addToSet

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

users 0.002 sec.

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

```
db.users.updateOne({"username": "joe"},
{"$addToSet": {"emails": {"$each":
    ["joe@naver.com", "joe@gmail.com", "joe@bit.com"]}}})
```

0.002 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 1.0,
    "modifiedCount" : 0.0
}
```

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

users 0.002 sec.

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

> addToSet이 훨씬 명시적이고 직관적

pop

```
db.users.updateOne({"username": "joe"},
{"$pop": {"emails": 1}})
```

0.003 sec.

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

> pop : 맨 뒤에서 한 개 뽑음(양수 1은 맨앞 음수 1은 맨 뒤)

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

users 0.002 sec.

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

> ROBO IDE에서 한 줄 씩 실행하기 : 해당 구문 선택 후 F5

pull

```
db.lists.insertOne({"todo": ["dishes", "laundry", "gardening"]})
db.lists.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c15b236acd8a3e4bde24"),
    "todo" : [
        "dishes",
        "laundry",
        "gardening"
    ]
}
```

```
db.lists.updateOne({}, {"$pull": {"todo": "laundry"}})
db.lists.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c15b236acd8a3e4bde24"),
    "todo" : [
        "dishes",
        "gardening"
    ]
}
```

> pull : 특정 항목을 삭제, 앞의 조건이 있어야 실행되므로 빈칸으로 채워놓음

인덱스 형식으로 참조하기

```
db.blog.posts.insertOne({"content": "...",
    "comments": [
        {"comment": "good post", "author": "John", "votes": 0},
        {"comment": "I thought it was too short", "author": "Claire", "votes": 3},
        {"comment": "free watches", "author": "Alice", "votes": -5},
        {"comment": "vacation getaways", "author": "Lynn", "votes": -7}
        ]
    })
db.blog.posts.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c386236acd8a3e4bde26"),
    "content" : "...",
    "comments" : [
        {
            "comment" : "good post",
            "author" : "John",
            "votes" : 0.0
        },
        {
            "comment" : "I thought it was too short",
            "author" : "Claire",
            "votes" : 3.0
        },
        {
            "comment" : "free watches",
            "author" : "Alice",
            "votes" : -5.0
        },
        {
            "comment" : "vacation getaways",
            "author" : "Lynn",
            "votes" : -7.0
        }
    ]
}
```

```
db.blog.posts.updateOne({"comments.author": "John"},
{"$inc": {"comments.0.votes": 1}})
db.blog.posts.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c386236acd8a3e4bde26"),
    "content" : "...",
    "comments" : [
        {
            "comment" : "good post",
            "author" : "John",
            "votes" : 2.0
        },
        {
            "comment" : "I thought it was too short",
            "author" : "Claire",
            "votes" : 3.0
        },
        {
            "comment" : "free watches",
            "author" : "Alice",
            "votes" : -5.0
        },
        {
            "comment" : "vacation getaways",
            "author" : "Lynn",
            "votes" : -7.0
        }
    ]
}
```

> comments의 0번째 votes를 지칭, 앞의 존은 갱신 할 업데이트 대상 도큐먼트(하지만 존이 첫 번째인 것을 알아서 이런식으로 작성한 것)

> 해당 화면은 실행 한 번을 더 해서 2로 증가한 모습

$ (위치 지정 연산자)

```
db.blog.posts.updateOne({"comments.author": "John"},
{"$set": {"comments.$.author": "Jim"}})
db.blog.posts.findOne()
```

0.003 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c386236acd8a3e4bde26"),
    "content" : "...",
    "comments" : [
        {
            "comment" : "good post",
            "author" : "Jim",
            "votes" : 2.0
        },
        {
            "comment" : "I thought it was too short",
            "author" : "Claire",
            "votes" : 3.0
        },
        {
            "comment" : "free watches",
            "author" : "Alice",
            "votes" : -5.0
        },
        {
            "comment" : "vacation getaways",
            "author" : "Lynn",
            "votes" : -7.0
        }
    ]
}
```

```
db.blog.posts.updateOne({"comments.author": "Jim"},
{"$inc": {"comments.$.votes": 1}})
db.blog.posts.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c386236acd8a3e4bde26"),
    "content" : "...",
    "comments" : [
        {
            "comment" : "good post",
            "author" : "Jim",
            "votes" : 3.0
        },
        {
            "comment" : "I thought it was too short",
            "author" : "Claire",
            "votes" : 3.0
        },
        {
            "comment" : "free watches",
            "author" : "Alice",
            "votes" : -5.0
        },
        {
            "comment" : "vacation getaways",
            "author" : "Lynn",
            "votes" : -7.0
        }
    ]
}
```

> author Jim의 위치 찾은 정보를 $연산자가 가지고 있어 위 처럼 인덱스 참조할 필요 없이 사용 가능

$[elem] (엘리먼트) & arrayFilter

```
db.blog.posts.updateOne({},
{"$set": {"comments.$[elem].hidden": true}},
{arrayFilters: [{"elem.votes": {$lte: -5}}]})
db.blog.posts.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c386236acd8a3e4bde26"),
    "content" : "...",
    "comments" : [
        {
            "comment" : "good post",
            "author" : "Jim",
            "votes" : 3.0
        },
        {
            "comment" : "I thought it was too short",
            "author" : "Claire",
            "votes" : 3.0
        },
        {
            "comment" : "free watches",
            "author" : "Alice",
            "votes" : -5.0,
            "hidden" : true
        },
        {
            "comment" : "vacation getaways",
            "author" : "Lynn",
            "votes" : -7.0,
            "hidden" : true
        }
    ]
}
```

> arrayFilter로 조건을 줄 수 있음(-5 이하의 값들에 hidden이라는 필드와 true값 추가)

> lte : Less than Equal / gte : greater than Equal

3. Upsert

필터에 매칭되는 도큐먼트가 없을 경우 새로운 도큐먼트가 생성되며 매칭되는 도큐먼트가 있을 경우엔 정상적인 update가 이뤄짐(없으면 insert 있으면 update)

```
db.analytics.find()
```

analytics 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02b88df502fc99d6451273"),
    "url" : "www.example.com",
    "pageviews" : 53.0
}
```

```
db.analytics.updateOne({"url": "/blog"},
{"$inc": {"pageviews": 1}}, {"upsert": true})
```

0.004 sec.

