7주차 준비하면서

- 준비하면서 주제를 모르면 설명할 수 없지만 너무 많다고 생각되는 부분은 준비하지 못했습니다.
  - geospatil operator
  - cursor collation
- 익숙한 연산자는 예제를 준비하지 않았습니다.
  - comparision operator
  - logical operator
- 이전 스터디 주제와 겹치는 부분 혹은 같은 소주제에서 다루는 경우는 생략하였습니다.
  - cursor method
  - array 검색
  - 서브 도큐먼트

Find

Find 연산자

find 메소드 정의

```
db.collection( query, projection )
```

연산자 분류

분류	operators
query	비교, 논리, 원소, 평가, 배열, 비트
projection	projection ($, $elemMatch, $meta, $slice)
Miscellaneous	Miscellaneous ($comment, $rand)

비교(Comparison) 연산자

op	desc
$eq	equal , =
$gt	greaterThan, >
$gte	greaterThanEqual, >=
$in	in
$lte	lessTheanEqual, <=
$ne	notEqual
$nin	notIn

논리(Logical) 연산자

op	desc

op	desc
$and	and
$not	not and
$nor	not or
$or	or

원소(Element) 연산자

op	desc
$exists	지정된 필드가 존재하는지
$type	지정된 type 의 필드를 가진 다큐먼트

평가(Evaluation) 연산자

op	desc
$expr	쿼리에 aggregation 표현식 을 사용할 수 있다.
$jsonSchema	주어진 JSON 스키마의 유효성을 평가
$mod	
$regex	정규표현식으로 매칭을 시도 한다.
$where	javascript 표현식으로 이용하여 매칭을 시도 한다.

$expr 예제

```
let discountedPrice = {
   $cond: {
      if: { $gte: ["$qty", 100] },
      then: { $multiply: ["$price", NumberDecimal("0.50")] },
      else: { $multiply: ["$price", NumberDecimal("0.75")] }
   }
};

db.supplies.find( { $expr: { $lt:[ discountedPrice,  NumberDecimal("5") ] } })
```

$jsonSchema 예제

```
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "address" ],
         properties: {
            name: {
```

```
                bsonType: "string",
                description: "must be a string and is required"
            },
            year: {
                bsonType: "int",
                minimum: 2017,
                maximum: 3017,
                description: "must be an integer in [ 2017, 3017 ] and is required"
            },
            major: {
                enum: [ "Math", "English", "Computer Science", "History", null ],
                description: "can only be one of the enum values and is required"
            },
            gpa: {
                bsonType: [ "double" ],
                description: "must be a double if the field exists"
            },
            address: {
                bsonType: "object",
                required: [ "city" ],
                properties: {
                    street: {
                        bsonType: "string",
                        description: "must be a string if the field exists"
                    },
                    city: {
                        bsonType: "string",
                        "description": "must be a string and is required"
                    }
                }
            }
        }
    }
})

db.students.insert({
    name: "Alice",
    year: NumberInt(2019),
    major: "History",
    gpa: NumberInt(3), // gpa 는 double 만 받을 수 있으므로 오류
    address: {
        city: "NYC",
        street: "33rd Street"
    }
})

// 결과
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 121,
        "errmsg" : "Document failed validation"
```

```
    }
})
```

$mod 예제

```
{ "_id" : 1, "item" : "abc123", "qty" : 0 }
{ "_id" : 2, "item" : "xyz123", "qty" : 5 }
{ "_id" : 3, "item" : "ijk123", "qty" : 12 }

// qty 를 4로 나누고 나머지가 0 인 경우를 찾는다.
db.inventory.find( { qty: { $mod: [ 4, 0 ] } } )
... ...
{ "_id" : 1, "item" : "abc123", "qty" : 0 }
{ "_id" : 3, "item" : "ijk123", "qty" : 12 }
```

