Mongo DB CRUD

CRUD

- Collection의 전체 데이터 조회
  - ✓ db.Collection이름.find(): Collection의 전체 데이터 조회

CRUD

- ❖ 도큐먼트 생성
  - ✓ 특징
    - Mongo DB에서 도큐먼트 생성 동작은 단일 Collection을 대상으로 수행
    - 단일 도큐먼트 레벨에서 원자적으로 실행
    - JSON 표현식으로 데이터를 만들어서 삽입하면 되는데 Collection을 생성하면서 삽입 가능하고 Collection을 먼저 만들고 삽입하는 것도 가능
    - 데이터를 삽입할 때 _id라는 key를 만들지 않고 데이터를 삽입하면 _id라는 key가 자동으로 생성된 후 ObjectId를 부여하는데 _id는 primary key 이면서 Index
    - db.Collection이름.insert(JSON표기법) 함수로 도큐먼트 생성이 가능하며 insert 대신 save 사용이 가능하고 3.4 버전 부터는 insertOne 과 insertMany 지원 – 최신 API 에서는 insert 는 deprecated
    - insert는 동일한 _id를 강제로 삽입하면 에러를 발생시키지만 save는 수정을 수행
    - db.collection.insert({key:value, key:value…}) 형태로 삽입

CRUD

❖ 도큐먼트 생성

```
> use adam
> db.users.insert( { name: "adam", age: 25, gender: "man"})
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6951bc9dd34ae78bf2284462')
  }
}
```

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

Key	Value	Type
(1) ObjectId("607f96828d76b7ddfdd44441")	{ 4 fields }	Object
_id	ObjectId("607f96828d76b7ddfdd44441")	ObjectId
name	adam	String
age	25.0	Double
gender	man	String

CRUD

- 도큐먼트 생성
  - ✓ 객체 안에 객체 나 배열을 가진 도큐먼트 생성

```
> db.inventory.insert(
   {
    item:"ABC1",
    details:{
        model : "14Q3",
        manufacturer : "xyz Company"
     },
    stock:[{size:"s",qty:25},{size:"M",qty:50}],
    category: "clothing"
   }
)
```

CRUD

- 도큐먼트 생성
  - ✓ 객체 안에 객체 나 배열을 가진 도큐먼트 생성

```
> db.inventory.find()
```

Key	Value	Type
(1) ObjectId("607f9e0c8d76b7ddfdd44442")	{ 5 fields }	Object
_id	ObjectId("607f9e0c8d76b7ddfdd44442")	ObjectId
item	ABC1	String
details	{ 2 fields }	Object
model	14Q3	String
manufacturer	xyz Company	String
stock	[2 elements]	Array
[0]	{ 2 fields }	Object
size	s	String
qty	25.0	Double
[1]	{ 2 fields }	Object
size	M	String
qty	50.0	Double
category	clothing	String

CRUD

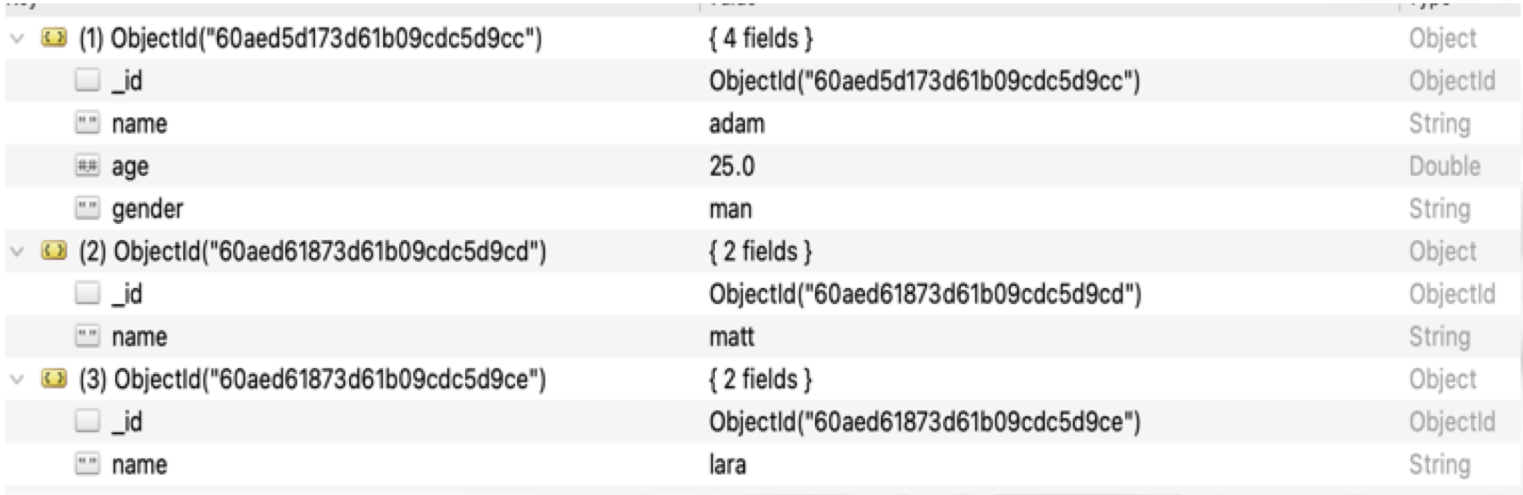
❖ 도큐먼트 생성

✓ 배열 삽입 – 분할해서 각각의 데이터로 삽입

```
> db.users.insert([{name:"matt"}, {name:"lara"}])
```

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

- Mongo DB에서는 최상위 배열을 지원하지 않기 때문에 배열 데이터를 삽입하면 배열의 데이터를 분해해서 각각의 데이터를 삽입



(1) ObjectId("60aed5d173d61b09cdc5d9cc")	{ 4 fields }	Object
_id	ObjectId("60aed5d173d61b09cdc5d9cc")	ObjectId
name	adam	String
age	25.0	Double
gender	man	String
(2) ObjectId("60aed61873d61b09cdc5d9cd")	{ 2 fields }	Object
_id	ObjectId("60aed61873d61b09cdc5d9cd")	ObjectId
name	matt	String
(3) ObjectId("60aed61873d61b09cdc5d9ce")	{ 2 fields }	Object
_id	ObjectId("60aed61873d61b09cdc5d9ce")	ObjectId
name	lara	String

CRUD

- 도큐먼트 생성
  - ✓ 배열 삽입 – 분할해서 각각의 데이터로 삽입

```
> var mydocuments = [
  {item:"ABC2", details:{model : "14Q3",manufacturer : "ABC Company"},
   stock:[{size:"M",qty:50}],
   category: "clothing"
  },
  {item:"ABC3",details:{ model : "14Q3",manufacturer : "CCC Company"},
   stock:[{size:"s",qty:25},{size:"M",qty:50},{size:"L",qty:50}],
   category: "clothing"
  },
  {item:"ABC3",details:{model : "14Q3",manufacturer : "AAA Company"},
   stock:[{size:"s",qty:30}],
   category: "clothing"
   },
]
```

CRUD

- 도큐먼트 생성
  - ✓ 배열 삽입 – 분할해서 각각의 데이터로 삽입

```
> db.inventory.insert(mydocuments)

{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("627998e6c174fa3271682bca"),
     '1': ObjectId("627998e6c174fa3271682bcb"),
     '2': ObjectId("627998e6c174fa3271682bcc") } }
```

CRUD

- 도큐먼트 생성
  - ✓ 배열 삽입 – 분할해서 각각의 데이터로 삽입

```
> db.inventory.find()

{ _id: ObjectId("637c01078ef7c80c4715046b"),
  item: 'ABC1',
  details: { model: '14Q3', manufacturer: 'xyz Company' },
  stock: [ { size: 's', qty: 25 }, { size: 'M', qty: 50 } ],
  category: 'clothing' }
{ _id: ObjectId("637c02168ef7c80c4715046e"),
  item: 'ABC2',
  details: { model: '14Q3', manufacturer: 'ABC Company' },
  stock: [ { size: 'M', qty: 50 } ],
  category: 'clothing' }
```

CRUD

- 도큐먼트 생성
  - ✓ insert 함수의 두번째 매개변수는 생략이 가능한데 ordered 옵션을 설정해서 첫번째 매개변수의 데이터가 배열일 때 싱글 스레드를 이용할지 멀티 스레드를 이용할 지를 설정할 수 있음
  - ✓ {ordered:true} 이면 싱글 스레드를 이용해서 순서대로 저장하고 false를 설정하면 멀티 스레드를 이용해서 저장
  - ✓ 기본값은 true 인데 싱글 스레드를 이용해서 데이터를 삽입하면 중간에 에러가 발생하면 에러가 발생한 지점까지 수행하고 중지되지만 false를 설정하면 어떤 스레드에서 에러가 발생하면 에러가 발생한 스레드는 중지되지만 다른 스레드는 작업을 계속 수행
  - ✓ sample Collection의 name 컬럼을 인덱스로 설정하고 중복 없이 저장하도록 설정

```
> db.sample.createIndex({name:1}, {unique:true})
'name_1'
```

CRUD

❖ 도큐먼트 생성

```
> db.sample.insert({name:"park"})
{ acknowledged: true,
  insertedIds: { '0': ObjectId("62799974c174fa3271682bcd") } }

> db.sample.insert([{name:"kim"}, {name:"park"},{name:"lee"}, {name:"choi"}],
    {ordered:true})

E11000 duplicate key error collection: adam.sample index: name_1 dup key: { name:
    "park" }
```

CRUD

❖ 도큐먼트 생성

✓ kim 데이터를 삽입하는 부분까지는 문제가 없어서 실행되었지만 park을 다시 삽입하려고 하는 부분에서 에러가 발생해서 kim 데이터까지 삽입하고 나머지는 삽입하지 않음

db.sample.find()