```
/* 1 */
{
    "acknowledged" : true,
    "matchedCount" : 0.0,
    "modifiedCount" : 0.0,
    "upsertedId" : ObjectId("5f03cea587ld43db14eceb9c")
}
```

> upsert를 true로 하여 없으면 만들어라는 코드로 작성

```
db.analytics.updateOne({"url": "/blog"},
{"$inc": {"pageviews": 1}}, {"upsert": true})

db.analytics.find()
```

analytics 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f02b88df502fc99d6451273"),
    "url" : "www.example.com",
    "pageviews" : 53.0
}

/* 2 */
{
    "_id" : ObjectId("5f03cea587ld43db14eceb9c"),
    "url" : "/blog",
    "pageviews" : 1.0
}
```

setOnInsert

해당 도큐먼트를 처음 생성할 때만 값을 넣어라

```
db.users.updateOne({"username": "kim"},
{"$setOnInsert": {"createdAt": new Date()}}, {"upsert": true})
db.users.find()
```

users 0.002 sec.

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

/* 2 */
{
    "_id" : ObjectId("5f03cfd2871d43db14ecebb7"),
    "username" : "kim",
    "createdAt" : ISODate("2020-07-07T01:28:50.860Z")
}
```

> 1번 째 실행

```
db.users.updateOne({"username": "kim"},
{"$setOnInsert": {"createdAt": new Date()}}, {"upsert": true})
db.users.find()
```

users 0.002 sec.

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

/* 2 */
{
    "_id" : ObjectId("5f03cfd2871d43db14ecebb7"),
    "username" : "kim",
    "createdAt" : ISODate("2020-07-07T01:28:50.860Z")
}
```

> 2번 째 실행, 시간 갱신되지 않음

```
db.users.updateOne({"username": "joe"},
{"$setOnInsert": {"createdAt": new Date()}}, {"upsert": true})
db.users.find()
```

users 0.002 sec.

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

/* 2 */
{
    "_id" : ObjectId("5f03cfd2871d43db14ecebb7"),
    "username" : "kim",
    "createdAt" : ISODate("2020-07-07T01:28:50.860Z")
}
```

> username이 joe인 도큐먼트가 있으므로 createdAt이라는 필드가 만들어지지 않음

updateMany

```
db.users.drop()
```

0.085 sec.

```
true
```

```
db.users.insertMany([
{"birthday": "10/13/1978"},
{"birthday": "10/13/1978"},
{"birthday": "10/13/1978"}])
db.users.find()
```

users 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03d1f6236acd8a3e4bde27"),
    "birthday" : "10/13/1978"
}

/* 2 */
{
    "_id" : ObjectId("5f03d1f6236acd8a3e4bde28"),
    "birthday" : "10/13/1978"
}

/* 3 */
{
    "_id" : ObjectId("5f03d1f6236acd8a3e4bde29"),
    "birthday" : "10/13/1978"
}
```

```
db.users.updateMany({"birthday": "10/13/1978"},
{"$set": {"gift": "Happy Birthday!"}})
db.users.find()
```

users 0.001 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03d1f6236acd8a3e4bde27"),
    "birthday" : "10/13/1978",
    "gift" : "Happy Birthday!"
}

/* 2 */
{
    "_id" : ObjectId("5f03d1f6236acd8a3e4bde28"),
    "birthday" : "10/13/1978",
    "gift" : "Happy Birthday!"
}

/* 3 */
{
    "_id" : ObjectId("5f03d1f6236acd8a3e4bde29"),
    "birthday" : "10/13/1978",
    "gift" : "Happy Birthday!"
}
```

1) find

find 메소드의 첫 번째 인자가 쿼리 조건을 명시하는 도큐먼트

컬렉션안의 모든 도큐먼트를 지칭하기 위해 빈 쿼리 도큐먼트({})를 사용할 수 있음

만약 find 메소드의 쿼리 도큐먼트가 생략되면 {}로 설정

find에는 실제 값이 와야됨(참조 값은 불가능)

```
db.users.drop()
```

0.062 sec.

```
true
```

```
db.users.insertMany([
{"username": "joe", "age": 27, "sex": "male"},
{"username": "Jane", "age": 26, "sex": "female"}])
db.users.find()
```

users 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03d3cc236acd8a3e4bde2a"),
    "username" : "joe",
    "age" : 27.0,
    "sex" : "male"
}

/* 2 */
{
    "_id" : ObjectId("5f03d3cc236acd8a3e4bde2b"),
    "username" : "Jane",
    "age" : 26.0,
    "sex" : "female"
}
```

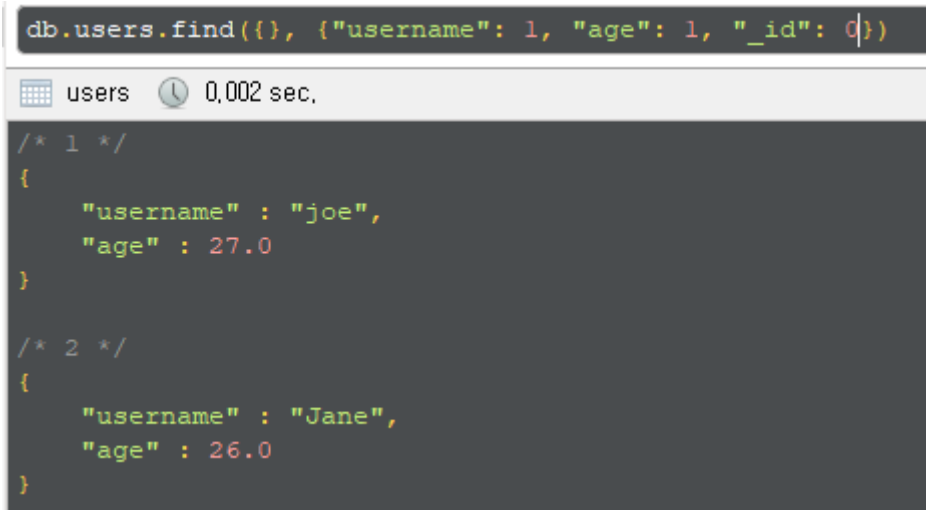
```
db.users.find({}, {"username": 1, "age": 1})
```

users 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03d3cc236acd8a3e4bde2a"),
    "username" : "joe",
    "age" : 27.0
}

