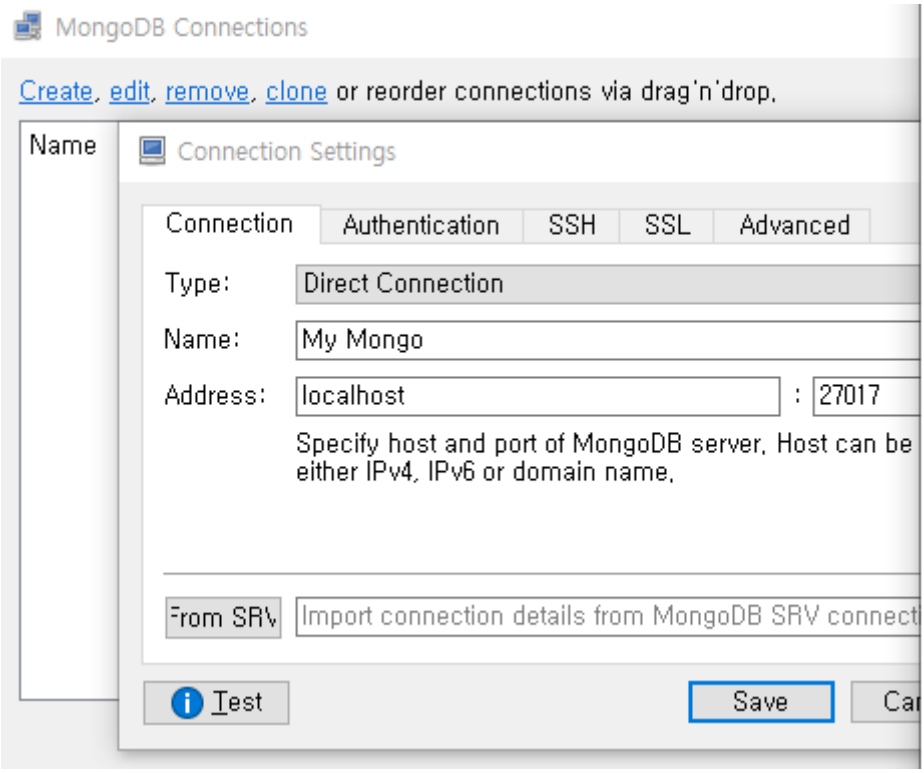
일반적으로 도큐먼트의 특정 필드만 업데이트가 필요함. 이 때 update 연산자 사용

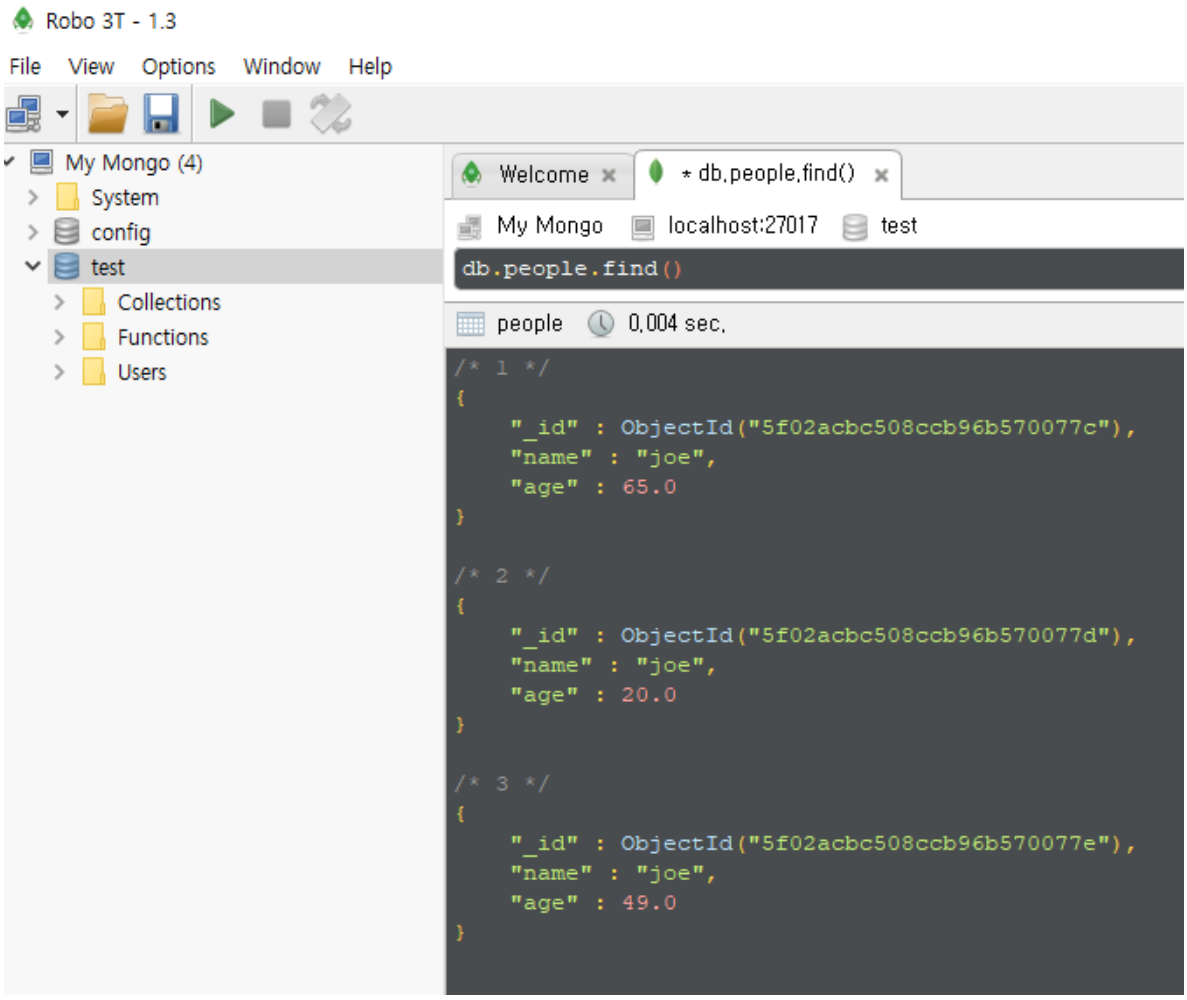
update 연산자는 변경, 더하기, 삭제, 배열 및 임베디드 도큐먼트등의 복잡한 연산을 위해 사용 됨