Key	Value	Type
(1) ObjectId("6080e3368d76b7ddfdd44455")	{ 2 fields }	Object
_id	ObjectId("6080e3368d76b7ddfdd44455")	ObjectId
name	park	String
(2) ObjectId("6080e35d8d76b7ddfdd44456")	{ 2 fields }	Object
_id	ObjectId("6080e35d8d76b7ddfdd44456")	ObjectId
name	kim	String

CRUD

❖ 도큐먼트 생성

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

> db.sample.createIndex({name:1}, {unique:true})
'name_1'

> db.sample.insert({name:"park"})
{ acknowledged: true,
  insertedIds: { '0': ObjectId("627999f5c174fa3271682bd2") } }

> db.sample.insert([{name:"kim"}, {name:"park"},{name:"lee"}, {name:"choi"}],
    {ordered:false})
E11000 duplicate key error collection: adam.sample index: name_1 dup key: { name:
    "park" }
```

CRUD

- 도큐먼트 생성
  - ✓ park이 중복되어서 에러가 발생하지만 독립된 스레드를 이용하기 때문에 다른 데이터는 삽입

```
> db.sample.find()
{ _id: ObjectId("627999f5c174fa3271682bd2"), name: 'park' }
{ _id: ObjectId("62799a1fc174fa3271682bd3"), name: 'kim' }
{ _id: ObjectId("62799a1fc174fa3271682bd5"), name: 'lee' }
{ _id: ObjectId("62799a1fc174fa3271682bd6"), name: 'choi' }
```

  - ✓ 조그마한 실패는 무시하고 최대한 빨리 데이터를 저장하고자 한다면 ordered 옵션을 false로 설정해서 여러 스레드를 이용해서 insert 하는 것이 좋음

CRUD

❖ 도큐먼트 생성

✓ Mongo DB에서는 없는 옵션을 이용해서 삽입, 삭제, 갱신 작업을 수행하면 에러를 발생시키기도 하지만 에러를 발생시키지 않고 작업을 수행하기도 하는데 Mongo DB에서는 필요한 옵션만 추출해서 작업을 수행하기 때문

```
> db.sample.insert({name:"hwang"}, {noption:true})
{ acknowledged: true,
  insertedIds: { '0': ObjectId("62799aa6c174fa3271682bd7") } }


> db.sample.find()
{ _id: ObjectId("627999f5c174fa3271682bd2"), name: 'park' }
{ _id: ObjectId("62799a1fc174fa3271682bd3"), name: 'kim' }
{ _id: ObjectId("62799a1fc174fa3271682bd5"), name: 'lee' }
{ _id: ObjectId("62799a1fc174fa3271682bd6"), name: 'choi' }
{ _id: ObjectId("62799aa6c174fa3271682bd7"), name: 'hwang' }
```

CRUD

- ❖ 도큐먼트 생성
  - ✓ ObjectId
    - MySQL이나 MariaDB에서는 Auto_Increment 또는 Sequence를 이용해서 일련번호를 제공
    - Mongo DB에서는 12byte 로 구성된 ObjectId를 제공
    - _id 필드에 ObjectId를 생성해서 할당하는 방식으로 Insert 될 때 일련번호를 부여하는데 new ObjectId()를 이용해서 삽입될 _id 필드를 생성 가능

```
> var newId = new ObjectId()
> db.sample.insert({_id:newId, name:"user01"})
{ acknowledged: true,
  insertedIds: { '0': ObjectId("62799b07c174fa3271682bd9") } }
```

CRUD

- ❖ 도큐먼트 생성
  - ✓ ObjectId
    - ● _id 필드에 ObjectId를 생성해서 할당하는 방식으로 Insert 될 때 일련번호를 부여하는데 new ObjectId()를 이용해서 삽입될 _id 필드를 생성 가능

```
> db.sample.find()
{ _id: ObjectId("627999f5c174fa3271682bd2"), name: 'park' }
{ _id: ObjectId("62799a1fc174fa3271682bd3"), name: 'kim' }
{ _id: ObjectId("62799a1fc174fa3271682bd5"), name: 'lee' }
{ _id: ObjectId("62799a1fc174fa3271682bd6"), name: 'choi' }
{ _id: ObjectId("62799aa6c174fa3271682bd7"), name: 'hwang' }
{ _id: ObjectId("62799b07c174fa3271682bd9"), name: 'user01' }
```

CRUD

❖ 도큐먼트 생성

✓ insertOne

- 하나의 데이터만 삽입하고자 하는 경우 사용
- 형식

```
db.collection.insertOne(
   <document>,
   {
      writeConcern: <document>
   }
)
```

```
{
   "acknowledged" : true,
   "insertedId" : <ObjectId>
}
```

◆ Document: 입력할 데이터

◆ WriteConcern: 선택적 파라미터로 WriteConcern을 사용하고자 하는 경우 사용

◆ ObjectId: 생성한 도큐먼트의 ObjectId 리턴

CRUD

❖ 도큐먼트 생성

✓ insertOne

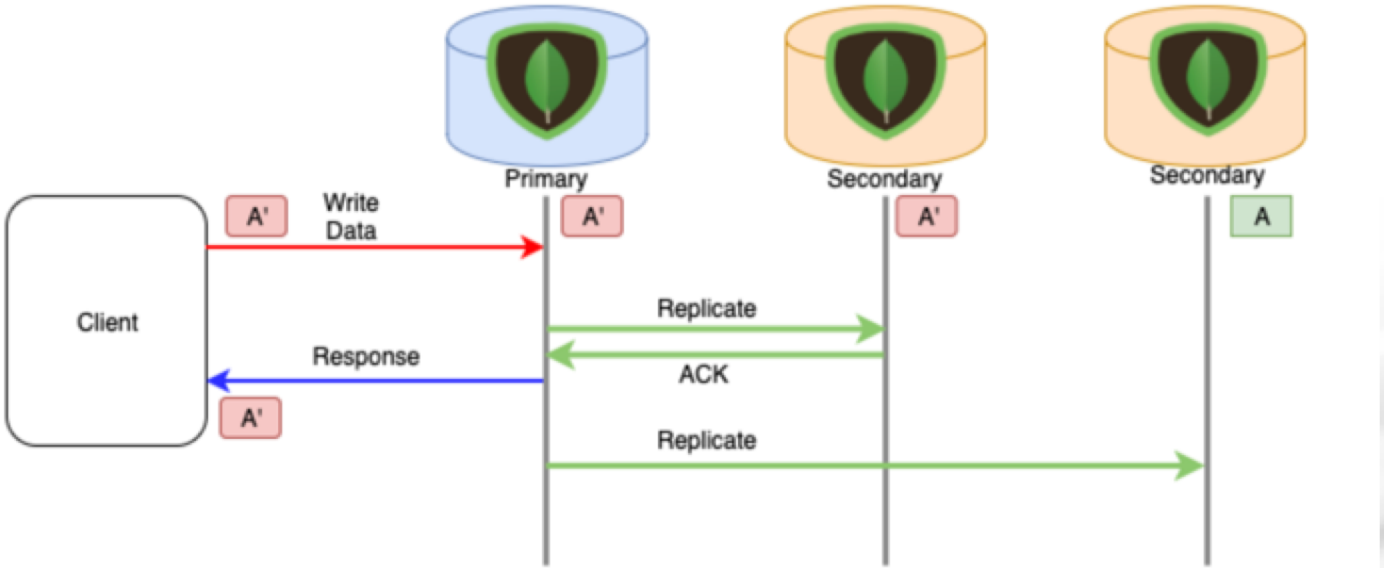
```
db.user.insertOne({username:"karoid",password:"1111"})
{ acknowledged: true,
  insertedId: ObjectId("62799dc7c174fa3271682bda") }

db.user.find().pretty()
{ _id: ObjectId("62799dc7c174fa3271682bda"),
  username: 'karoid',
  password: '1111' }
```

CRUD

- ❖ 도큐먼트 생성
  - ✓ WriteConcern
    - CRUD 작업에 대한 명령어는 거의 대부분 WriteConcern 또는 ReadConcern 옵션을 설정할 수 있음
    - WriteConcern 옵션은 insertOne 명령어가 도큐먼트를 컬렉션에 넣으려고 할 때 이미 이 컬렉션이 쓰기(write) 작업을 하고 있다면 어떤 영향을 줄지(concern)에 대한 것을 설정하는 옵션
    - Mongo DB는 CRUD 작업을 성공적으로 마쳤다고 결과를 반환해도 실제로 반영구적 저장 장치에 바로 데이터를 저장하지 않고 메모리에 저장한 후 천천히 반영구적 저장 장치로 데이터를 옮기기 때문에 데이터베이스에 갑자기 문제가 생기면 메모리에만 담겨 있는 데이터를 잃어 버리게 됨
    - WriteConcern 값은 이러한 데이터 손실이 일어나는 최악의 상황을 어느 정도 수준으로 막을지 결정하는 옵션

CRUD

- ❖ 도큐먼트 생성
  - ✓ WriteConcern
    - Primary 클러스터와 Secondary 클러스터들 간의 Sync가 다 완료된 이후에 Client에게 데이터를 반환해 일관성을 보장해 주는 효과를 가질 수 있음



CRUD

- ❖ 도큐먼트 생성
  - ✓ insertMany
    - ● 여러 개의 데이터를 삽입하고자 하는 경우 사용

```
db.collection.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   }
)
```

```
{
   "acknowledged" : true,
   "insertedIds" : [<ObjectId>, <ObjectId>, ... ]
}
```

CRUD

- ❖ 도큐먼트 생성
  - ✓ insertMany
    - ● 파라미터 와 리턴 데이터
      - ◆ document 1,2: 입력할 도큐먼트를 입력
      - ◆ WriteConcern
      - ◆ ordered: 순서대로 입력할 지 여부
      - ◆ ObjectId

CRUD

❖ 도큐먼트 생성

✓ insertMany