/* 2 */
{
    "_id" : ObjectId("5f03d3cc236acd8a3e4bde2b"),
    "username" : "Jane",
    "age" : 26.0
}
```

> username과 age를 보겠다는 코드, id는 디폴트로 나옴
>
> 2번 째 항목 : 출력할 필드



> 명시적으로 id를 출력하지 않게 하는 코드

lt / lte / gt / gte

< / <= / > / >=

2) OR 쿼리

- "$in"
- "$in"에 제공되는 배열 항목이 한 개일때는 다이렉트 매칭과 동일하게 작동
- "$in"의 반대는 $nin
- 비교할 키가 2개 이상인 경우 "$or" 사용

```
db.raffle.find({"$or": [{"tiket_no": {"$in": [123, 124, 324]}}, {"winner": true}]})
```

- 가능하다면 "$or" 보다는 "$in"을 사용 -> 쿼리 optimizer가 작동

3) $not

- 모든 조건에 함께 사용 가능
- "id_num" 값을 5로 나눈 후 나머지가 1인 경우

```
find({"id_num": {"$mod": [5, 1]}})
```

- 위와 반대인 경우

```
find({"id_num": {"$not": {"$mod": [5, 1]}}})
```

```
db.null_check.insertMany([
{"y": null},
{"y": 1},
{"y": 2}])
db.null_check.find()
```

null_check 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2c"),
    "y" : null
}

/* 2 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2d"),
    "y" : 1.0
}

/* 3 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2e"),
    "y" : 2.0
}
```

```
db.null_check.insertMany([
{"y": null},
{"y": 1},
{"y": 2}])
db.null_check.find()
db.null_check.find({"y": null})
```

null_check 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2c"),
    "y" : null
}
```

> y 필드에 널 값인 것 find

```
db.null_check.insertMany([
{"y": null},
{"y": 1},
{"y": 2}])
db.null_check.find()
db.null_check.find({"x": null})
```

null_check 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2c"),
    "y" : null
}

/* 2 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2d"),
    "y" : 1.0
}

/* 3 */
{
    "_id" : ObjectId("5f03dc69236acd8a3e4bde2e"),
    "y" : 2.0
}
```

> x라는 필드가 존재하지 않느냐(null) 라는 조건으로 들어감, 따라서 전체 값을 find

```
db.null_check.insertMany([
{"y": null},
{"y": 1},
{"y": 2}])
db.null_check.find()
db.null_check.find({"x": {"$eq": null, "$exists": true}})
```

0.005 sec.

```
Fetched 0 record(s) in 4ms
```

> 해당 필드가 실제 존재하느냐부터 검사를 하는 코드가 올바른 코드

```
db.food.insertOne({"fruit": ["apple", "banna", "peach"]})
db.food.find({"fruit": "banna"})
db.food.find({"fruit": ["banna", "peach"]})
```

food 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03de8a236acd8a3e4bde2f"),
    "fruit" : [
        "apple",
        "banna",
        "peach"
    ]
}
```

```
db.food.insertOne({"fruit": ["apple", "banna", "peach"]})
db.food.find({"fruit": "banna"})
db.food.find({"fruit": ["banna", "peach"]})
```

0.001 sec.

```
Fetched 0 record(s) in 1ms
```

> 같은 배열인가를 찾음

all

```
db.food.insertOne({"_id": 1, "fruit": ["apple", "banna", "peach"]})
db.food.insertOne({"_id": 2, "fruit": ["apple", "kumquat", "orange"]})
db.food.insertOne({"_id": 3, "fruit": ["cherry", "banna", "apple"]})
db.food.find()
```

food 0.001 sec.

```
/* 1 */
{
    "_id" : 1.0,
    "fruit" : [
        "apple",
        "banna",
        "peach"
    ]
}

/* 2 */
{
    "_id" : 2.0,
    "fruit" : [
        "apple",
        "kumquat",
        "orange"
    ]
}

/* 3 */
{
    "_id" : 3.0,
    "fruit" : [
        "cherry",
        "banna",
        "apple"
    ]
}
```

```
db.food.insertOne({"_id": 1, "fruit": ["apple", "banna", "peach"]})
db.food.insertOne({"_id": 2, "fruit": ["apple", "kumquat", "orange"]})
db.food.insertOne({"_id": 3, "fruit": ["cherry", "banna", "apple"]})
db.food.find({"fruit": {$all: ["apple", "banna"]}})
```

food 0.002 sec.

```
/* 1 */
{
    "_id" : 1.0,
    "fruit" : [
        "apple",
        "banna",
        "peach"
    ]
}

/* 2 */
{
    "_id" : 3.0,
    "fruit" : [
        "cherry",
        "banna",
        "apple"
    ]
}
```

> apple과 banna 둘 다 들어있는 도큐먼트(값의 순서는 상관없음)

```
db.food.insertOne({"_id": 1, "fruit": ["apple", "banna", "peach"]})
db.food.insertOne({"_id": 2, "fruit": ["apple", "kumquat", "orange"]})
db.food.insertOne({"_id": 3, "fruit": ["cherry", "banna", "apple"]})
db.food.find({"fruit": ["apple", "banna", "peach"]})
```

food 0.002 sec.

```
/* 1 */
{
    "_id" : 1.0,
    "fruit" : [
        "apple",
        "banna",
        "peach"
    ]
}
```

> 그냥 find를 하게 되면 해당 배열이 동일한 지를 찾으므로 값과 갯수 및 순서에도 영향을 받음

```
db.food.insertOne({"_id": 1, "fruit": ["apple", "banna", "peach"]})
db.food.insertOne({"_id": 2, "fruit": ["apple", "kumquat", "orange"]})
db.food.insertOne({"_id": 3, "fruit": ["cherry", "banna", "apple"]})
db.food.find({"fruit.2": "peach"})
```

food 0.002 sec.

```
/* 1 */
{
    "_id" : 1.0,
    "fruit" : [
        "apple",
        "banna",
        "peach"
    ]
}
```

> 3번 째 배열에 peach가 있는 도큐먼트 찾음(0부터 시작이므로)

```
db.food.insertOne({"_id": 1, "fruit": ["apple", "banna", "peach"]})
db.food.insertOne({"_id": 2, "fruit": ["apple", "kumquat", "orange"]})
db.food.insertOne({"_id": 3, "fruit": ["cherry", "banna", "apple"]})
db.food.find({"fruit": {"$size": 3}})
```

food 0.004 sec.