- 로보몽고



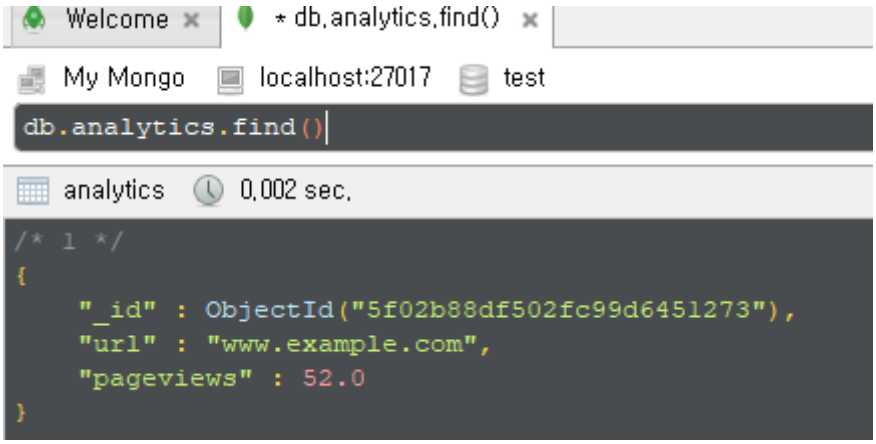
> download robo 3T -> exe 다운 -> 다음으로 계속 설치 -> agree -> finish

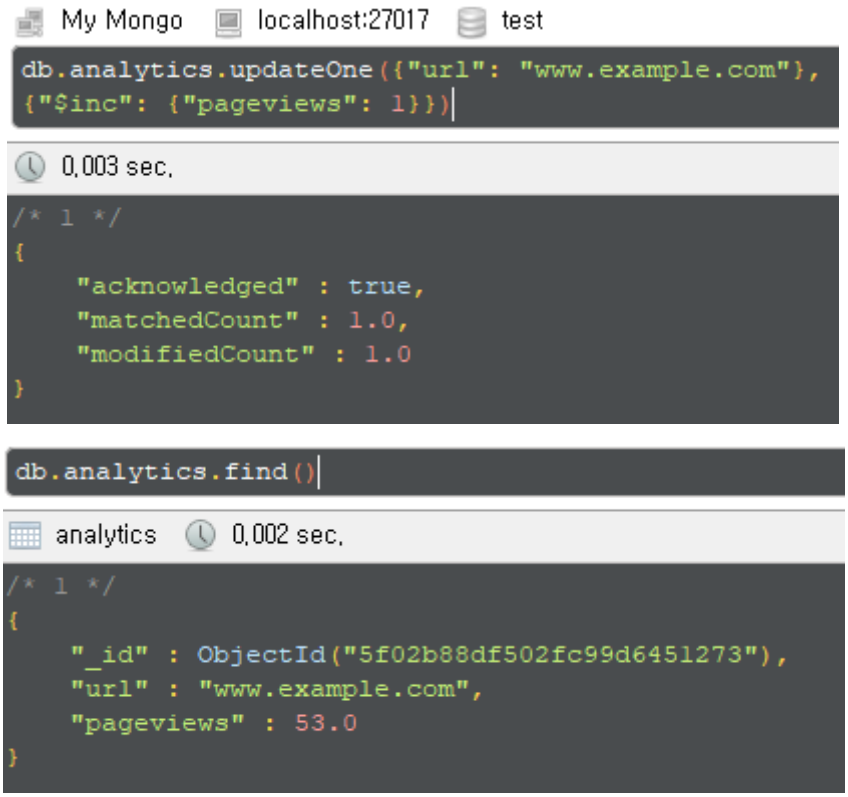




> test -> openShell (마우스 우측)