```
db.user.insertMany([
{username:"John", password:4321},
{username:"K", password:4221},
{username:"Mark", password:5321}
])
```

```
{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("6279a1b1c174fa3271682bdb"),
     '1': ObjectId("6279a1b1c174fa3271682bdc"),
     '2': ObjectId("6279a1b1c174fa3271682bdd") } }
```

CRUD

- 도큐먼트 생성
  - ✓ insertMany
    - 몽고DB의 현재 버전은 48메가바이트보다 큰 메시지를 허용하지 않으므로 한 번에 일괄 삽입할 수 있는 데이터의 크기에는 제한이 있는데 48메가바이트보다 큰 삽입을 시도하면 많은 드라이버는 삽입된 데이터를 48메가바이트 크기의 일괄 삽입 여러 개로 분할해서 수행
    - 에러 발생 – WriteError 대신에 BulkWriteError 객체로 리턴
      - WriteConcernErrors: WriteConcern 관련된 에러
      - nInserted: 도큐먼트 생성에 성공한 횟수
      - nUpserted: upsert 에 성공한 횟수
      - nMatched: 도큐먼트 조회에 성공한 횟수
      - nRemoved: 도큐먼트 제거에 성공한 횟수
      - upserted: upsert 작업이 완료된 도큐먼트에 대한 정보

CRUD

- 도큐먼트 생성
  - ✓ insertMany

```
> db.myCollections.insertMany( [
{ _id: 13, item: "envelopes", qty: 60 },{ _id: 13, item: "stamps", qty: 110 },{ _id: 14,
    item: "packing tape", qty: 38 } ] );
```

```
MongoBulkWriteError: E11000 duplicate key error collection: adam.myCollections
    index: _id_ dup key: { _id: 13 }
```

CRUD

- 도큐먼트 생성
  - ✓ 자바스크립트의 반복문 사용 가능

```
> var num=1
for(var i=0; i<3; i++){ db.sample.insertOne({name:"user"+i, score:num})}

{
    "acknowledged" : true,
    "insertedId" : ObjectId("5e080f85a4d6e94519c6aeac")
}

> db.sample.find()

{ _id: ObjectId("6279a43ec174fa3271682bde"),
  name: 'user0',
  score: 1 }
{ _id: ObjectId("6279a43ec174fa3271682bdf"),
  name: 'user1',
  score: 1 }
{ _id: ObjectId("6279a43ec174fa3271682be0"),
  name: 'user2',
  score: 1 }
```

CRUD

- ❖ 도큐먼트 생성
  - ✓ 삽입 유효성 검사
    - 몽고DB는 삽입된 데이터에 최소한의 검사를 수행
    - 도큐먼트의 기본 구조를 검사해 "_id" 필드가 존재하지 않으면 새로 추가하고 모든 도큐먼트는 16메가바이트보다 작아야 하므로 크기를 검사
    - 크기 제한은 다소 상대적인데 는 대개 나쁜 스키마 설계를 예방하고 일관된 성능을 보장하는데 doc라는 도큐먼트의 Binary JSON(BSON)크기를 보려면 셸에서 object.bsonsize(doc)를 실행
    - 모든 주요 언어용 드라이버와 대부분의 비주류 언어용 드라이버는 데이터를 데이터베이스에 보내기 전에 다양한 유효성 검증을 수행하는데 대표적인 것이 도큐먼트가 너무 크지 않은지 UTF-8이 아닌 문자열을 쓰는지 인식할 수 없는 데이터형을 포함하는지 등을 확인

CRUD

- ❖ 데이터 조회
  - ✓ 개요
    - 쿼리 또는 읽기 동작은 Mongo DB에서 데이터를 반환하는 핵심 동작으로 단일 Collection에서 도큐먼트를 선택
    - 쿼리는 도큐먼트에서 필드를 설정하는 프로젝션 기능을 포함할 수 있음
    - 프로젝션 기능을 통해 네트워크 상에서 클라이언트에 반환하는 데이터의 양을 제한
    - Cursor: 쿼리는 전체 결과 셋을 소유하는 커서라고 부르는 반복적인 객체를 리턴
    - 쿼리 최적화: 쿼리 성능을 분석하고 개선
    - 분산 쿼리: 샤드 된 클러스터 와 복제 셋에 대한 쿼리

CRUD

❖ 데이터 조회

✓ find()

```
db.user.find( <query>,
<projection>
)
```

```
<documents>
```

- query: 쿼리 연산자를 이용해서 데이터 선택
- projection: 조회할 필드 이름 나열
- 리턴되는 documents: 조회된 도큐먼트의 커서
- 매개변수가 없으면 컬렉션의 모든 데이터를 조회

CRUD

❖ 데이터 조회

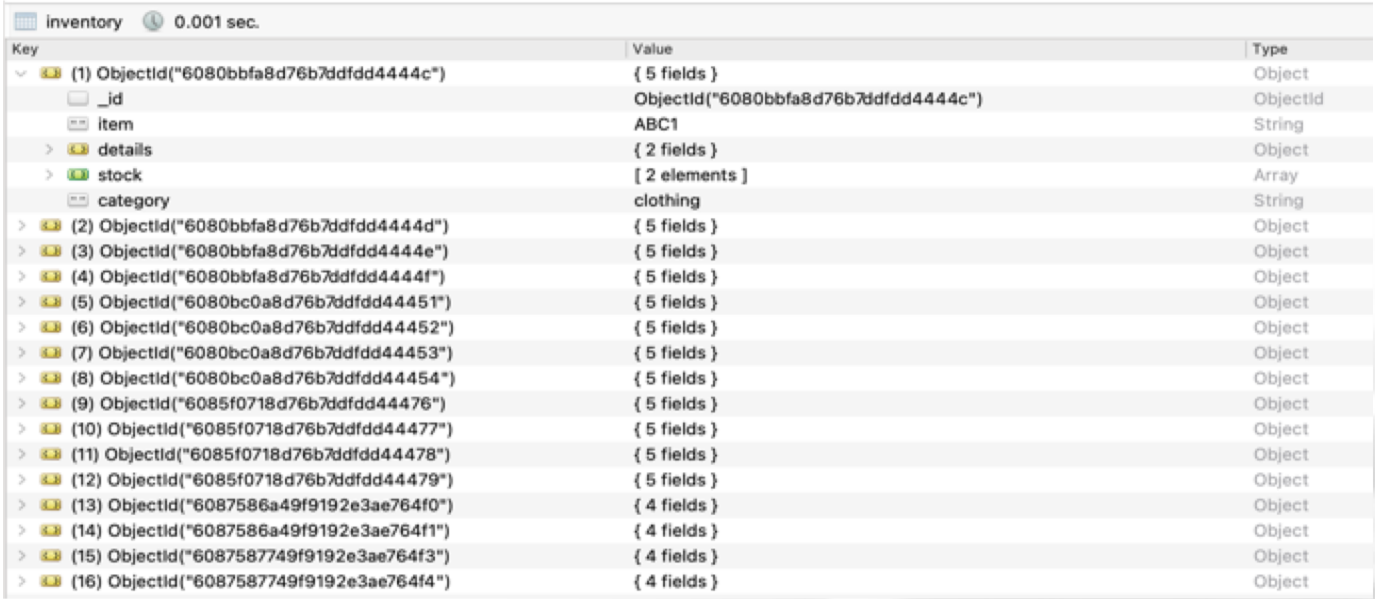
✓ 컬렉션의 전체 데이터 검색

```
> db.users.find()

{ _id: ObjectId("62799767c174fa3271682bc7"),
  name: 'adam',
  age: 25,
  gender: 'man' }
{ _id: ObjectId("627998b8c174fa3271682bc8"), name: 'matt' }
{ _id: ObjectId("627998b8c174fa3271682bc9"), name: 'lara' }
```

CRUD

- 데이터 조회
  - ✓ 컬렉션의 전체 데이터 검색

    > db.inventory.find()



inventory 0.001 sec.

Key	Value	Type
(1) ObjectId("6080bbfa8d76b7ddfdd4444c")	{ 5 fields }	Object
_id	ObjectId("6080bbfa8d76b7ddfdd4444c")	ObjectId
item	ABC1	String
details	{ 2 fields }	Object
stock	[2 elements]	Array
category	clothing	String
(2) ObjectId("6080bbfa8d76b7ddfdd4444d")	{ 5 fields }	Object
(3) ObjectId("6080bbfa8d76b7ddfdd4444e")	{ 5 fields }	Object
(4) ObjectId("6080bbfa8d76b7ddfdd4444f")	{ 5 fields }	Object
(5) ObjectId("6080bc0a8d76b7ddfdd44451")	{ 5 fields }	Object
(6) ObjectId("6080bc0a8d76b7ddfdd44452")	{ 5 fields }	Object
(7) ObjectId("6080bc0a8d76b7ddfdd44453")	{ 5 fields }	Object
(8) ObjectId("6080bc0a8d76b7ddfdd44454")	{ 5 fields }	Object
(9) ObjectId("6085f0718d76b7ddfdd44476")	{ 5 fields }	Object
(10) ObjectId("6085f0718d76b7ddfdd44477")	{ 5 fields }	Object
(11) ObjectId("6085f0718d76b7ddfdd44478")	{ 5 fields }	Object
(12) ObjectId("6085f0718d76b7ddfdd44479")	{ 5 fields }	Object
(13) ObjectId("6087586a49f9192e3ae764f0")	{ 4 fields }	Object
(14) ObjectId("6087586a49f9192e3ae764f1")	{ 4 fields }	Object
(15) ObjectId("6087587749f9192e3ae764f3")	{ 4 fields }	Object
(16) ObjectId("6087587749f9192e3ae764f4")	{ 4 fields }	Object

CRUD

❖ 데이터 조회

✓ 조회를 위한 샘플 데이터 생성 - 터미널에서 수행

```
> mongoimport -d adam -c area < area.json

2022-05-11T15:46:52.234+0900    connected to: Mongo DB://localhost/
2022-05-11T15:46:52.352+0900    228 document(s) imported successfully.
    0 document(s) failed to import.