$regex 예제

- MongoDB use Perl compatible regular expressions (i.e. “PCRE”) version 8.42 with UTF-8 support.
- 대소문자를 구분하는 정규식의 경우 필드에 대한 인덱스가 있으면 컬렉션 스캔보다 빠르다.

```
// $in 안에서는 javascript regex 만 사용가능 하다.
{ name: { $in: [ /^acme/i, /^ack/ ] } }
// , 로 구분하여 복수의 컨디션을 사용할 수 있다.
{ name: { $regex: /acme.*corp/, $options: 'i', $nin: [ 'acmeblahcorp' ] } }
// PCRE vs JavaScript PCRE 를 위해서는 문자열로 표현해야 한다.
{ name: { $regex: '(?i)a(?-i)cme' } }
// 4.0.6 ealrier 에서는  regex 객체에서만 사용 가능
db.inventory.find( { item: { $not: /^p.*/ } } )
// 4.0.7 부터는 $not 를 사용할 수 있다.
db.inventory.find( { item: { $not: { $regex: "^p.*" } } } )
db.inventory.find( { item: { $not: { $regex: /^p.*/ } } } )
```

$text 예제

```
// text index 생성 (텍스트 검색을 위한)
db.product.createIndex( { p_text: "text" } )


// $text operator : text index 로 구성된 필드에 대해 검색을 수행함
// $search field : 입력된 문자열을 검색한다.
// "boot " 가 포함된
> db.product.find( { $text : { $search : "boot " } } ).pretty()
// velcro 혹은 belts 가 포함된
> db.product.find( { $text : { $search : "velcro belts" } } ).pretty()
// "velcro belts" 가 포함된
> db.product.find( { $text : { $search : "\"velcro belts\"" } } ).pretty()
// boot 는 포함되지만 velcro 는 포함되지 않는
```

```
> db.product.find( { $text : { $search : "boot -velcro" } } ).pretty()
// bicycles 가 포함된
// { $meta : "textScore" } 관련성 점수
> db.product.find(
    { $text : { $search : "bicycles" } },
    { score : { $meta : "textScore" } }
).pretty()
```

$where 예제

- mongodb 에서 제공하는 native operator 로 해결할 수 없을때만 사용한다.
- mongod 에서 security.javascriptEnabled 를 활성화 한다.
- mongos 에서 security.javascriptEnabled 를 활설화 한다.
- 인덱스를 사용할 수 없다.
- 성능 향상을 위해 $where 에 해당되지 않는 부분은 상위 레벨에서 미리 필터링해주면 좋다.
- nested 다큐먼트에는 사용할 수 없다.
- 전역 변수를 사용하면 안된다.

```
db.players.insertMany([
   { _id: 12378, name: "Steve", username: "steveisawesome", first_login: "2017-01-
01" },
   { _id: 2, name: "Anya", username: "anya", first_login: "2001-02-02" }
])
... ...
db.players.find( { $where: function() {
   return (hex_md5(this.name) == "9b53e667f30cd329dca1ec9e6a83e994")
} } );
... ...
{
   "_id" : 2,
   "name" : "Anya",
   "username" : "anya",
   "first_login" : "2001-02-02"
}
```

지리 공간(Geospatial) 연산자

op	desc
$geoIntersects	GeoJSON geometry 간의 교차되는 geometry 를 구한다.
$geoWithin	GeoJSON geometry 범위 내의 geometry 룰 구한다.
$near	point 에 근접한 geospatial 객체를 구한다.
$nearSphere	sphere 에 근접한 geospatial 객체를 구한다.

배열(Array) 연산자

op	desc
$all	쿼리에 명시된 모든 element 를 포함하는 array 를 매칭한다.
$elemMatch	명시된 element 를 포함하는 array 필드의 다큐먼트를 구한다.
$size	지정된 크기의 array 필드를 가진 다큐먼트를 구한다.