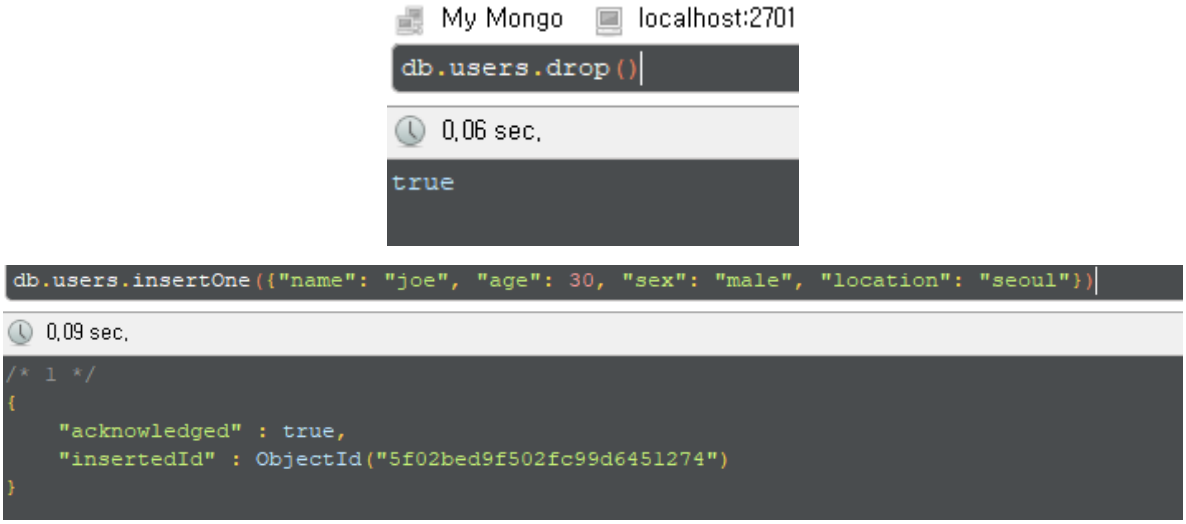
> F5로 실행



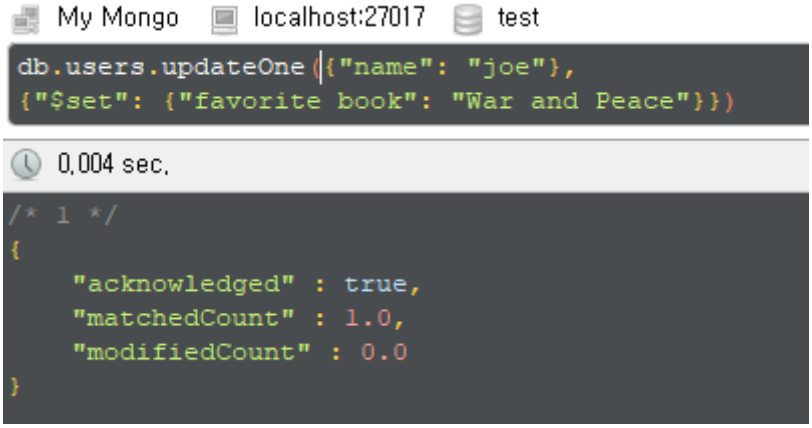


> $inc : 증가 연산자 역할(음수 영역으로 하면 빼기 가능)

- 실습



> $set : 존재하지 않은 필드를 set하면 만들어줌(존재하면 덮어쓰기)



My Mongo localhost:27017 test

```
db.users.find()
```

users 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02bed9f502fc99d6451274"),
    "name" : "joe",
    "age" : 30.0,
    "sex" : "male",
    "location" : "seoul",
    "favorite book" : "War and Peace"
}
```

My Mongo localhost:27017 test

```
db.users.updateOne({"name": "joe"},
{"$set": {"favorite book": "Green Eggs and Ham"}})
```

0.003 sec.

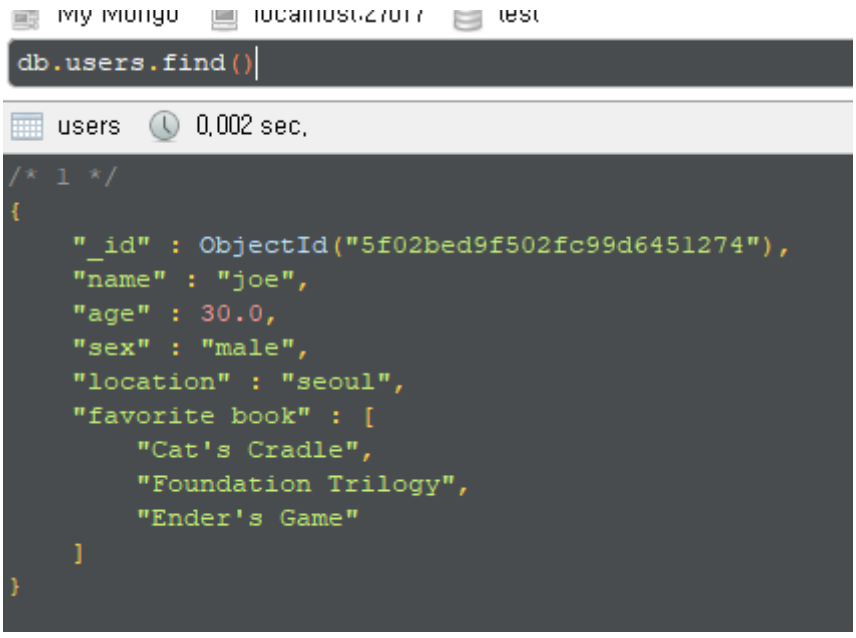
```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 1.0,
    "modifiedCount" : 1.0
}
```

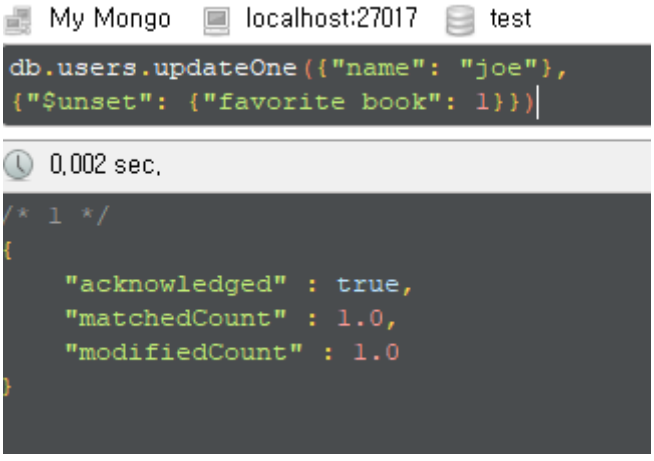
My Mongo localhost:27017 test

```
db.users.find()
```

users 0.003 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02bed9f502fc99d6451274"),
    "name" : "joe",
    "age" : 30.0,
    "sex" : "male",
    "location" : "seoul",
    "favorite book" : "Green Eggs and Ham"
}
```