> mongoimport -d adam -c by_month < by_month.json

2022-05-11T15:48:24.879+0900    connected to: Mongo DB://localhost/
2022-05-11T15:48:25.048+0900    227 document(s) imported successfully.
    0 document(s) failed to import.

> mongoimport -d adam -c by_road_type < by_road_type.json

2022-05-11T15:48:48.129+0900    connected to: Mongo DB://localhost/
2022-05-11T15:48:48.293+0900    227 document(s) imported successfully.
    0 document(s) failed to import.
```

CRUD

❖ 데이터 조회

✓ 조회를 위한 샘플 데이터 생성 - 터미널에서 수행

```
> mongoimport -d adam -c by_type < by_type.json
```

```
2022-05-11T15:51:03.158+0900    connected to: Mongo DB://localhost/
2022-05-11T15:51:03.285+0900    687 document(s) imported successfully.
    0 document(s) failed to import.
```

CRUD

- ❖ 데이터 조회
  - ✓ Docker에 설치 한 경우
    - ● 도커 컨테이너 내부로 파일 복사

      docker cp 파일이름 컨테이너경로:/tmp/복사될파일이름

      - ◆ 데이터 파일 복사

        docker cp area.json mongodb-container:/tmp/area.json

        docker cp month.json mongodb-container:/tmp/month.json

        docker cp by_month.json mongodb-container:/tmp/by_month.json

        docker cp by_road_type.json mongodb-container:/tmp/by_road_type.json
    - ● 도커 컨테이너 내부의 셸 접속

      docker exec -it 컨테니어이름 셸종류

      - ◆ 셸 접속

        docker exec -it mongodb-container bash

CRUD

- ❖ 데이터 조회
  - ✓ Docker에 설치 한 경우
    - ● 데이터 복사 명령 수행

```
mongoimport -d adam -c area < /tmp/area.json
mongoimport -d adam -c by_month < /tmp/by_month.json
mongoimport -d adam -c by_road_type < /tmp/by_road_type.json
mongoimport -d adam -c by_type < /tmp/by_type.json
```

CRUD

- ❖ 데이터 조회
  - ✓ 조회를 위한 샘플 데이터 구조
    - area 컬렉션: 전국 각지의 지역 정보와 인구 수를 저장한 컬렉션
      - ◆ city_or_province: 1차 지역 이름으로 시, 도 와 같은 광역 단체 이름
      - ◆ county: 2차 지역 이름으로 시, 군, 구 와 같은 이름
      - ◆ population: 해당 지역의 인구 수를 의미
    - by_month 컬렉션: 각 지역의 교통 사고 통계를 월 별로 저장한 컬렉션
      - ◆ city_or_province, county: area 컬렉션에서 와 같은 의미
      - ◆ areajd: 해당 지역 정보를 갖는 area 컬렉션의 도큐먼트 _id
      - ◆ month_data: 월별 교통사고 통계로 도큐먼트가 요소인 배열을 소유
      - ◆ month: 해당 도큐먼트의 월
      - ◆ accident_count: 총 사고 수
      - ◆ death_toll: 사망자 수
      - ◆ casualties: 부상자 수
      - ◆ heavyjnjury: 중상자 수
      - ◆ lightjnjury: 경상자 수
      - ◆ injury_report: 부상 신고

CRUD

- ❖ 데이터 조회
  - ✓ 조회를 위한 샘플 데이터 구조
    - by_road_type 컬렉션: 각 지역의 도로 형태 별 사고 통
      - city_or_province, county, area_id 필드: by_month 컬렉션과 같은 의미
      - 교량위, 기타단일로, 교차로내, 교차로부근, 횡단보도상, 횡단보도부근, 기타/불명, 터널안 필드들: 도로 형태별 교통사고 정보로 값으로 교통 사고 통계 오브젝트를 소유
      - accident_count, deathjoll, casualties, heavy_injury, lightjnjury, injury_ report 필드: by_month 컬렉션과 같은 의미
    - by_type 컬렉션: 각 지역의 사고 형태 별 사고 통
      - city_or_province, county, area_id 필드: by_month 컬렉션과 같은 의미
      - type 필드: 사고 형태(차 대 차, 차량 단독, 차 대 사람)를 저장
      - accident_count, deathjoll, casualties, heavy_injury, lightjnjury, injury_ report 필드: by_month 컬렉션과 같은 의미

CRUD

❖ 데이터 조회

✓ 조건을 설정한 데이터 검색

● 특정 값과 일치하는 조건 검색은 find() 안에 {<field>:<value>, …} 를 사용하여 조회

```
> db.users.find({name:"adam"})

{ _id: ObjectId("62799767c174fa3271682bc7"),
  name: 'adam',
  age: 25,
  gender: 'man' }
```

CRUD

❖ 데이터 조회

✓ 조건을 설정한 데이터 검색

```
> db.inventory.find({category:"clothing"})

{ _id: ObjectId("627998e6c174fa3271682bca"),
  item: 'ABC2',
  details: { model: '14Q3', manufacturer: 'ABC Company' },
  stock: [ { size: 'M', qty: 50 } ],
  category: 'clothing' }
{ _id: ObjectId("627998e6c174fa3271682bcb"),
  item: 'ABC3',
  details: { model: '14Q3', manufacturer: 'CCC Company' },
  stock:
   [ { size: 's', qty: 25 },
     { size: 'M', qty: 50 },
     { size: 'L', qty: 50 } ],
  category: 'clothing' }
{ _id: ObjectId("627998e6c174fa3271682bcc"),
  item: 'ABC3',
  details: { model: '14Q3', manufacturer: 'AAA Company' },
  stock: [ { size: 's', qty: 30 } ],
  category: 'clothing' }
```

CRUD

- 데이터 조회
  - ✓ 조건을 설정한 데이터 검색

```
> db.containerBox.insertMany([
{name: 'bear', weight: 60, category: 'animal'},
{name: 'bear', weight: 10, category: 'animal'},
{name: 'cat', weight: 2, category:'animal'},
{name: 'phone', weight: 1, category: 'electronic'}])

{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("6279e4315fc4fd5668ae45b6"),
     '1': ObjectId("6279e4315fc4fd5668ae45b7"),
     '2': ObjectId("6279e4315fc4fd5668ae45b8"),
     '3': ObjectId("6279e4315fc4fd5668ae45b9") } }
```

CRUD

- 데이터 조회
  - ✓ 조건을 설정한 데이터 검색

```
> db.containerBox.find({category: 'animal', name: 'bear'})

{ _id: ObjectId("6279e4315fc4fd5668ae45b6"),
  name: 'bear',
  weight: 60,
  category: 'animal' }
{ _id: ObjectId("6279e4315fc4fd5668ae45b7"),
  name: 'bear',
  weight: 10,
  category: 'animal' }
```

CRUD

❖ 데이터 조회

✓ projection – 필드 단위 추출

{field: <Boolean>, field: <Boolean>, …}

- 특정 키만 검색하고자 할 때는 필드 명에 true(1)를 설정하고 제외를 하고자 할 때는 false(0)를 설정하면 되는데 여러 개 일 때는 ,로 구분해서 입력

```
> b.containerBox.find({}, {name:true})

{ _id: ObjectId("6279e4315fc4fd5668ae45b6"), name: 'bear' }
{ _id: ObjectId("6279e4315fc4fd5668ae45b7"), name: 'bear' }
{ _id: ObjectId("6279e4315fc4fd5668ae45b8"), name: 'cat' }
{ _id: ObjectId("6279e4315fc4fd5668ae45b9"), name: 'phone' }
```

CRUD

❖ 데이터 조회

✓ projection – 필드 단위 추출

```
> db.containerBox.find(null,{_id:false, name: 0})
```

```
{ weight: 60, category: 'animal' }
{ weight: 10, category: 'animal' }
{ weight: 2, category: 'animal' }
{ weight: 1, category: 'electronic' }
```

CRUD

- ❖ 데이터 조회
  - ✓ 비교 오퍼레이터
    - $eq: equals
    - $ne: not equal
    - $gt: greather than
    - $gte: greather than or equals
    - $lt: less than
    - $lte: less than or equals
    - $in: 주어진 배열 안에 속한 값
    - $nin: 주어진 배열 안에 속하지 않은 값

      {<field>: {<operator>: <value>}, ...}

      {<field>: {<operator1>: <value>, <operator2>: <value>}, ...}
    - and
      - ◆ 키가 175 이상 180 이하

        {height: {$gte: 175, $lte:180}}
      - ◆ 키는 180센티미터 이상이면서 몸무게는 60킬로그램 이하

        {height: {$gte: 180}, weight: {$lte: 60}}

CRUD

- ❖ 데이터 조회
  - ✓ 비교 오퍼레이터
    - inventory 컬렉션에서 item 컬럼에서 hello 값인 데이터 조회

      db.inventory.find({item: {$eq: "hello"}})
    - inventory 컬렉션에서 tags 컬럼의 값이 blank 나 blue 인 데이터 조회

      db.inventory.find({tags:{$in:["blank", "blue"]}})
    - inventory 컬렉션에서 tags 컬럼의 값이 blank 나 blue 가 아닌 데이터 조회

      db.inventory.find({tags:{$nin:["blank", "blue"]}})
    - users 컬렉션에서 id 컬럼의 값이 admin이 아닌 데이터 조회

      db.users.find({id:{$ne:"admin"}})
    - users 컬렉션에서 id 컬럼의 값이 root 이거나 user01 인 데이터 조회

      db.users.find({id:{$in:["root","user01"]}})
    - inventory 컬렉션에서 tags 컬럼의 값이 영문 소문자로 시작하고 la로 끝나는 데이터 조회

      db.inventory.find({tags: {$in: [/^[a-z]la/]} })
    - inventory 컬렉션에서 tags 컬럼의 값이 b로 시작하지 않는 데이터 조회

      db.inventory.find({tags: {$nin: [/^b/]} })

CRUD

- ❖ 데이터 조회
  - ✓ 논리 결합 오퍼레이터
    - $not: 주어진 조건을 만족하지 않는 데이터 조회
    - $or: 주어진 조건 중 하나라도 만족하는 데이터 조회
    - $and: 주어진 모든 조건을 만족하는 데이터 조회
    - $nor: 주어진 조건 중 하나라도 만족하지 않는 데이터 조회

      {$not:<operator-expression>}

      {$or:[<query>,...],<field>:{<operator>:<value>, ...}, ...}

CRUD

- ❖ 데이터 조회
  - ✓ 논리 결합 오퍼레이터
    - inventory 컬렉션에서 qty 가 2보다 크지 않은 데이터 조회

      db.inventory.find({qty:{$not:{$gt:2}}})
    - inventory 컬렉션에서 qty 가 100보다 크거나 qty 가 10 보다 작은 데이터 조회

      db.inventory.find({$or:[{qty:{$gt:100}}, {qty:{$lt:10}}]})
    - inventory 컬렉션에서 price 필드의 값이 0.99 또는 1.99 이면서 sale 필드 값이 true 또는 qty 값이 20 미만 인 데이터 조회

      db.inventory.find({$and:[{$or:[{price:0.99}, {price:1.99}]}, {$or:[{sale:true}, {qty:{$lt:20}}]}]})

CRUD

- 데이터 조회
  - ✓ 논리 결합 연산자