$all 예제

```
{tags: [ "school", "book", "bag", "headphone", "appliance" ]}
{tags: [ "appliance", "school", "book" ]}
{tags: [ "school", "book" ]}
{tags: [ "electronics", "school" ]}
// 지정한 3개를 모두 포함하는 array 를 검색한다.
db.inventory.find( { tags: { $all: [ "appliance", "school", "book" ] } } )
... ...
{tags: [ "school", "book", "bag", "headphone", "appliance" ]}
{tags: [ "appliance", "school", "book" ]}
```

$elemMatch 예제

```
   { "_id": 1, "results": [ { "product": "abc", "score": 10 },
                            { "product": "xyz", "score": 5 } ] },
   { "_id": 2, "results": [ { "product": "abc", "score": 8 },
                            { "product": "xyz", "score": 7 } ] },
   { "_id": 3, "results": [ { "product": "abc", "score": 7 },
                            { "product": "xyz", "score": 8 } ] },
   { "_id": 4, "results": [ { "product": "abc", "score": 7 },
                            { "product": "def", "score": 8 } ] }
... ...
db.survey.find(
   { results: { $elemMatch: { product: "xyz", score: { $gte: 8 } } } }
)
... ...
{ "_id" : 3, "results" : [ { "product" : "abc", "score" : 7 },
                           { "product" : "xyz", "score" : 8 } ] }
```

> $all 에서 필드의 포함이 아닌 조건을 걸고 싶다면 $elemMatch 를 함께 사용해야 한다.

$size 예제

```
db.collection.find( { field: { $size: 2 } } );
```

비트(Bitwise) 연산자

op	desc

op	desc
$bitsAllClear	지정된 position 의 비트가 0 인 것을 찾는다.
$bitsAllSet	지정된 position 의 비트가 1 인 것을 찾는다.
$bitsAnyClear	지정된 position 의 비트가 일부라도 0 인 것을 찾는다.
$bitsAnySet	지정된 position 의 비트가 일부라도 1 인 것을 찾는다.

> 공통적으로 signbit 는 0 이어야 한다. singed int64 로 표현되지 않는 값을 취급되지 않는다.

$bitsAllClear 예제

```
db.collection.save({ _id: 1, a: 54, binaryValueofA: "00110110" })
db.collection.save({ _id: 2, a: 20, binaryValueofA: "00010100" })
db.collection.save({ _id: 3, a: 20.0, binaryValueofA: "00010100" })
db.collection.save({ _id: 4, a: BinData(0, "Zg=="), binaryValueofA: "01100110" })

// bit position array
// 오른쪽 부터 시작하여 0부터 시작하는 position 값을 array 로 입력하여
// 해당 position 의 bit 값이 0 인 값을 찾는다.
//   5   1
// 00010100
db.collection.find( { a: { $bitsAllClear: [ 1, 5 ] } } )
... ...
{ "_id" : 2, "a" : 20, "binaryValueofA" : "00010100" }
{ "_id" : 3, "a" : 20, "binaryValueofA" : "00010100" }

// Integer Bitmask
// position 을 값으로 입력하는 방식
// 35 -> 00100011 즉 0,1,5 번째가 0인 값을 찾는다.
//   5   10
// 00010100
db.collection.find( { a: { $bitsAllClear: 35 } } )
... ...
{ "_id" : 2, "a" : 20, "binaryValueofA" : "00010100" }
{ "_id" : 3, "a" : 20, "binaryValueofA" : "00010100" }

// BinData Bitmask
// > help misc
//     b = new BinData(subtype,base64str)  create a BSON BinData value
// BinData(0, "ID==") -> 0010100
// 해당 position 이 0 이 아닌 1인 값을 찾음???
db.bit.find( { a: { $bitsAllClear: BinData(0, "ID==") } } )
... ...
{ "_id" : 2, "a" : 20, "binaryValueofA" : "00010100" }
{ "_id" : 3, "a" : 20, "binaryValueofA" : "00010100" }
```

$bitsAllSet 예제

```
// $bitsAllSet: [ 1, 5 ] 00100010
// 54                    00110110
// BinData               01100110
db.collection.find( { a: { $bitsAllSet: [ 1, 5 ] } } )
... ...
{ "_id" : 1, "a" : 54, "binaryValueofA" : "00110110" }
{ "_id" : 4, "a" : BinData(0,"Zg=="), "binaryValueofA" : "01100110" }
```