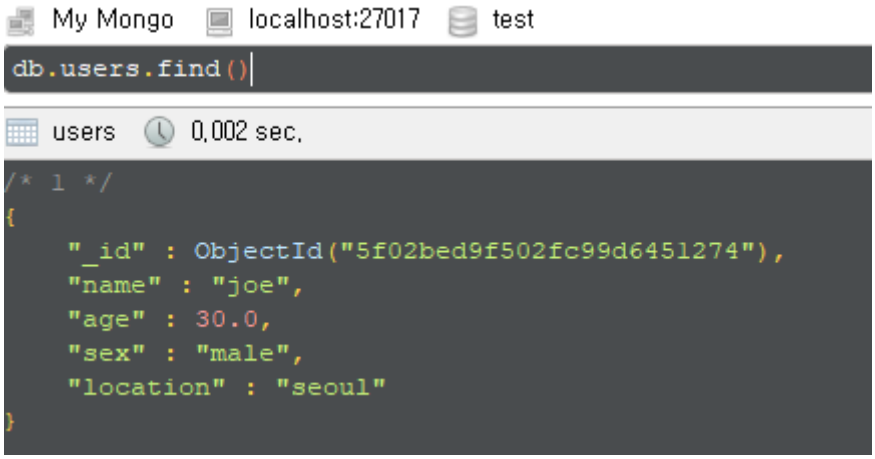


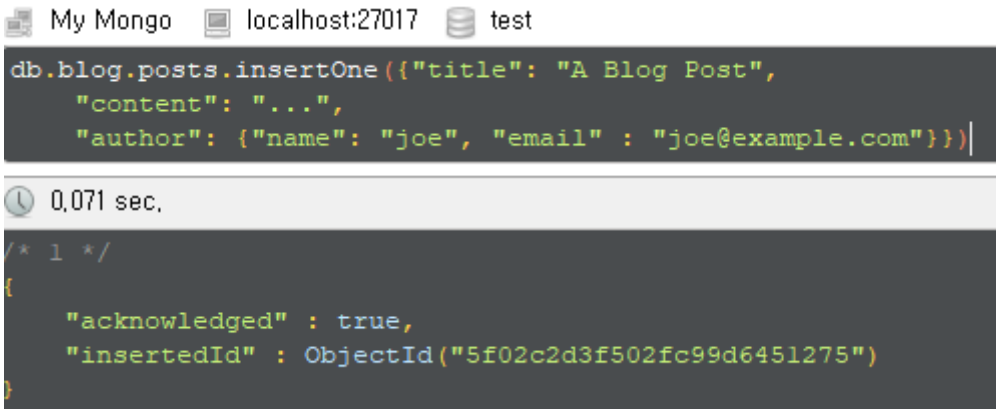


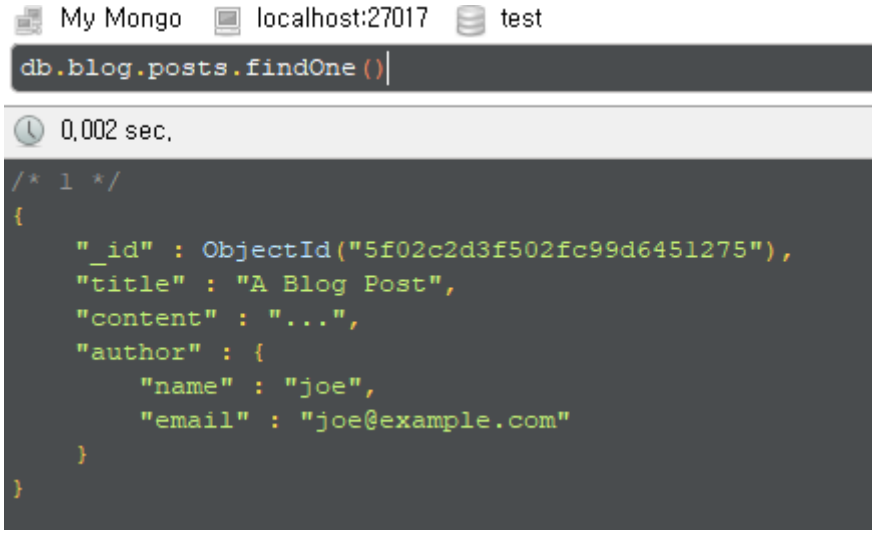
> $unset : 삭제

> 1은 아무 의미 없는 값(오류 발생 방지를 위함)