```
db.users.find( {$or: [ {id:"root"}, {status:"D"} ] })

db.users.find({"age":{$gte:50, $lte:60}})
```

CRUD

- ❖ 데이터 조회
  - ✓ NULL 조회
    - ● 데이터 삽입

```
> db.c.insert({ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null})
> db.c.insert({ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1})
> db.c.insert({ "_id" : ObjectId("4ba0f0dfd22aa494fd523623"), "y" : 2})
```

    - ● 데이터 조회

```
> db.c.find({"y" : null})
{
  _id: ObjectId('4ba0f0dfd22aa494fd523621'),
  y: null
}
```

CRUD

❖ 데이터 조회

✓ NULL 조회

- null은 존재하지 않음과 도 일치하기 때문에 키를 갖지 않는 도큐먼트도 반환

```
> db.c.find({"z" : null})
{
  _id: ObjectId('4ba0f0dfd22aa494fd523621'),
  y: null
}
{
  _id: ObjectId('4ba0f0dfd22aa494fd523622'),
  y: 1
}
{
  _id: ObjectId('4ba0f0dfd22aa494fd523623'),
  y: 2
}
```

CRUD

- ❖ 데이터 조회
  - ✓ NULL 조회
    - ● 값이 null인 키만 찾고 싶으면 키가 null인 값을 쿼리하고 존재 여부를 확인

```
> db.c.find({"z":{"$eq":null, "$exists":true}})
```

CRUD

- ❖ 데이터 조회
  - ✓ 정규 표현식
    - ● $regex: 값이 정규 표현식 과 동일한 데이터 조회

      {<field>: { $regex: /pattern/, $options:'<options>'}}

      {<field>: { $regex: 'pattern', $options:'<options>'}}

      { <field>: { $regex: /pattern/<options>}}

      - ◆ options 연산자
        - ▪ i: 대소문자 무시
        - ▪ m: 정규식에 ^를 사용할 때 값에 ₩n이 있으면 무시
        - ▪ x: 정규식 안에 있는 모든 공백을 무시
        - ▪ s: .을 사용할 때 ₩n을 포함해서 매치

CRUD

- 데이터 조회
  - ✓ 정규 표현식

```
> db.users.insert({name: 'paulo'})
> db.users.insert({name: 'patric'})
> db.users.insert({name: 'pedro'})
```

CRUD

- 데이터 조회
  - ✓ 정규 표현식

```
> db.users.find({name: /a/})  //like '%a%'
{ "_id" : ObjectId("5ebdf4e51a9cae8745cd221a"), "name" : "paulo" }
{ "_id" : ObjectId("5ebdf4e51a9cae8745cd221b"), "name" : "patric" }
> db.users.find({name: /^pa/}) //like 'pa%'
{ "_id" : ObjectId("5ebdf4e51a9cae8745cd221a"), "name" : "paulo" }
{ "_id" : ObjectId("5ebdf4e51a9cae8745cd221b"), "name" : "patric" }
> db.users.find({name: /ro$/}) //like '%ro'
{ "_id" : ObjectId("5ebdf4e71a9cae8745cd221c"), "name" : "pedro" }
```

CRUD

- ❖ 데이터 조회
  - ✓ $text:
    - 문자열 검색을 수행하는데 해당 컬렉션의 텍스트 인덱스 안에서만 동작
    - 연산자
      - ◆ $search: 검색 내용
      - ◆ $language: 검색할 언어 설정
      - ◆ $caseSensitive: 문자의 대소문자 구분 여부
      - ◆ $diacriticSensitive: 알파벳의 위아래에 붙이는 기호 무시 여부

CRUD

- ❖ 데이터 조회
  - ✓ $text:
    - ● 문자열 검색

```
> db.stores.insertMany( [
{ _id: 1, name: "Java Hut", description: "Coffee and cakes"},
{ _id: 2, name: "Burger Buns", description: "Gourmet hamburgs"}, { _id: 3,
    name: "Coffee Shopping", description: "Just coffee"},
{ _id: 4, name: "Clothes Clothes Clothes", description: "Discount clothing" },
{ _id: 5, name: "Java Shopping", description: "Indonesian goods"}, { _1d: 6,
    name: "영희의 옷", description: "옷이 펄럭거리다"},
{ _id: 7, name: "달리는 철수", description: "철수는 인도인"}])


> db.stores.createIndex({name: "text", description: "text" } )
```

CRUD

- ❖ 데이터 조회
  - ✓ $text:
    - ● 문자열 검색

```
> db.stores.find( { $text: { $search: "coffee" }})

{ _id: 3, name: 'Coffee Shopping', description: 'Just coffee' }
{ _id: 1, name: 'Java Hut', description: 'Coffee and cakes' }

> db.stores.find( { $text: { $search: "bake coffee hamburgs" }}) //or

{ _id: 1, name: 'Java Hut', description: 'Coffee and cakes' }
{ _id: 2, name: "Burger Buns", description: "Gourmet hamburgs"}, { _id: 3,
    name: "Coffee Shopping", description: "Just coffee"}
{ _id: 3, name: 'Coffee Shopping', description: 'Just coffee' }
```

CRUD

- ❖ 데이터 조회
  - ✓ 배열 연산자
    - $all: 순서와 상관없이 배열 안의 요소가 모두 포함되면 선택
    - $elemMatch: 조건과 맞는 배열 속 요소를 가진 Document를 선택
    - $size: 해당 배열의 크기가 같은 Document를 선택

CRUD

- ❖ 데이터 조회
  - ✓ 배열 연산자
    - ● 샘플 데이터 입력

```
> db.inventory.insertMany([
 { item: "journal", qty: 25, tags: ["blank", "red"] },
 { item: "notebook", qty: 50, tags: ["red", "blank"] },
 { item: "paper", qty: 100, tags:[]},
 { item: "planner", qty: 75, tags: ["blank", "red"] },
 { item: "postcard", qty: 45, tags: ["blue"] }
]);
```

CRUD

❖ 데이터 조회

✓ 배열 연산자

● tags 안에 red를 소유하고 있는 데이터 조회

```
> db.inventory.find ({tags: "red"}, {_id: false})
```

```
{ item: 'journal', qty: 25, tags: [ 'blank', 'red' ] }
{ item: 'notebook', qty: 50, tags: [ 'red', 'blank' ] }
{ item: 'planner', qty: 75, tags: [ 'blank', 'red' ] }
```

CRUD

❖ 데이터 조회

✓ 배열 연산자

● tags 안에 red 와 blank를 순서대로 소유하고 있는 데이터 조회

```
> db.inventory.find({tags: ["red", "blank"]}, {_id: false})
```

```
{ item: 'notebook', qty: 50, tags: [ 'red', 'blank' ] }
```

CRUD

- ❖ 데이터 조회
  - ✓ 배열 연산자
    - ● 80보다 큰 데이터가 1개라도 있는지 확인하고 90보다 작은 데이터가 1개라도 있는지 확인

```
> db.users.insert({name:"matt", scores:[79, 85, 93]})
> db.users.insert({name:"lara", scores:[91, 74, 63]})


> db.users.find({scores:{$gt:80, $lt:90}})
```

CRUD

- ❖ 데이터 조회
  - ✓ 배열 연산자
    - ● 배열에서 특정 인덱스 의 데이터를 조회하고자 하는 경우는 $slice
      - ◆ tags 필드의 첫번째 데이터만 조회

        > db.inventory.find({},{tags: {$slice: 1} })
      - ◆ tags 필드의 마지막 2개의 데이터만 조회

        > db.inventory.find({},{tags: {$slice: -2} })
      - ◆ tags 필드의 1 부터 3 까지의 데이터만 조회

        > db.inventory.find({},{tags: {$slice: [1,3]} })

CRUD

- ❖ 데이터 조회
  - ✓ 배열 연산자
    - ● 배열에서 특정 값만 조회 – .$

```
> db.inventory.find({tags: "red"},{"tags.$":true})

{ _id: ObjectId("627c472c5fc4fd5668ae45cb"), tags: [ 'red' ] }
{ _id: ObjectId("627c472c5fc4fd5668ae45cc"), tags: [ 'red' ] }
{ _id: ObjectId("627c472c5fc4fd5668ae45ce"), tags: [ 'red' ] }
```

CRUD

❖ 데이터 조회

✓ 배열 연산자

● 80보다 큰 데이터 나 90보다 작은 데이터가 1개라도 있는지 확인