$bitsAnyClear 예제

```
// 35      00100011
// 54      00110110
// 20      00010100
// 20.0    00010100
// bindata 01100110
db.collection.find( { a: { $bitsAnyClear: 35 } } )
... ...
{ "_id" : 1, "a" : 54, "binaryValueofA" : "00110110" }
{ "_id" : 2, "a" : 20, "binaryValueofA" : "00010100" }
{ "_id" : 3, "a" : 20.0, "binaryValueofA" : "00010100" }
{ "_id" : 4, "a" : BinData(0,"Zg=="), "binaryValueofA" : "01100110" }
```

$bitsAnyClear 예제

```
// $bitsAllSet: [ 1, 5 ] 00100010
// 54                    00110110
// BinData               01100110
db.collection.find( { a: { $bitsAnySet: [ 1, 5 ] } } )
{ "_id" : 1, "a" : 54, "binaryValueofA" : "00110110" }
{ "_id" : 4, "a" : BinData(0,"Zg=="), "binaryValueofA" : "01100110" }
```

projection 연산자

op	desc
$	쿼리에 매칭되는 array 의 첫번째 요소를 투영한다.
$elemMatch	쿼리에 매칭되는 다큐먼트에서 $elemMatch 에 매칭되는 array 첫번째 요소를 투영한다.
$meta	$text 연산 중 관련 다큐먼트의 관련성 점수를 투영한다.
$slice	array 에서 투영된 요소에 제한을 한다.

$ 예제

```
{ "_id" : 1, "semester" : 1, "grades" : [ 70, 87, 90 ] }
{ "_id" : 2, "semester" : 1, "grades" : [ 90, 88, 92 ] }
```

```
{ "_id" : 3, "semester" : 1, "grades" : [ 85, 100, 90 ] }
{ "_id" : 4, "semester" : 2, "grades" : [ 79, 85, 80 ] }
{ "_id" : 5, "semester" : 2, "grades" : [ 88, 88, 92 ] }
{ "_id" : 6, "semester" : 2, "grades" : [ 95, 90, 96 ] }
... ...
db.students.find( { semester: 1, grades: { $gte: 85 } },
                  { "grades.$": 1 } )
... ...
{ "_id" : 1, "grades" : [ 87 ] }
{ "_id" : 2, "grades" : [ 90 ] }
{ "_id" : 3, "grades" : [ 85 ] }
```

$elemMatch 예제

```
{
 _id: 1,
 zipcode: "63109",
 students: [
              { name: "john", school: 102, age: 10 },
              { name: "jess", school: 102, age: 11 },
              { name: "jeff", school: 108, age: 15 }
            ]
}
{
 _id: 2,
 zipcode: "63110",
 students: [
              { name: "ajax", school: 100, age: 7 },
              { name: "achilles", school: 100, age: 8 },
            ]
}
{
 _id: 3,
 zipcode: "63109",
 students: [
              { name: "ajax", school: 100, age: 7 },
              { name: "achilles", school: 100, age: 8 },
            ]
}
{
 _id: 4,
 zipcode: "63109",
 students: [
              { name: "barney", school: 102, age: 7 },
              { name: "ruth", school: 102, age: 16 },
            ]
}
... ...
db.schools.find( { zipcode: "63109" },
                 { students: { $elemMatch: { school: 102, age: { $gt: 10} } } } )
... ...
// 쿼리필터에 만족하는 다큐먼트 중에 $elemMatch 매칭되는 array 중 첫번째만 선택된다.
```

```
{ "_id" : 1, "students" : [ { "name" : "jess", "school" : 102, "age" : 11 } ] }
{ "_id" : 3 }
{ "_id" : 4, "students" : [ { "name" : "ruth", "school" : 102, "age" : 16 } ] }
```