```
/* 1 */
{
    "_id" : 1.0,
    "fruit" : [
        "apple",
        "banna",
        "peach"
    ]
}

/* 2 */
{
    "_id" : 2.0,
    "fruit" : [
        "apple",
        "kumquat",
        "orange"
    ]
}

/* 3 */
{
    "_id" : 3.0,
    "fruit" : [
        "cherry",
        "banna",
        "apple"
    ]
}
```

> size : 배열의 사이즈, 배열의 항목이 3개인 것

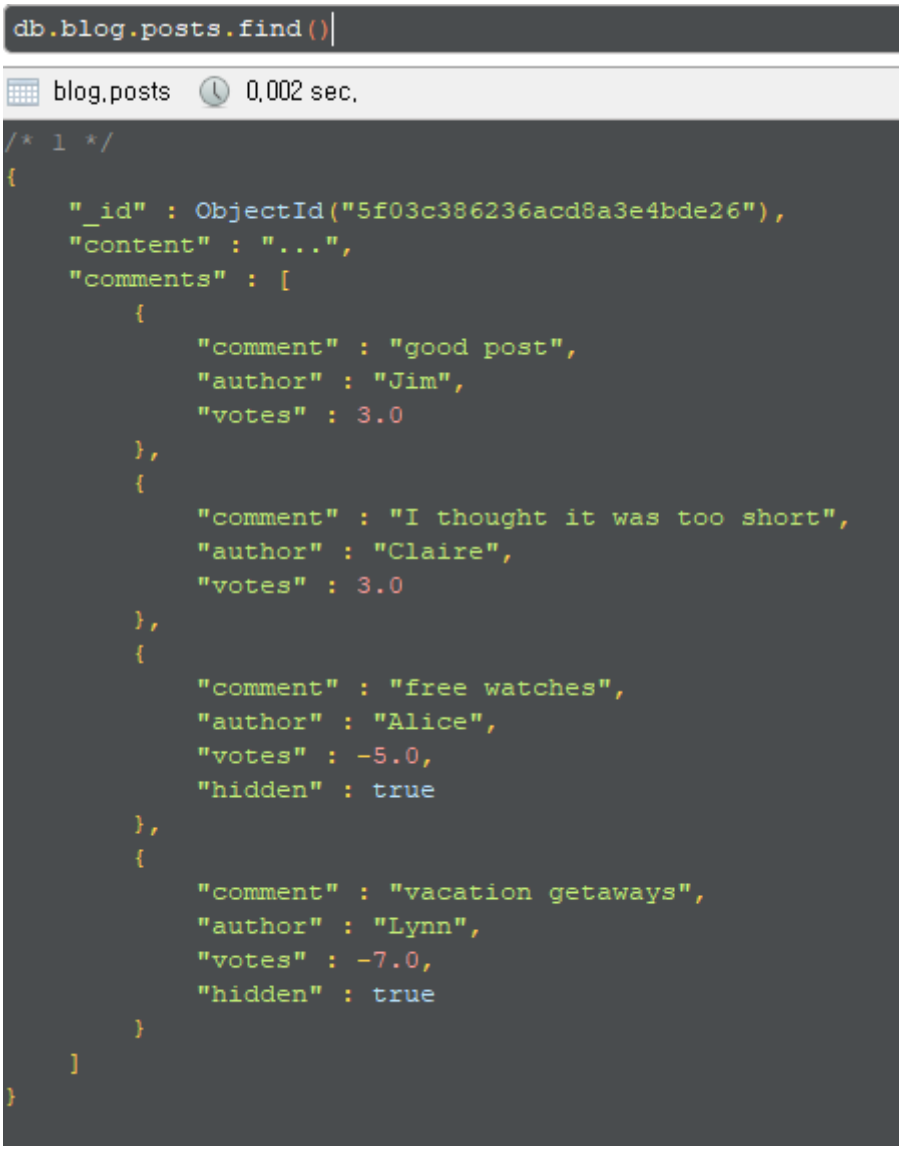
5) slice

- find의 두 번째 인자는 반환값에 대한 것 / "$slice"는 반환되는 배열의 subset을 얻게 함
- 배열의 처음 10개 항목 추출(-10 : 마지막 10개)

```
findOne(criteria, {"comments": {"$slice": 10}})
```

- 처음 23개 항목 건너뛰고 24번 째 부터 10개

```
findOne(criteria, {"comments": {"$slice": [23, 10]}})
```



```
db.blog.posts.find({"comments.author": "Alice"}, {"comments.$": 1})
```

blog.posts 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03c386236acd8a3e4bde26"),
    "comments" : [
        {
            "comment" : "free watches",
            "author" : "Alice",
            "votes" : -5.0,
            "hidden" : true
        }
    ]
}
```

> 1 : JSON형식을 맞추기 위한 값

```
db.array_test.insertMany([
{"x": 5},
{"x": 15},
{"x": 25},
{"x": [5, 25]}
])
db.array_test.find()
```

array_test 0.001 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03f6fc236acd8a3e4bde30"),
    "x" : 5.0
}

/* 2 */
{
    "_id" : ObjectId("5f03f6fc236acd8a3e4bde31"),
    "x" : 15.0
}

/* 3 */
{
    "_id" : ObjectId("5f03f6fc236acd8a3e4bde32"),
    "x" : 25.0
}

/* 4 */
{
    "_id" : ObjectId("5f03f6fc236acd8a3e4bde33"),
    "x" : [
        5.0,
        25.0
    ]
}
```

```
db.array_test.insertMany([
{"x": 5},
{"x": 15},
{"x": 25},
{"x": [5, 25]}
])
db.array_test.find({"x": {"$gt": 10, "$lt": 20}})
```

array_test 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03f6fc236acd8a3e4bde31"),
    "x" : 15.0
}

/* 2 */
{
    "_id" : ObjectId("5f03f6fc236acd8a3e4bde33"),
    "x" : [
        5.0,
        25.0
    ]
}
```