```
> db.users.find({scores:{$elemMatch:{$gt:80, $lt:90}}})

{ _id: ObjectId("627c50115fc4fd5668ae45d0"),
  name: 'matt',
  scores: [ 79, 85, 93 ] }
```

CRUD

❖ 데이터 조회

✓ 배열 연산자

● tags 안에 red 와 blank를 순서에 상관없이 소유하고 있는 데이터 조회

```
> db.inventory.find( { tags: { $all: ["red", "blank"]} }, {_id: false})
```

```
{ item: 'journal', qty: 25, tags: [ 'blank', 'red' ] }
{ item: 'notebook', qty: 50, tags: [ 'red', 'blank' ] }
{ item: 'planner', qty: 75, tags: [ 'blank', 'red' ] }
```

CRUD

- ❖ 데이터 조회
  - ✓ 배열 연산자
    - ● tags 안에 데이터가 0개 있는 데이터 조회

```
> db.inventory.find( { "tags": { $size: 0 } }, {_id: false} )
```

```
{ item: 'paper', qty: 100, tags: [] }
```

CRUD

- ❖ 데이터 조회
  - ✓ 필드 메타 오퍼레이터
    - ● $exist: 필드 소유 여부

```
> db.inventory.find( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
```

CRUD

- ❖ 데이터 조회
  - ✓ 필드 메타 오퍼레이터
    - ● $type: 데이터 타입 확인

```
> db.addressBook.insertMany(
   [
     { "_id" : 1, address : "2030 Martian Way", zipCode : "90698345" },
     { "_id" : 2, address: "156 Lunar Place", zipCode : 43339374 },
     { "_id" : 3, address : "2324 Pluto Place", zipCode: NumberLong(3921412) },
     { "_id" : 4, address : "55 Saturn Ring" , zipCode : NumberInt(88602117) },
     { "_id" : 5, address : "104 Venus Drive", zipCode : ["834847278",
    "1893289032"]}
   ]
)
```

CRUD

❖ 데이터 조회

✓ 필드 메타 오퍼레이터

- $type: 데이터 타입 확인

```
> db.addressBook.find( { "zipCode" : { $type : "string" } } );

{ _id: 1, address: '2030 Martian Way', zipCode: '90698345' }
{ _id: 5,
  address: '104 Venus Drive',
  zipCode: [ '834847278', '1893289032' ] }
```

CRUD

❖ 데이터 조회

✓ $mod: 나머지 연산자

```
> db.inventory.insertMany( [
   { "_id" : 1, "item" : "abc123", "qty" : 0 },
   { "_id" : 2, "item" : "xyz123", "qty" : 5 },
   { "_id" : 3, "item" : "ijk123", "qty" : 12 }
] )
> db.inventory.find( { qty: { $mod: [ 4, 0 ] } } )

{ _id: ObjectId("627c472c5fc4fd5668ae45cd"),
  item: 'paper',
  qty: 100,
  tags: [] }
{ _id: 1, item: 'abc123', qty: 0 }
{ _id: 3, item: 'ijk123', qty: 12 }
```

CRUD

- ❖ 데이터 조회
  - ✓ $where
    - MongoDB에서 $where 연산자는 일반적인 쿼리 연산자(예: $gt, $lt, $mod 등)로 표현하기 어려운 복잡한 논리를 자바스크립트(JavaScript) 표현식이나 함수를 통해 실행할 때 사용
    - 데이터베이스 내에서 자바스크립트 코드를 돌려 문서를 필터링하겠다는 의미

CRUD

- 데이터 조회
  - ✓ $where

```
> 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' }
```

CRUD

- 데이터 조회
  - ✓ 서브 도큐먼트 필드 검색 쿼리

```
> db.users.insert({name:"matt", contact:{type:"office", phone:"031-000-0000"}})

> db.users.find({contact:{type:"office", phone:"031-000-0000"}})

> db.users.find({"contact.type":"office", "contact.phone":"031-000-0000"})
```

CRUD

❖ 데이터 조회

✓ 데이터 개수 제한은 limit()

```
> db.users.find().limit(2)
```

```
{ "_id" : ObjectId("55d2667d30321643a2c1aa22"), "id" : "user01", "password" : "u
ser01" }
{ "_id" : ObjectId("55d2667330321643a2c1aa21"), "id" : "root", "password" : "sys
tem" }
```

CRUD

❖ 데이터 조회

✓ 데이터 1개 검색은 findOne()

```
> db.users.findOne()

{
        "_id" : ObjectId("55d2667d30321643a2c1aa22"),
        "id" : "user01",
        "password" : "user01"
}
```

CRUD

- 데이터 조회
  - ✓ 건너뛰기는 skip()

```
> db.users.find().skip(1)
```

```
{ "_id" : ObjectId("55d26673303216643a2c1aa21"), "id" : "root", "password" : "sys
tem" }
```

CRUD

❖ 데이터 조회

✓ 정렬은 sort() – 매개변수의 값으로 -1을 주면 내림차순

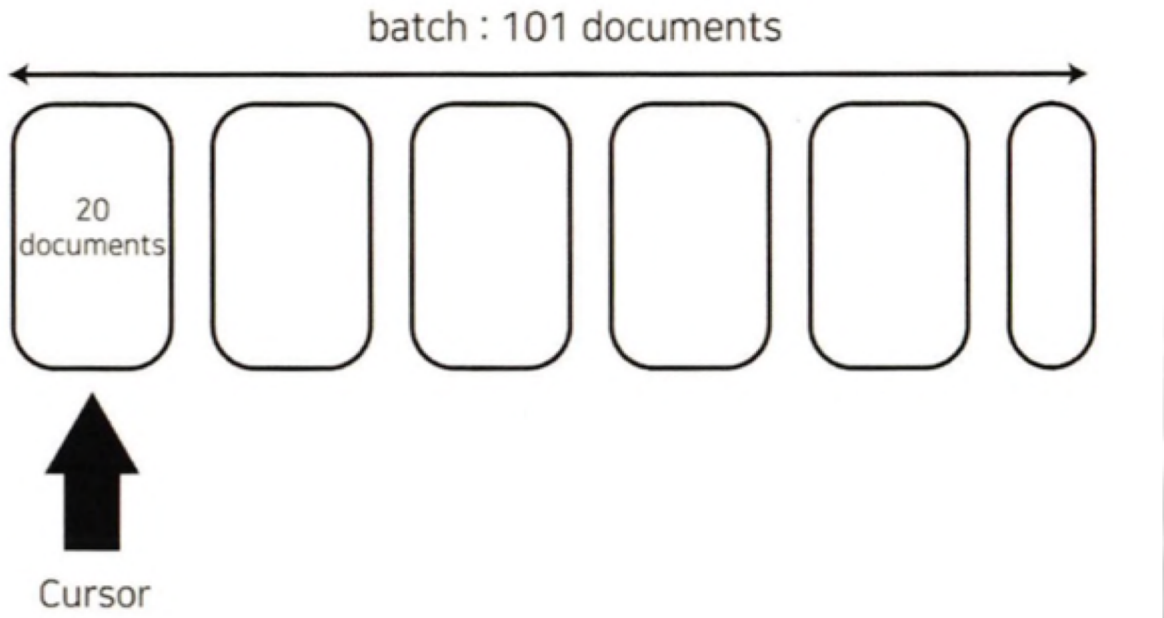
```
> db.users.find().sort({id:1})
```

```
{ "_id" : ObjectId("55d26673303216 43a2c1aa21"), "id" : "root", "password" : "sys
tem" }
{ "_id" : ObjectId("55d2667d303216 43a2c1aa22"), "id" : "user01", "password" : "u
ser01" }
```

- $natural:1 을 설정하면 삽입한 순서대로 오름차순 정렬하고 -1이면 내림차순 정렬

CRUD

- 데이터 조회
  - ✓ Cursor
    - 커서(Cursor)는 쿼리 결과에 대한 포인터로 find 명령어는 결과로 도큐먼트를 직접 반환하지 않고 더 나은 성능을 위해서 커서를 반환



CRUD

❖ 데이터 조회

✓ Cursor

- hasNext()를 이용해서 다음 데이터 존재 여부를 판단하고 next로 다음 데이터에 접근

```
> var cursor = db.users.find()
> cursor.hasNext()

true

> cursor.next()

{
  _id: ObjectId("62799767c174fa3271682bc7"),
  name: 'adam',
  age: 25,
  gender: 'man'
}

> cursor.hasNext() ? cursor.next() : null

{ _id: ObjectId("627998b8c174fa3271682bc8"), name: 'matt' }
```

CRUD

- ❖ 데이터 조회
  - ✓ 쿼리 성능
    - ● 샘플 데이터 입력