$meta 예제

```
예제 이해 못함
```

$slice 예제

```
db.posts.insertMany([
   {
     _id: 1,
     title: "Bagels are not croissants.",
     comments: [ { comment: "0. true" }, { comment: "1. croissants aren't
bagels."} ]
   },
   {
     _id: 2,
     title: "Coffee please.",
     comments: [ { comment: "0. fooey" }, { comment: "1. tea please" }, { comment:
"2. iced coffee" }, { comment: "3. cappuccino" }, { comment: "4. whatever" } ]
   }
])
... ...
db.posts.find( {}, { comments: { $slice: 3 } } )
... ...
{
   "_id" : 1,
   "title" : "Bagels are not croissants.",
   "comments" : [ { "comment" : "0. true" }, { "comment" : "1. croissants aren't
bagels." } ]
}
// comments 의 element 는 5개 이지만 3개까지만 나온다.
{
   "_id" : 2,
   "title" : "Coffee please.",
   "comments" : [ { "comment" : "0. fooey" }, { "comment" : "1. tea please" }, {
"comment" : "2. iced coffee" } ]
}
```

기타(Miscellaneous) 연산자

op	desc
$comment	쿼리에 주석을 남길 수 있다.
$rand	0 ~ 1 (소수점포함) randome 하게 수를 생성한다.

$comment 예제

```
db.records.find(
   {
     x: { $mod: [ 2, 0 ] },
     $comment: "Find even values."
   }
)

// database profiler enable, system.profile collectin
{
  "op" : "query",
  "ns" : "test.records",
  "command" : {
      "find" : "records",
      "filter" : {
         "x" : {
         "$mod" : [
            2,
            0
         ]
      },
      "$comment" : "Find even values."
   },
   "comment" : "Find even values.",
   ...

// profilling level 2, slow 0
// db.setProfilingLevel(2, 0)
// log 에도 남음

{"t":{"$date":"2020-09-17T11:32:20.415-07:00"},"s":"I",
"c":"COMMAND",  "id":51803,   "ctx":"conn7","msg":"Slow query",
"attr":{"type":"command","ns":"test.records","appName":"MongoDB
Shell","command":{"find":"records","filter":{"x":{"$mod":[2.0,0.0]},
"$comment":"Find even values."},"comment":"Find even values."
...
```

Find 조건

서브 도큐먼트의 검색

배열의 검색

Cursor

커서의 옵션 및 명령

Collation

cursor 반환되는 데이터의 정렬 기준을 명시할 수 있다.

filed	type	desc
local	string	The ICU locale. , ko (korean)
strength	integer	비교레벨 설정 ICU Comparison Levels 참고
caseLevel	boolean	
caseFirst	string	
numericOrdering	boolean	
alternate	string	
maxVariable	string	
backwards	boolean	
normalization	boolean	

Read Concern

read concern 은 client 가 replica set 혹은 sharded cluster 에 대해 데이터 일관성에 대하 수준을 정의할 수 있다.

level	desc
local	데이터를 반환 받을때 replica set 의 절반에 기록되 있는것을 보장하지 않는다.
available	local 와 동일하나 샤딩 클러스터에서는 분할된 collection 에 대해 일관성을 보장하지 않는다.
majority	데이터를 반환 받을 때 replica set 의 절반이상의 응답을 받았다는 것을 보장한다.
linearizable	majority와 유사하지만 replica set 의 다수 멤버가 작업이 끝날때 까지 기다린다.
snapshot	???

> operation 별로 지원되는 read concern 레벨이 다르므로 메뉴얼을 참고해야 한다. causally consistent sessions. ???

Read Preference

mongodb client 가 repica set 를 대상으로 read 연산을 어떻게 라우팅할 지를 설정하는 기능이다. Read Preference 는 mode 를 베이스로 아래 세개의 옵션의 조합으로 구성할 수 있다.

- tag set
- MaxStalenessSeconds
- hedged read

Read Preference Mode

Read Preference Mode	Desc

Read Preference Mode	Desc
primary	default, 모든 연산을 primary 에 요청한다., 읽기 연산으로 포함한 multi-document trasaction 은 primary 를 사용해야 한다.
primaryPreferred	가능한 primary 에 요청하며 불가능할 경우 secondary 에 요청한다.
secondary	모든 연산을 secondary 에 요청 한다.
secondaryPreferred	가능한 secondary 에 요청하며 가능한 secondary 멤버가 없는 경우 primary 에 요청 한다.
nearest	타입에 관계없이 latency 가 적은 멤버에 요청한다. localThresholdMS 를 기준으로 평가한다.