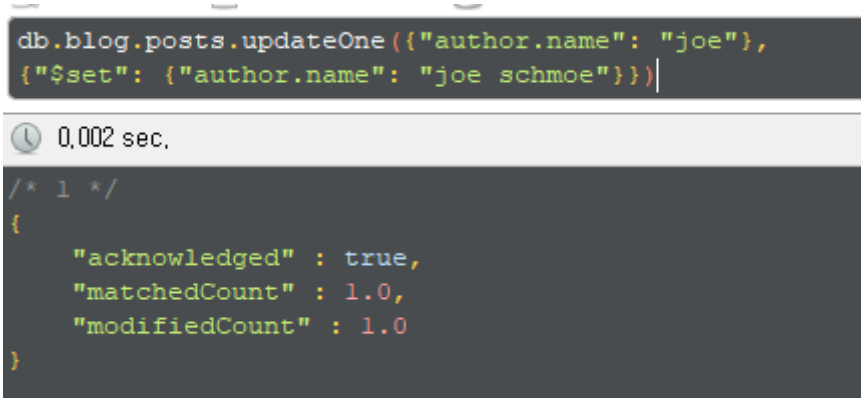
- 임베디드 도큐먼트

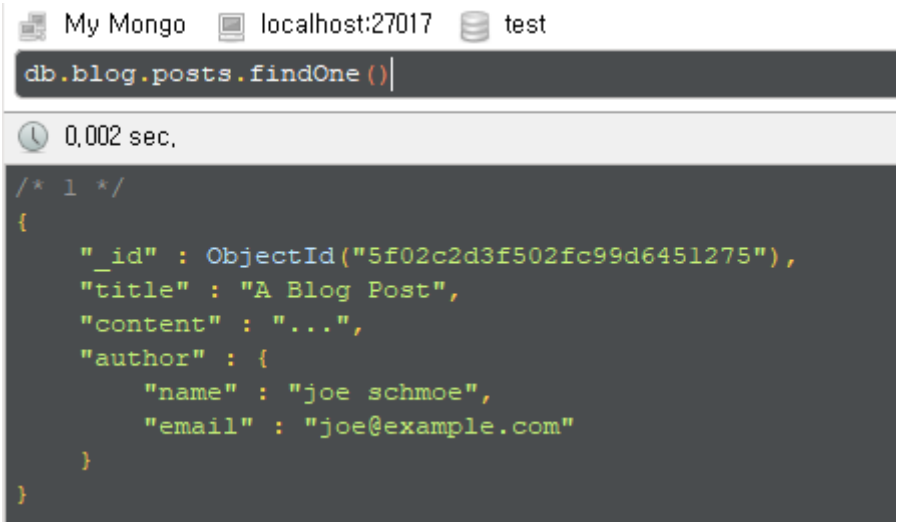


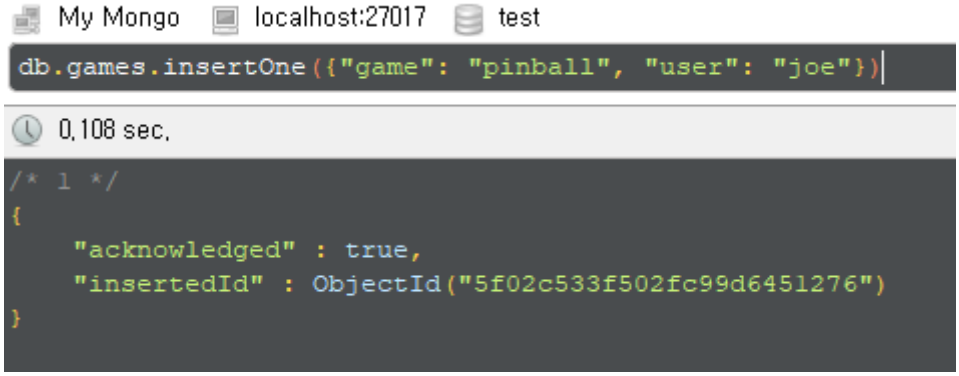


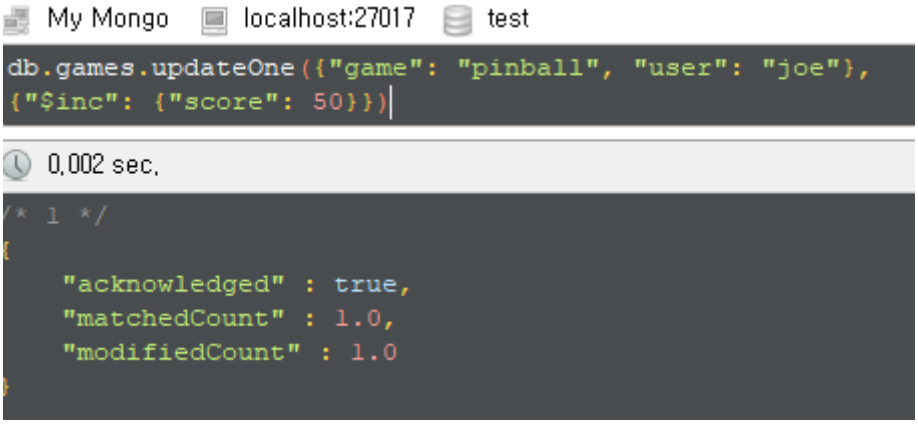


> 비정규화로 사용할 경우(임베디드 도큐먼트, 테이블 하나에 모든 정보 다 넣음) : 빠른 접근 가능
> 하지만 수정이 잦을 경우 효율을 위해 해당 정보만 collection을 나눠서 Join을 시킴