> x가 10보다 크고 20보다 작은 값을 출력 -> 배열도 출력됨
>
> 10보다 크다는 메커니즘을 먼저 돌린 후 20보다 작다는 메커니즘을 돌리므로 두 상황 모두 AND 연산으로 옳은 것 또한 배열 내부 값이기 때문

```
db.array_test.insertMany([
{"x": 5},
{"x": 15},
{"x": 25},
{"x": [5, 25]}
])
db.array_test.find({"x": {"$elemMatch": {"$gt": 10, "$lt": 20}}})
```

0.004 sec.

```
Fetched 0 record(s) in 4ms
```

> elemMatch : 배열 내 항목에 대해 검사하는 연산자
>
> 대신 스칼라(값)을 검사하지 못함
>
> 따라서 배열 도큐먼트들 따로, 스칼라 도큐먼트들 따로 설계해야 함(혹은 스칼라 값들도 배열로 선언하여 대처)

```
db.people.drop()
db.people.insertOne({"name": {"first": "Joe", "last": "Kim"}, "age": 45})
db.people.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03f911236acd8a3e4bde34"),
    "name" : {
        "first" : "Joe",
        "last" : "Kim"
    },
    "age" : 45.0
}
```

```
db.people.findOne({"name": {"first": "Joe", "last": "Kim"}})
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03f911236acd8a3e4bde34"),
    "name" : {
        "first" : "Joe",
        "last" : "Kim"
    },
    "age" : 45.0
}
```

> 위 형식으로 하면 다른 필드가 추가 될 시(수정) 해당 쿼리문은 동작하지 않음

```
db.people.findOne({"name.first": "Joe", "name.last": "Kim"})
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03f911236acd8a3e4bde34"),
    "name" : {
        "first" : "Joe",
        "last" : "Kim"
    },
    "age" : 45.0
}
```

> 도큐먼트의 키에 접근하는 방식으로 하면 가능(임베디드)
>
> 단, 필드 네임은 . 을 쓰면 안됨(필드이기 때문)

```
db.blog.posts.drop()
db.blog.posts.insertOne({"content": "...",
    "comments": [
    {"author": "kim", "score": 3, "comment": "nice post"},
    {"author": "park", "score": 6, "commetn": "terrible post"}
    ]
})
db.blog.posts.findOne()
```

0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03fa75236acd8a3e4bde35"),
    "content" : "...",
    "comments" : [
        {
            "author" : "kim",
            "score" : 3.0,
            "comment" : "nice post"
        },
        {
            "author" : "park",
            "score" : 6.0,
            "commetn" : "terrible post"
        }
    ]
}
```

```
db.blog.posts.find({"comments": {"author": "kim", "score": {"$gte": 5}}})
```

0.002 sec.

```
Fetched 0 record(s) in 2ms
```

> 원래 대로라면 출력이 되지 않음

```
db.blog.posts.find({"comments.author": "kim", "comments.score": {"$gte": 5}})
```

blog.posts 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03fa75236acd8a3e4bde35"),
    "content" : "...",
    "comments" : [
        {
            "author" : "kim",
            "score" : 3.0,
            "comment" : "nice post"
        },
        {
            "author" : "park",
            "score" : 6.0,
            "commetn" : "terrible post"
        }
    ]
}
```

> 검색이 되게 출력 됨(앞의 fruit와 동일한 문제)

```
db.blog.posts.find({"comments": {"$elemMatch": {"author": "kim", "score": {"$gte": 5}}}})
```

```
0.002 sec.

Fetched 0 record(s) in 1ms
```

> elemMatch로 항목별로 검사

3. where Query

- 기존의 키/벨류 표현으로 처리할 수 없는 경우 $where 쿼리를 사용
- 임의의 javascript를 사용할 수 있게 함
- 하지만 보안상의 이유로 유저의 "$where"의 사용이 엄격히 제한되거나 아예 사용할 수 없게 막아 버리는 경우도 있음

```
db.foo.insertOne({"apple": 1, "banna": 6, "peach": 3})
db.foo.insertOne({"apple": 8, "spinach": 4, "watermelon": 4})
db.foo.find()
```

```
foo  0.002 sec.

/* 1 */
{
    "_id" : ObjectId("5f03fda8236acd8a3e4bde36"),
    "apple" : 1.0,
    "banna" : 6.0,
    "peach" : 3.0
}

/* 2 */
{
    "_id" : ObjectId("5f03fdaa236acd8a3e4bde37"),
    "apple" : 8.0,
    "spinach" : 4.0,
    "watermelon" : 4.0
}
```

```
db.foo.insertOne({"apple": 1, "banna": 6, "peach": 3})
db.foo.insertOne({"apple": 8, "spinach": 4, "watermelon": 4})
db.foo.find({"$where": function(){
    for(var current in this){
        for(var other in this){
            if(current != other && this[current] == this[other]){
                return true;
            }
        }
    }
    return false;
}});
```

foo 0.093 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f03fdaa236acd8a3e4bde37"),
    "apple" : 8.0,
    "spinach" : 4.0,
    "watermelon" : 4.0
}
```

> current는 필드 / this는 도큐먼트

forEach / limit / skip

```
for(i = 0; i < 100; i++){
    db.test_collection.insertOne({x : i});
}
db.test_collection.find()
```

test_collection 0.002 sec.

```
/* 1 */
{
    "_id" : ObjectId("5f040327236acd8a3e4bde38"),
    "x" : 0.0
}

/* 2 */
{
    "_id" : ObjectId("5f040327236acd8a3e4bde39"),
    "x" : 1.0
}

/* 3 */
{
    "_id" : ObjectId("5f040327236acd8a3e4bde3a"),
    "x" : 2.0
}

/* 4 */
{
    "_id" : ObjectId("5f040327236acd8a3e4bde3b"),
```

```
var cursor = db.test_collection.find().sort({"x": 1}).limit(10).skip(10);
```

> sort / 리미트 / 스킵 모두 커서 객체에 있는 함수
>
> 함수의 순서는 상관 없음(내부적으로 우선순위가 정해져 있음)