```
> db.inventory.drop()
> db.inventory.insert({"_id":1, "item":"f1", "type":"food",quantity:500})
> db.inventory.insert({"_id":2, "item":"f2", "type":"food",quantity:100})
> db.inventory.insert({"_id":3, "item":"p1", "type":"paper",quantity:200})
> db.inventory.insert({"_id":4, "item":"p2", "type":"paper",quantity:150})
> db.inventory.insert({"_id":5, "item":"f3", "type":"food",quantity:350})
> db.inventory.insert({"_id":6, "item":"t1", "type":"toys",quantity:500})
> db.inventory.insert({"_id":7, "item":"a1", "type":"apparel",quantity:250})
> db.inventory.insert({"_id":8, "item":"a2", "type":"apparel",quantity:400})
> db.inventory.insert({"_id":9, "item":"t2", "type":"toys",quantity:50})
> db.inventory.insert({"_id":10, "item":"f4", "type":"food",quantity:75})
```

CRUD

❖ 데이터 조회

✓ 쿼리 성능

- quantity 필드 값이 100에서 200사이인 데이터 조회

  > db.inventory.find({quantity:{$gte:100, $lte:200}})

  { "_id" : 2, "item" : "f2", "type" : "food", "quantity" : 100 }
  { "_id" : 3, "item" : "p1", "type" : "paper", "quantity" : 200 }
  { "_id" : 4, "item" : "p2", "type" : "paper", "quantity" : 150 }

- explain("executionStates")를 연달아 호출하면 쿼리 계획을 확인할 수 있음

  > db.inventory.find({quantity:{$gte:100, $lte:200}}).explain("executionStats")

CRUD

- ❖ 데이터 조회
  - ✓ 쿼리 성능
    - 인덱스 생성
    - db.Collection이름.createIndex({필드이름:1})

      > db.inventory.createIndex({quantity:1})

      ```
      {
          "createdCollectionAutomatically" : false,
          "numIndexesBefore" : 1,
          "numIndexesAfter" : 2,
          "ok" : 1
      }
      ```

    - explain(“executionStates”)를 연달아 호출하면 쿼리 계획을 확인할 수 있음

      > db.inventory.find({quantity:{$gte:100, $lte:200}}).explain("executionStats")

CRUD

❖ 데이터 집계

✓ 저장되어 있는 정보에서 다른 정보를 합해서 출력하거나, 그룹화를 통해 다른 형태로 정보를 생성해서 리턴하는 작업

✓ 구현 방법

- 데이터베이스의 모든 정보를 불러와 애플리케이션 단계에서 집계
- Mongo DB의 맵-리듀스 기능을 이용
- Mongo DB의 집계 파이프라인 기능을 이용

✓ 구현 방법에 따른 장단점

	1번: 애플리케이션	2번: 맵-리듀스	3번: 파이프라인
자유도	좋다.	좋다.	나쁘다.
처리 속도	가장 나쁘다.	보통	가장 좋다.
램 사용량	매우 높음	높음	낮음
처리 위치	애플리케이션 내부	자바스크립트 엔진	MongoDB 내부

CRUD

- 데이터 집계
  - ✓ 샘플 데이터 입력

```
> db.products.insert({'name':'iPad 16GB Wifi', 'manufacture':"Apple",
                      'category':'Tablets', 'price':499.00})
> db.products.insert({'name':'iPad 32GB Wifi', 'category':'Tablets',
                      'manufacture':"Apple", 'price':599.00})
> db.products.insert({'name':'iPad 64GB Wifi', 'category':'Tablets',
                      'manufacture':"Apple",  'price':699.00})
> db.products.insert({'name':'Galaxy S3', 'category':'Cell Phones',
                      'manufacture':'Samsung', 'price':563.99})
> db.products.insert({'name':'Galaxy Tab 10', 'category':'Tablets',
                      'manufacture':'Samsung', 'price':450.99})
> db.products.insert({'name':'Vaio', 'category':'Laptops',
                      'manufacture':"Sony", 'price':499.00})
> db.products.insert({'name':'Macbook Air 13inch', 'category':'Laptops',
                      'manufacture':"Apple", 'price':499.00})
```

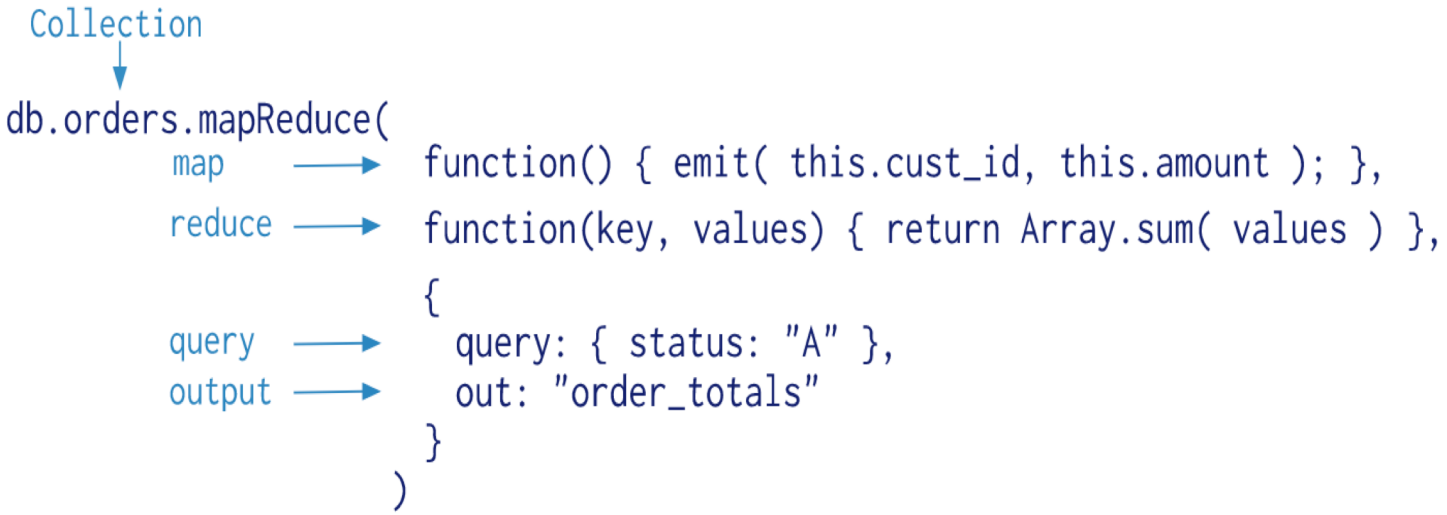
CRUD

- 데이터 집계
  - ✓ 샘플 데이터 입력

```
> db.products.insert({'name':'Nexus 7', 'category':'Tablets',
                      'manufacture':"Google", 'price':199.00})
> db.products.insert({'name':'Kindle Paper White', 'category':'Tablets',
                      'manufacture':"Amazon", 'price':129.00})
> db.products.insert({'name':'Kindle Fire', 'category':'Tablets',
                      'manufacture':"Amazon", 'price':199.00})
```

CRUD

- 데이터 집계
  - ✓ Map Reduce
    - 형식



CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 형식

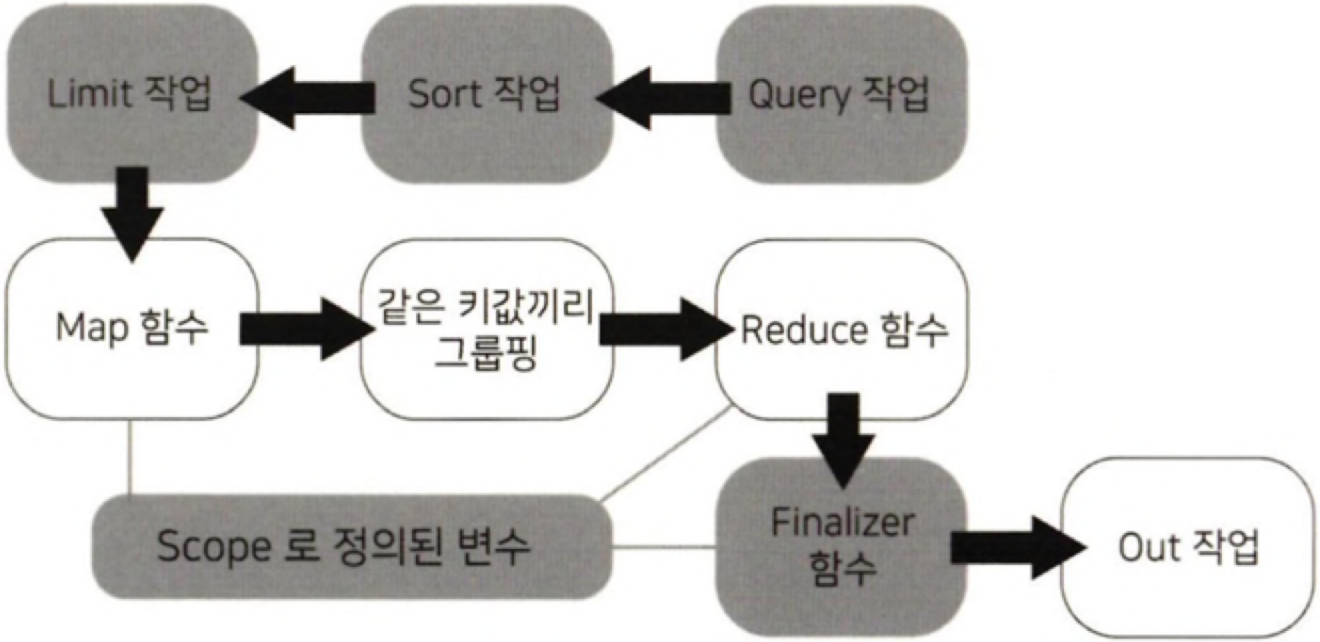
```
db.collection.mapReduce(
    <map>,
    <reduce>,
    {
      out: <collection>,
      query: <document>,
      sort: <document>,
      limit: <number>,
      finalize: <function>,
      scope: <document>,
      jsMode: <boolean>,
      verbose: <boolean>
    }
)
```

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 형식
      - ◆ map: 어떤 정보 들끼리 서로 묶일 수 있을지 정하는 함수
      - ◆ reduce: 수행할 연산 함수
      - ◆ out: 출력된 정보를 데이터베이스 내에 기록으로 남기도록 설정
      - ◆ query: map을 수행하기 전에 필터링할 조건
      - ◆ sort: map을 수행하기 전에 어떤 데이터를 가지고 정렬할 지 설정
      - ◆ limit: 선택적. map 과정을 실행하기 전에 함수에 넣을 도큐먼트의 수를 제한
      - ◆ finalize: reduce 과정이 끝난 후 호출될 함수
      - ◆ scope: 전역 변수 설정
      - ◆ jsMode: map 과 reduce 사이의 정보를 자바스크립트로 남길 지 여부
      - ◆ verbose: 연산 처리에 걸린 시간 출력 여부

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 동작 과정



CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 샘플 데이터 추가