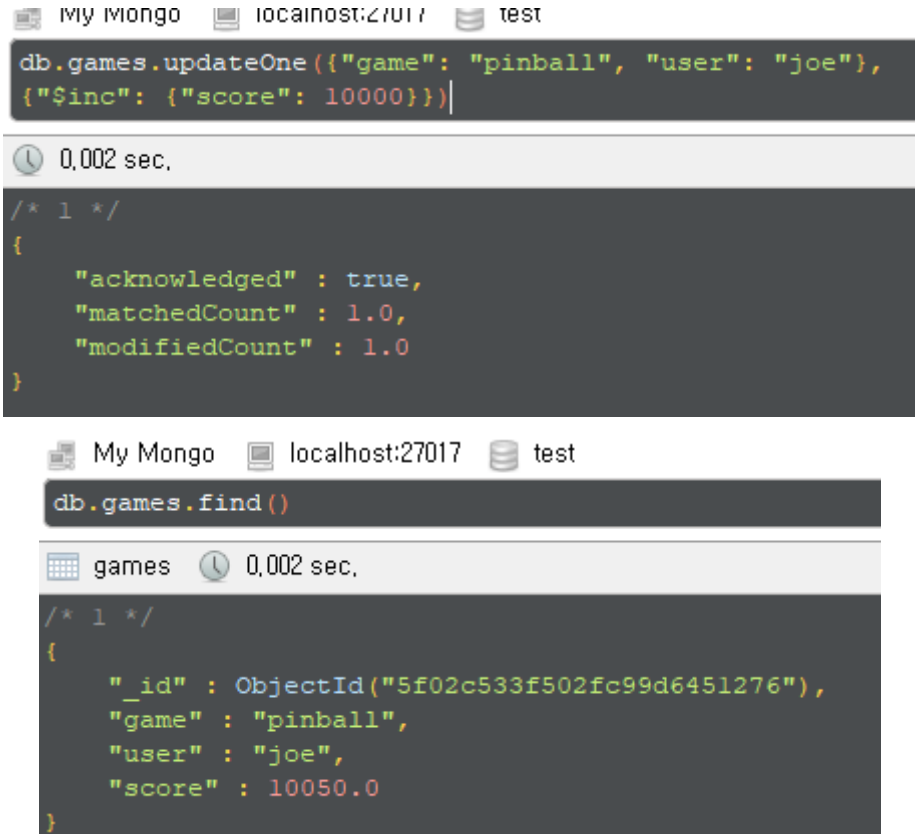




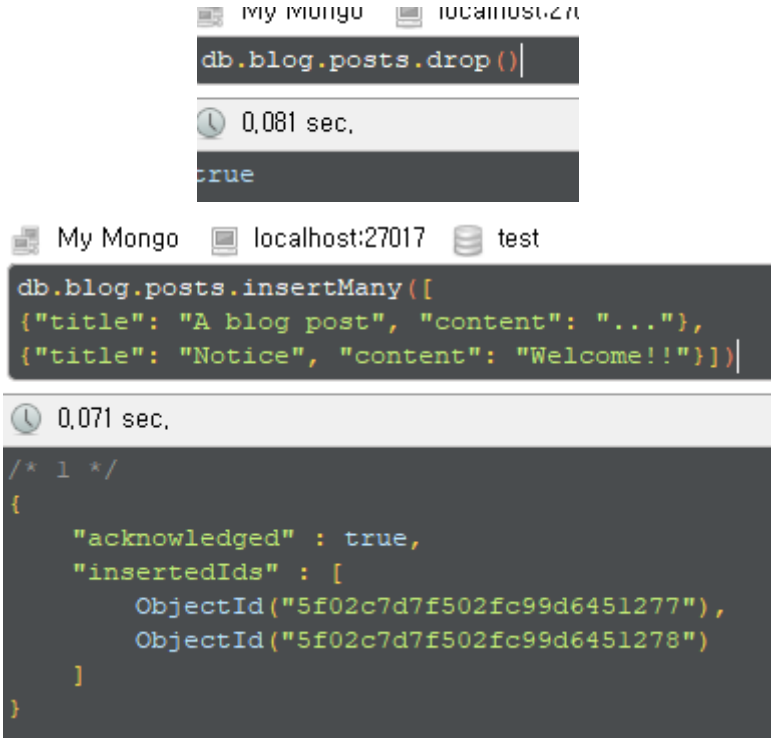


> inc 또한 존재하지 않는 필드는 만들어 줌(문자는 불가)

- inc를 이용하여 Score 1만점 추가해보기 실습


My Mongo
localhost:27017
test
db.games.updateOne({"game": "pinball", "user": "joe"},
{"$inc": {"score": 10000}})
0.002 sec.
/* 1 */
{
"acknowledged" : true,
"matchedCount" : 1.0,
"modifiedCount" : 1.0
}
My Mongo
localhost:27017
test
db.games.find()
games
0.002 sec.
/* 1 */
{
"_id" : ObjectId("5f02c533f502fc99d6451276"),
"game" : "pinball",
"user" : "joe",
"score" : 10050.0
}

- 항목이 배열일 경우


db.blog.posts.drop()
0.081 sec.
true
My Mongo
localhost:27017
test
db.blog.posts.insertMany([
{"title": "A blog post", "content": "..."},
{"title": "Notice", "content": "Welcome!!"}])
0.071 sec.
/* 1 */
{
"acknowledged" : true,
"insertedIds" : [
ObjectId("5f02c7d7f502fc99d6451277"),
ObjectId("5f02c7d7f502fc99d6451278")
]
}

My Mongo localhost:27017 test

```
db.blog.posts.find()
```

blog.posts 0.004 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02c7d7f502fc99d6451277"),
    "title" : "A blog post",
    "content" : "..."
}

/* 2 */
{
    "_id" : ObjectId("5f02c7d7f502fc99d6451278"),
    "title" : "Notice",
    "content" : "Welcome!!"
}
```

My Mongo localhost:27017 test

```
db.blog.posts.updateOne({"title": "A blog post"},
{"$push": {"comments":
    {"name": "joe", "email": "joe@example.com", "content": "nice post."}}})
```

0.002 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 1.0,
    "modifiedCount" : 1.0
}
```

> push : 배열에 새로운 항목 추가

My Mongo localhost:27017 test

```
db.blog.posts.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02c7d7f502fc99d6451277"),
    "title" : "A blog post",
    "content" : "...",
    "comments" : [
        {
            "name" : "joe",
            "email" : "joe@example.com",
            "content" : "nice post."
        }
    ]
}
```

My Mongo localhost:27017 test

```
db.blog.posts.updateOne({"title": "A blog post"},
{"$push": {"comments":
    {"name": "bob", "email": "bob@naver.com", "content": "good!!"}}})
```

0.005 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 1.0,
    "modifiedCount" : 1.0
}
```

```
db.blog.posts.find()
```

blog.posts 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02c7d7f502fc99d6451277"),
    "title" : "A blog post",
    "content" : "...",
    "comments" : [
        {
            "name" : "joe",
            "email" : "joe@example.com",
            "content" : "nice post."
        },
        {
            "name" : "bob",
            "email" : "bob@naver.com",
            "content" : "good!!"
        }
    ]
}

/* 2 */
{
    "_id" : ObjectId("5f02c7d7f502fc99d6451278"),
    "title" : "Notice",
    "content" : "Welcome!!"
}
```