tag set

replica set 구성시 "members[n].tags" 를 성정하였다면 read preference 에서 적합멤버를 찾을때 tag 를 이용할 수 있다.

```
// tag 및 value 는 모두 문자열이여야 한다.
[ { "<tag1>": "<string1>", "<tag2>": "<string2>",... }, ... ]
// 예제
[ { "region": "South", "datacenter": "A" },  { "rack": "rack-1" }, { } ]
// Order of Tag Matching
1. { "region": "South", "datacenter": "A" } 로 우선 적합멤버를 찾는다.
2. 못찾을 경우 { "rack": "rack-1" }
3. {} 는 모든 element 로 적합한 멤버를 찾지 못한 경우 오류를 방지하기위해 사용한다.
```

tag set 은 Read Preference Modes 모드 중에서 primary 에서는 적용할 수 없다. 오직 secondary 를 선택할 수 있는 mode 에서만 사용 가능 하다.

사용가능한 모드

- primaryPreferred
- secondary
- secondaryPreferred
- nearest

maxStalenessSeconds

secondary 중에서 지정된 시간 이상으로 replication lag 이 지정값을 초과하는 멤버를 read preference 에서 제거 하기 위한 옵션이다. 과도한 쓰기나 실질적 장애 등이 발생하는 경우에 사용할 수 있다.

사용가능한 모드

- primaryPreferred
- secondary
- secondaryPreferred

- nearest

Hedged Read Option

mongodb 4.4 sharded cluster 에서부터 지원하며 primary 를 제외한 read preference 에서 사용 가능하다. hedged read 옵션을 사용하면 mongos 는 shard 에 대한 replica set 멤버 두개를 지정해서 동시에 요청을 하게 되고 먼저 도착한 결과값을 사용하게 하게 된다.

hedged read 를 지원하는 연산자

- collStats
- count
- dataSize
- dbStats
- distinct
- filemd5
- find
- listCollections
- listIndexes
- planCacheListFilters

사용가능한 모드

- primaryPreferred
- secondary
- secondaryPreferred
- nearest

read preference 사용법

```
# replica set myRepl
# mode secondary
# maxStalenessSeconds 120초
# tag set { "region": "South", "datacenter": "A" }
mongodb://sv1,sv2,sv3/?
replicaSet=myRepl&readPreference=secondary&maxStalenessSeconds=120&readPreferenceT
ags=region:South&readPreferenceTags=datacenter:A
```

> hedged read 는 api 를 통해서만 제공되는 듯 하다.

comment

cursor 를 통해 comment 필드를 추가할 수 있다.

```
// comment 를 명시하면
db.restaurants.find(
   { "borough" : "Manhattan" }
).comment( "Find all Manhattan restaurants" )
```

```
// system.profile
{
   "op" : "query",
   "ns" : "guidebook.restaurant",
   "query" : {
      "find" : "restaurant",
      "filter" : {
         "borough" : "Manhattan"
      },
      "comment" : "Find all Manhattan restaurants"
   },
   ...
}

// mongod log
2015-11-23T13:09:16.202-05:00 I COMMAND  [conn1]
   command guidebook.restaurant command: find {
      find: "restaurant",
      filter: { "borough" : "Manhattan" },
      comment: "Find all Manhattan restaurants"
   }
   ...

// db.currentOp()
db.restaurants.find(
   { "borough" : "Manhattan" }
).comment("Find all Manhattan restaurants")
... ...
{
   "inprog" : [
      {
         "host" : "198.51.100.1:27017",
         "desc" : "conn3",
         "connectionId" : 3,
         ...

         "op" : "query",
         "ns" : "test.$cmd",
         "command" : {
            "find" : "restaurants",
            "filter" : {
               "borough" : "Manhattan"
            },
            "comment" : "Find all Manhattan restaurants",
            "$db" : "test"
         },
         "numYields" : 0,
         ...
      }
   ],
   "ok" : 1
}
```

FindAndModify

FindAndModify