```
var cursor = db.test_collection.find().sort({"x": 1}).limit(10).skip(10);
cursor.forEach(function(x){
    print(x.x);
})
```

0.003 sec.

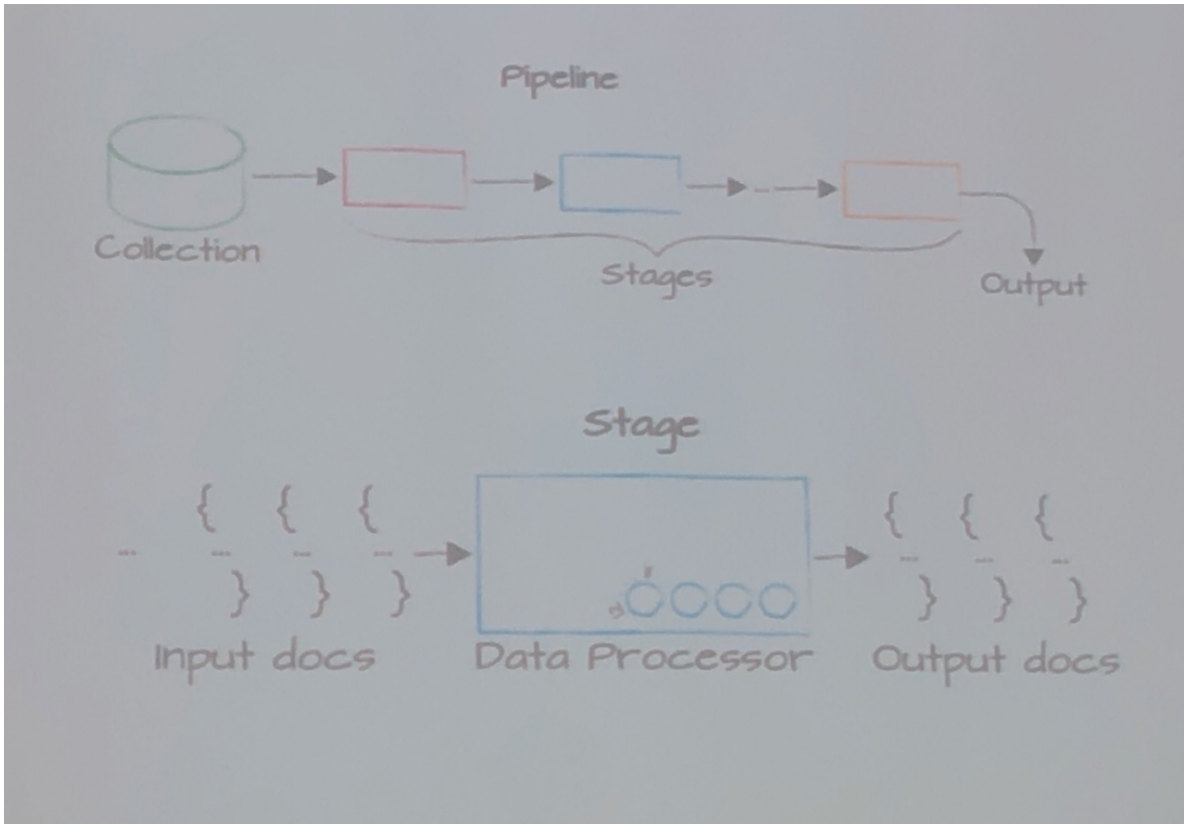
```
10
11
12
13
14
15
16
17
18
19
```

> print(도큐먼트.필드)

```
var cursor = db.test_collection.find().sort({"x": 1}).limit(10).skip(10);
cursor.forEach(function(x){
    print(x.x);
})
```

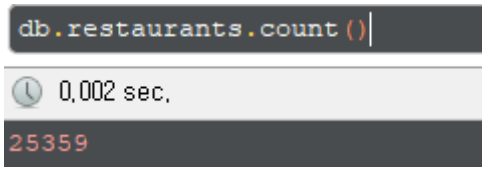
> 다시 한 번 실행하면 값이 없음

mongoDB aggregartion 프레임 워크



> 한 개 이상의 스테이지로 파이프라인을 구성함

restaurants.json 실습



> 데이터가 25359개라는 뜻

```
db.restaurants.findOne()
```

0.002 sec.


```
/* 1 */
{
    "_id" : ObjectId("5f040a018b6d6655e5d47266"),
    "address" : {
        "building" : "469",
        "coord" : [
            -73.961704,
            40.662942
        ],
        "street" : "Flatbush Avenue",
        "zipcode" : "11225"
    },
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "grades" : [
        {
            "date" : ISODate("2014-12-30T00:00:00.000Z"),
            "grade" : "A",
            "score" : 8
        },
        {
            "date" : ISODate("2014-07-01T00:00:00.000Z"),
            "grade" : "B",
            "score" : 23
        },
        {
            "date" : ISODate("2013-04-30T00:00:00.000Z"),
            "grade" : "A",
            "score" : 12
        },
        {
            "date" : ISODate("2012-05-08T00:00:00.000Z"),
            "grade" : "A",
            "score" : 12
        }
    ],
    "name" : "Wendy'S",
    "restaurant_id" : "30112340"
}
```


match 스테이지

```
db.restaurants.aggregate([{$match: {cuisine: "Hamburgers"}}, ])
```

restaurants 0.011 sec.


```
/* 1 */
{
    "_id" : ObjectId("5f040a018b6d6655e5d47266"),
    "address" : {
        "building" : "469",
        "coord" : [
            -73.961704,
            40.662942
        ],
        "street" : "Flatbush Avenue",
        "zipcode" : "11225"
    },
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "grades" : [
        {
            "date" : ISODate("2014-12-30T00:00:00.000Z"),
            "grade" : "A",
            "score" : 8
        },
        {
            "date" : ISODate("2014-07-01T00:00:00.000Z"),
            "grade" : "B",
            "score" : 23
        },
        {
            "date" : ISODate("2013-04-30T00:00:00.000Z"),
            "grade" : "A",
            "score" : 12
        },
        {
            "date" : ISODate("2012-05-08T00:00:00.000Z"),
            "grade" : "A",
            "score" : 12
        }
    ],
    "name" : "Wendy'S",
    "restaurant_id" : "30112340"
}

/* 2 */
{
    "_id" : ObjectId("5f040a018b6d6655e5d4727f"),
```


> 무조건 배열로 생성

```
db.restaurants.aggregate([{$match: {cuisine: "Hamburgers"}},
{$project: {
    _id: 0,
    cuisine: 1,
    borough: 1,
    name: 1
    }}
])
```

restaurants 0.047 sec.