My Mongo localhost:27017 test

```
db.stock.ticker.updateOne({"_id": "GooG"},
{"$push": {"hourly": {"$each": [562.762, 562.792, 234.123]}}})
```

0.002 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 0.0,
    "modifiedCount" : 0.0
}
```

> each : 배열의 항목을 한꺼번에 push

> 항목이 없기 때문에 modificount가 0

My Mongo localhost:27017 test

```
db.movies.insertOne({"genre": "horror", "top10":
    ["aaa", "bbb", "ccc", "ddd", "eee", "fff", "ggg", "hhh", "iii", "jjj"]})
```

0,112 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5f02cflef502fc99d645127a")
}
```

My Mongo localhost:27017 test

```
db.movies.find()
```

movies 0,002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02cflef502fc99d645127a"),
    "genre" : "horror",
    "top10" : [
        "aaa",
        "bbb",
        "ccc",
        "ddd",
        "eee",
        "fff",
        "ggg",
        "hhh",
        "iii",
        "jjj"
    ]
}
```

Welcome × * db.movies.updateOne({"··· ×

My Mongo localhost:27017 test

```
db.movies.updateOne({"genre": "horror"},
{"$push": {"top10": {"$each": ["Nightmare on Elm Street", "Saw"],
    "$slice": -10}}})
```

0,002 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 1.0,
    "modifiedCount" : 1.0
}
```

> slice : 옵션(조건), 마이너스일 경우 뒤에서부터 N개 플러스일 경우 앞에서부터 N개(항상 each와 써야함)

My Mongo localhost:27017 test

```
db.movies.findOne()
```

0,002 sec.

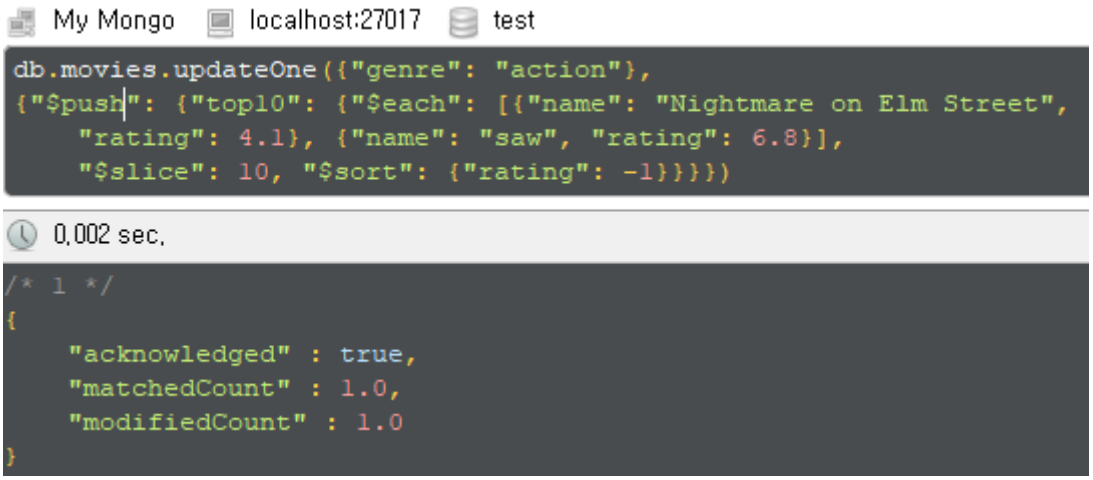
```
/* 1 */
{
    "_id" : ObjectId("5f02cf1ef502fc99d645127a"),
    "genre" : "horror",
    "top10" : [
        "ccc",
        "ddd",
        "eee",
        "fff",
        "ggg",
        "hhh",
        "iii",
        "jjj",
        "Nightmare on Elm Street",
        "Saw"
    ]
}
```

My Mongo localhost:27017 test

```
db.movies.insertOne({"genre": "action", "top10": [
    {"name": "111", "rating": 4.3},
    {"name": "222", "rating": 6.6},
    {"name": "333", "rating": 5.5}]})
```

0,002 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5f02d271f502fc99d645127c")
}
```



> sort : 해당 필드를 오름차순 혹은 내림차순으로 정렬(항상 each랑 써야 함)

My Mongo localhost:27017 test

```
db.movies.find()
```

movies 0,001 sec.


```
{
    "_id" : ObjectId("5f02d271f502fc99d645127c"),
    "genre" : "action",
    "top10" : [
        {
            "name" : "saw",
            "rating" : 6.8
        },
        {
            "name" : "222",
            "rating" : 6.6
        },
        {
            "name" : "333",
            "rating" : 5.5
        },
        {
            "name" : "111",
            "rating" : 4.3
        },
        {
            "name" : "Nightmare on Elm Street",
            "rating" : 4.1
        }
    ]
}
```

```
db.products.insertOne({"product name": "Television"})
```

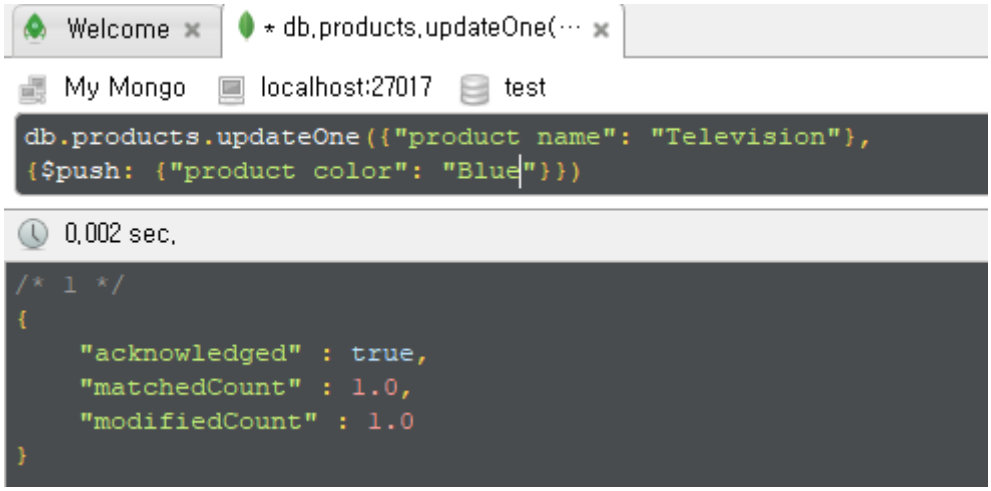
0,091 sec.


```
/* 1 */
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5f02d4cff502fc99d645127d")
}
```

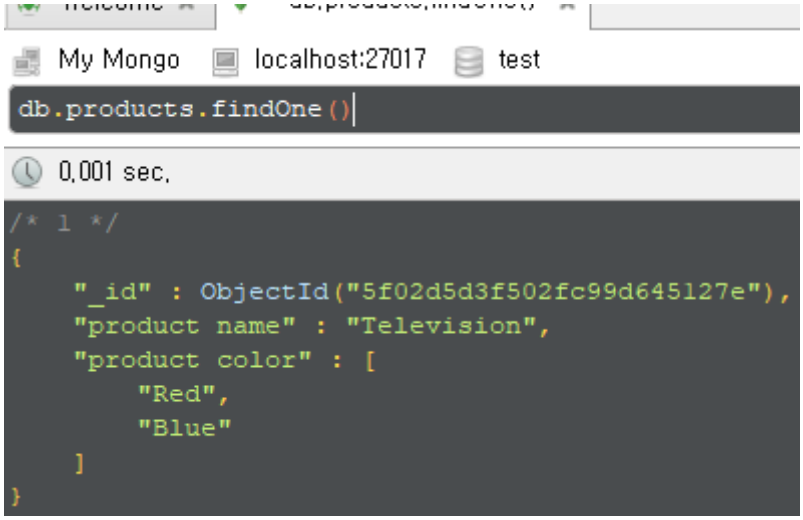

```
db.products.updateOne({"product name": "Television"},
{$push: {"product color": "Red"}})
```

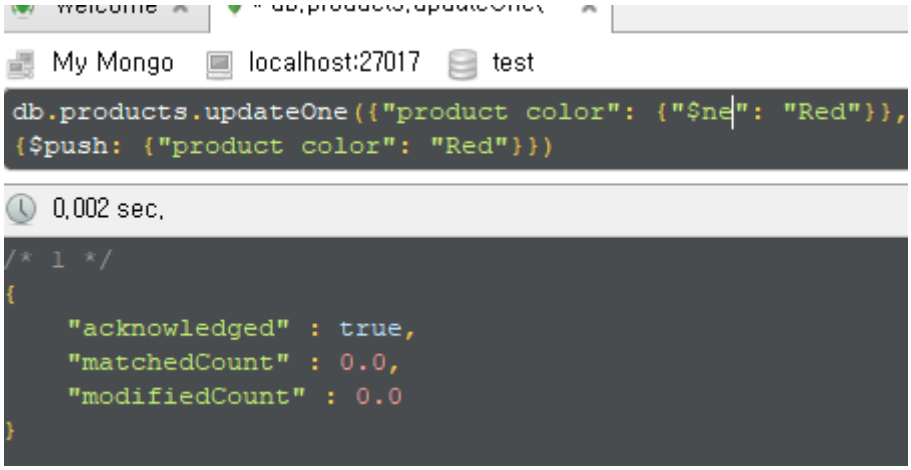
0,002 sec.


```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 1.0,
    "modifiedCount" : 1.0
}
```



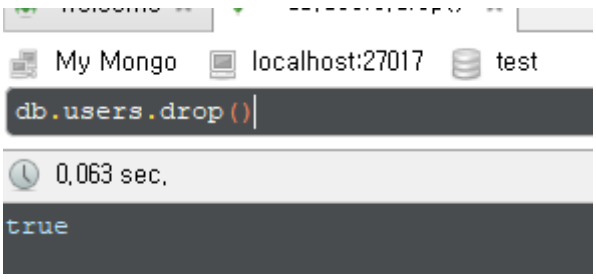
"" 쌍따옴표를 필드에 안 써도 되지만 쓰는 것이 좋은 습관



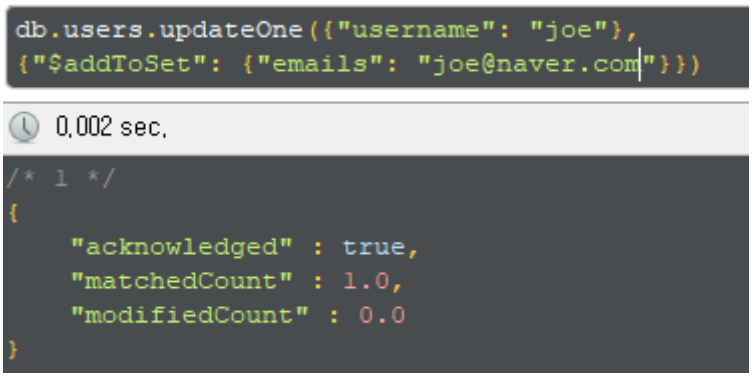


ne : not exist

Red 색이 없으면 Red 색을 추가



> addToSet : 있으면 넣지말고 없으면 추가하라



> 중복되는 것은 모디파이카운트가 0

```
db.users.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02db88f502fc99d645127f"),
    "username" : "joe",
    "emails" : [
        "joe@example.com",
        "joe@naver.com",
        "joe@gmail.com",
        "joe@daum.net"
    ]
}
```