```
/* 1 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "name" : "Wendy'S"
}

/* 2 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "name" : "White Castle"
}

/* 3 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
```

> projection stage를 전달

```
db.restaurants.aggregate([
{$match: {cuisine: "Hamburgers"}},
{$limit: 10},
{$project: {
    _id: 0,
    cuisine: 1,
    borough: 1,
    name: 1
    }}
])
```

restaurants 0.004 sec.

```
/* 1 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "name" : "Wendy'S"
}

/* 2 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "name" : "White Castle"
}

/* 3 */
{
    "borough" : "Brooklyn",
```

> limit로 stage로 작성

```
db.restaurants.aggregate([
{$match: {cuisine: "Hamburgers"}},
{$sort: {name: 1}},
{$limit: 10},
{$project: {
    _id: 0,
    cuisine: 1,
    borough: 1,
    name: 1
    }}
])
```

restaurants 0.053 sec.

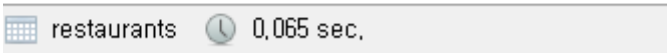
```
/* 1 */
{
    "borough" : "Manhattan",
    "cuisine" : "Hamburgers",
    "name" : "5 Napkin Burger"
}

/* 2 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
    "name" : "67 Burger"
}

/* 3 */
{
    "borough" : "Brooklyn",
    "cuisine" : "Hamburgers",
```

> sort로 스테이지 작성

```
db.restaurants.aggregate([
{$match: {cuisine: "Hamburgers"}},
{$sort: {name: 1}},
{$skip: 10},
{$limit: 10},
{$project: {
    _id: 0,
    cuisine: 1,
    borough: 1,
    name: 1
    }}
])
```




```
/* 1 */
{
    "borough" : "Manhattan",
    "cuisine" : "Hamburgers",
    "name" : "Bill'S Bar & Burgers"
}

/* 2 */
{
    "borough" : "Manhattan",
    "cuisine" : "Hamburgers",
    "name" : "Black Iron Burger"
}

/* 3 */
{
    "borough" : "Manhattan",
    "cuisine" : "Hamburgers",
```

> skip으로 스테이지 작성

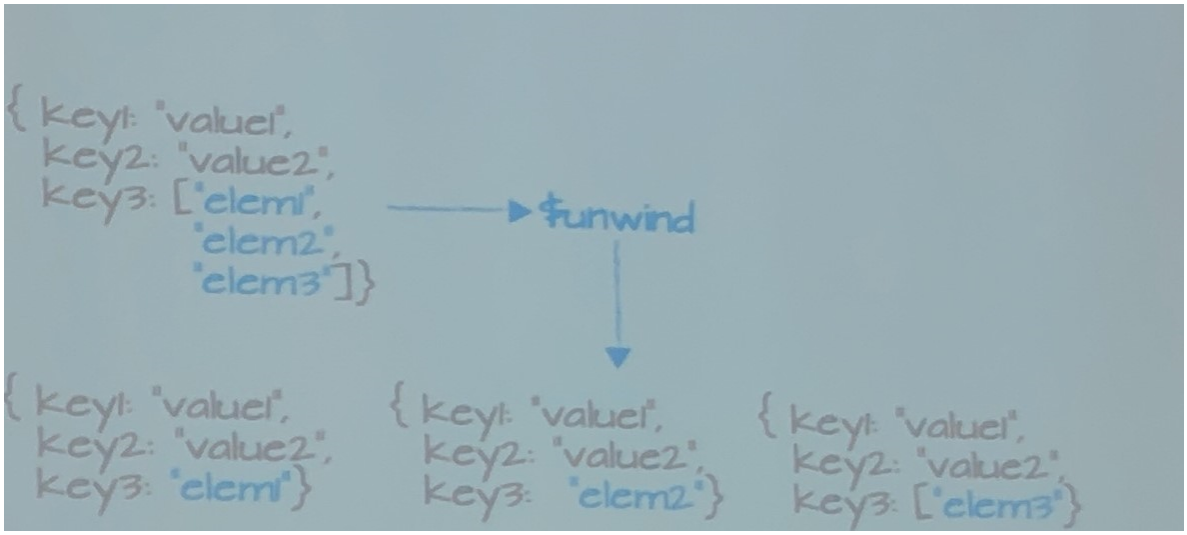
```
db.restaurants.aggregate([
{$match: {"address.street": "Stillwell Avenue"}},
{$project: {
    _id: 0,
    cuisine: 1,
    name: 1,
    dates: "$grades.date",
    valuation: "$grades.grade",
    score: "$grades.score"
    }}
])
```


```
/* 1 */
{
    "cuisine" : "American",
    "name" : "Riviera Caterer",
    "dates" : [
        ISODate("2014-06-10T00:00:00.000Z"),
        ISODate("2013-06-05T00:00:00.000Z"),
        ISODate("2012-04-13T00:00:00.000Z"),
        ISODate("2011-10-12T00:00:00.000Z")
    ],
    "valuation" : [
        "A",
        "A",
        "A",
        "A"
    ],
    "score" : [
        5,
        7,
        12,
        12
    ]
}

/* 2 */
{
```

> Alias 주는 방식으로 작성

unwind




```
db.restaurants.aggregate([
{$match: {"address.street": "Stillwell Avenue"}},
{$unwind: "$grades"},
{$project: {
    _id: 0,
    cuisine: 1,
    name: 1,
    dates: "$grades.date",
    valuation: "$grades.grade",
    score: "$grades.score"
    }}
])
```

restaurants 0.022 sec.

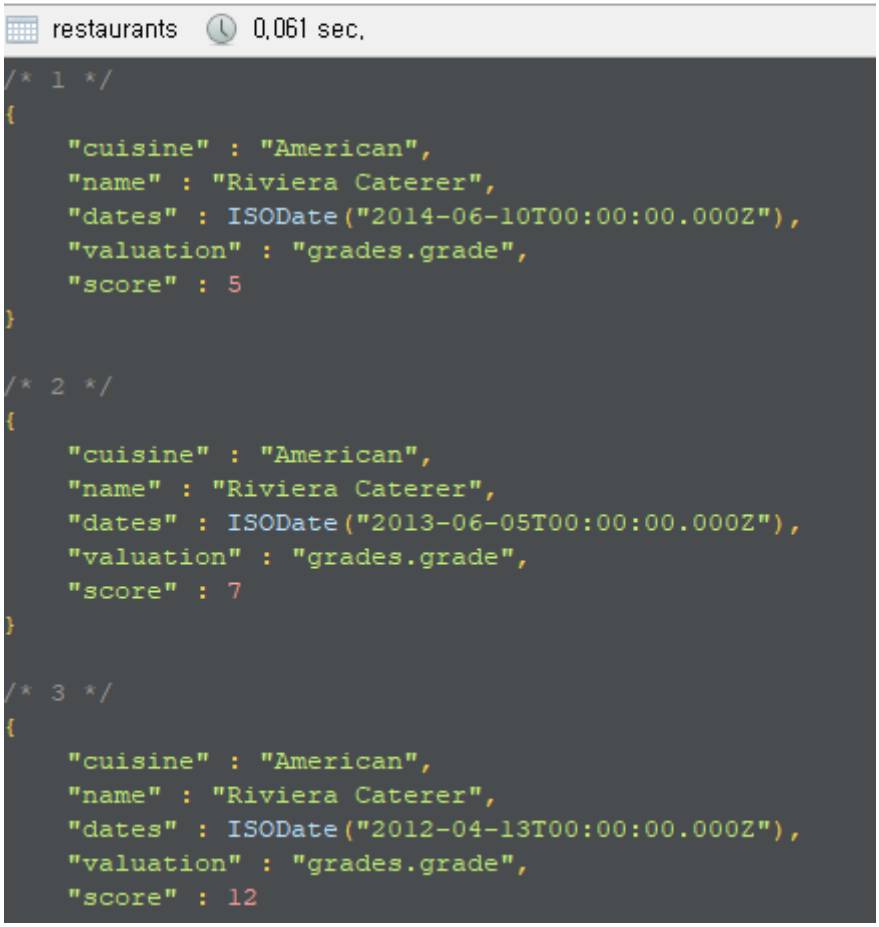
```
/* 1 */
{
    "cuisine" : "American",
    "name" : "Riviera Caterer",
    "dates" : ISODate("2014-06-10T00:00:00.000Z"),
    "valuation" : "A",
    "score" : 5
}

/* 2 */
{
    "cuisine" : "American",
    "name" : "Riviera Caterer",
    "dates" : ISODate("2013-06-05T00:00:00.000Z"),
    "valuation" : "A",
    "score" : 7
}

/* 3 */
{
    "cuisine" : "American",
    "name" : "Riviera Caterer",
    "dates" : ISODate("2012-04-13T00:00:00.000Z"),
    "valuation" : "A",
    "score" : 12
}
```

> unwind로 grades를 쪼갠 출력 화면
>
> aggregation에선 도큐먼트 내의 필드 값을 참조할 땐 반드시 $를 사용

```
db.restaurants.aggregate([
{$match: {"address.street": "Stillwell Avenue"}},
{$unwind: "$grades"},
{$project: {
    _id: 0,
    cuisine: 1,
    name: 1,
    dates: "$grades.date",
    valuation: "grades.grade",
    score: "$grades.score"
    }}
])
```



> $를 사용하지 않았을 때의 출력 화면

```
db.restaurants.aggregate([
{$match: {"cuisine": "Italian", "borough": "Brooklyn"}},
{$project: {
    _id: 0,
    cuisine: 1,
    name: 1,
    praise: {
        $filter: {
            input: "$grades",
            as: "good",
            cond: {$gte: ["$$good.score", 20]}
            }
        }
    }}
])
```

restaurants 0.044 sec.

```
/* 1 */
{
    "cuisine" : "Italian",
    "name" : "Philadelhia Grille Express",
    "praise" : []
}

/* 2 */
{
    "cuisine" : "Italian",
    "name" : "New Corner",
    "praise" : []
}

/* 3 */
{
    "cuisine" : "Italian",
    "name" : "Gargiulo'S Restaurant",
    "praise" : []
}

/* 4 */
{
    "cuisine" : "Italian",
    "name" : "Michael'S Restaurant",
    "praise" : [
        {
            "date" : ISODate("2011-12-01T00:00:00.000Z"),
            "grade" : "B",
            "score" : 20
        }
    ]
}
```

> 변수를 지칭할 땐 $$ 작성
>
> cond : condition(조건)
>
> as : 그냥 만든 변수명
>
> 해당 조건이 없는 도큐먼트의 경우 빈배열이 생성됨

```
db.restaurants.aggregate([
{$match: {"cuisine": "Italian", "borough": "Brooklyn"}},
{$project: {
    _id: 0,
    cuisine: 1,
    name: 1,
    praise: {
        $filter: {
            input: "$grades",
            as: "good",
            cond: {$gte: ["$$good.score", 20]}
            }
        }
    }
   }, {$match: {"praise.0": {"$exists": true}}}
])
```

restaurants 0,063 sec.

```
/* 1 */
{
    "cuisine" : "Italian",
    "name" : "Michael'S Restaurant",
    "praise" : [
        {
            "date" : ISODate("2011-12-01T00:00:00.000Z"),
            "grade" : "B",
            "score" : 20
        }
    ]
}

/* 2 */
{
    "cuisine" : "Italian",
    "name" : "Bamonte'S Restaurant",
    "praise" : [
        {
            "date" : ISODate("2012-12-17T00:00:00.000Z"),
            "grade" : "B",
            "score" : 27
        }
    ]
}
```

> stage를 파이프라인 식으로 연결한 코드(match 사용)
>
> praise의 0번 째가 존재할 경우에만 aggregate(20점 이상)

```
db.restaurants.aggregate([
{$match: {"cuisine": "Italian", "borough": "Brooklyn"}},
{$project: {
    _id: 0,
    cuisine: 1,
    name: 1,
    praise: {
        $filter: {
            input: "$grades",
            as: "good",
            cond: {$gte: ["$$good.score", 20]}
            }
        }
    }
   }, {$match: {"praise.0": {"$exists": true}}}
]).toArray().length
```

0.046 sec.

```
44
```

> 나온 결과를 array로 바꿈 -> length 작성 -> 갯수 확인 가능