```
> db.rating.insertMany([
  {_id: 1, rating: 1, user_id: 2},
  {_id: 2, rating: 2, user_id: 3},
  {_id: 3, rating: 3, user_id: 4},
  {_id: 4, rating: 3, user_id: 1},
  {_id: 5, rating: 4, user_id: 5},
  {_id: 6, rating: 4, user_id: 8},
  {_id: 7, rating: 5, user_id: 9},
  {_id: 8, rating: 5, user_id: 10},
  {_id: 9, rating: 5, user_id: 11},
  {_id: 10, rating: 5, user_id: 12}
])
```

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● Map 함수

```
var mapper = function(){
        emit(this.rating, this.user_id)
}
```

      - ◆ Map, Reduce, Finalize 함수에서 입력 받은 도큐먼트는 this로 표현
      - ◆ Map 함수에서 다음 단계로 도큐먼트를 넘겨주기 위해서는 emit이라는 특별한 함수를 써야 하는데 첫 번째 파라미터로는 그룹핑의 기준을 넣어야 하고 두 번째 파라미터로 다음 단계에 넘겨줄 값을 설정

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● Reduce 함수

      var reducer = function(key, values){ return values.length}

      - ◆ Reduce 함수는 두 개의 파라미터를 가지는데 첫 번째 파라미터는 그룹핑 된 기준값을 두 번째 파라미터는 배열에 각각의 값이 담긴 형태
      - ◆ Reduce 함수에서 연산을 처리하고 다음 단계로 값을 넘길 때는 return을 이용

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● out

```
out: {
        <action>: <collectionName>,
        db: <dbName>,
        sharded: <boolean>,
        nonAtomic: <boolean>
}
```

      - ◆ action
        - ▪ replace: 저장할 컬렉션 이름
        - ▪ merge: 저장할 컬렉션 이름 – 동일한 _id가 있으면 덮어씌움
        - ▪ reduce: 저장할 컬렉션 이름 – 동일한 _id가 있으면 더해서 적용
      - ◆ db: 저장할 데이터베이스 이름
      - ◆ shared: true로 설정하면 컬렉션을 새딩하면서 저장
      - ◆ nonAtomic: action 값이 merge 나 reduce 일 때만 사용되는 옵션으로 Map-Reduce 의 결과로 저장될 때 해당 데이터베이스는 잠금 상태가 되는데 true로 설정하면 잠기지 않음

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 맵 리듀스 수행

```
> db.rating.mapReduce(mapper, reducer, {out: {inline: 1}})
```

```
{
  results: [
    { _id: 4, value: 2 },
    { _id: 3, value: 2 },
    { _id: 2, value: 1 },
    { _id: 1, value: 1 },
    { _id: 5, value: 4 }
  ],
  ok: 1
}
```

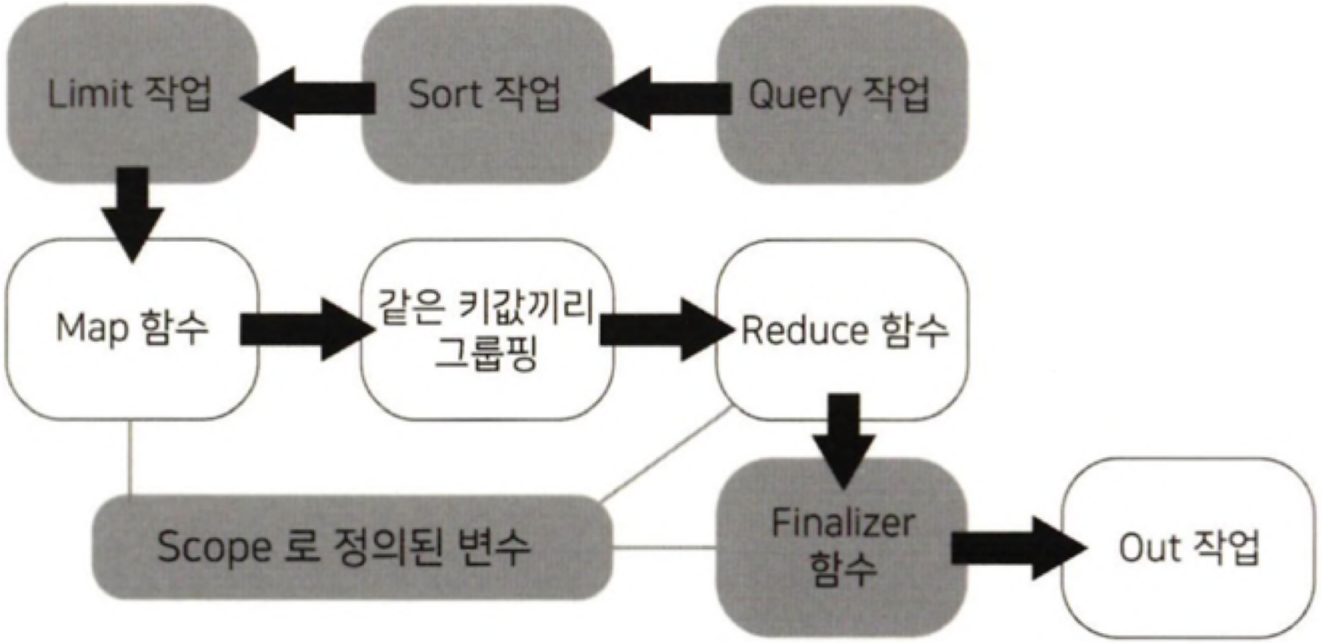
CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 맵 리듀스 수행

```
> db.users.mapReduce(mapper, reducer, {out:{inline:1}});

{ results: [ { _id: null, value: 10 } ], ok: 1 }
```

CRUD

- ❖ 데이터 집계
  - ✓ Map Reduce
    - ● 선택 단계



CRUD

❖ 데이터 집계

✓ 집계 파이프라인

- count 함수: 데이터 개수

```
> db.products.countDocuments()

10
```

- distinct 함수: 중복 제거

```
> db.products.distinct('manufacture')

[ 'Amazon', 'Apple', 'Google', 'Samsung', 'Sony' ]
```

CRUD

- ❖ 데이터 집계
  - ✓ 집계 파이프라인
    - ● aggregate(): 집계 함수
      - ◆ $group : 그룹화
        - ▪ $sum, max, min, avg, first, last, push, addToSet, stdDevPop, stdDevSamp 연산자
        - ▪ products Collection에서 manufacture 별로 그룹화해서 price 합계 추출하기

```
> db.products.aggregate([{$group:{_id:{"maker":"$manufacture"},
    sum_prices:{$sum :"$price"}}}])
```

```
{ _id: { maker: 'Samsung' }, sum_prices: 1014.98 }
{ _id: { maker: 'Sony' }, sum_prices: 499 }
{ _id: { maker: 'Google' }, sum_prices: 199 }
{ _id: { maker: 'Apple' }, sum_prices: 2296 }
{ _id: { maker: 'Amazon' }, sum_prices: 328 }
```

CRUD

- ❖ 데이터 집계
  - ✓ 집계 파이프라인
    - ● aggregate(): 집계 함수
      - ◆ $group : 그룹화
        - ▪ $sum, max, min, avg, first, last, push, addToSet, stdDevPop, stdDevSamp 연산자
        - ▪ products Collection에서 category 별로 그룹화해서 price 합계 추출하기

```
> db.products.aggregate([{$group:{_id:{"category":"$category"},
    sum_prices:{$sum:"$price"}}}])
```

```
{ _id: { category: 'Cell Phones' }, sum_prices: 563.99 }
{ _id: { category: 'Tablets' }, sum_prices: 2774.99 }
{ _id: { category: 'Laptops' }, sum_prices: 998 }
```

CRUD

- ❖ 데이터 집계
  - ✓ 집계 파이프라인
    - ● aggregate(): 집계 함수
      - ◆ $push : 세부 그룹화

```
> db.products.aggregate([{$group:{_id:{"maker":"$manufacture"},
    categories:{$push:"$category"}}}])
```

```
{ "_id" : { "maker" : "Google" }, "categories" : [ "Tablets" ] }
{ "_id" : { "maker" : "Sony" }, "categories" : [ "Laptops" ] }
{ "_id" : { "maker" : "Amazon" }, "categories" : [ "Tablets", "Tablets" ] }
{ "_id" : { "maker" : "Apple" }, "categories" : [ "Tablets", "Tablets",
    "Tablets", "Laptops" ] }
{ "_id" : { "maker" : "Samsung" }, "categories" : [ "Cell Phones",
    "Tablets" ] }
```

CRUD

- ❖ 데이터 집계
  - ✓ 집계 파이프라인
    - ● aggregate(): 집계 함수
      - ◆ $sort : 정렬

```
> db.products.aggregate([{$sort:{price:-1}}])
```

```
{ "_id" : ObjectId("5ec09e673c8c5329a914c845"), "name" : "iPad 64GB
    Wifi", "category" : "Tablets", "manufacture" : "Apple", "price" : 699 }
{ "_id" : ObjectId("5ec09e673c8c5329a914c844"), "name" : "iPad 32GB
    Wifi", "category" : "Tablets", "manufacture" : "Apple", "price" : 599 }
{ "_id" : ObjectId("5ec09e673c8c5329a914c846"), "name" : "Galaxy S3",
    "category" : "Cell Phones", "manufacture" : "Samsung", "price" :
    563.99 }…
```

CRUD

- ❖ 데이터 수정
  - ✓ 갱신을 위한 함수
    - db.collection.update()
    - db.collection.updateOne()
    - db.collection.updateMany()
    - db.collection.replaceOne()

CRUD

- 데이터 수정
  - ✓ replaceOne

```
db.collection.replaceOne(
<query>,
<replacement>,
{
    upsert: <boolean>,
    writeConcern: <document>,
    collation: <document>
})
```

- query: 업데이트 할 도큐먼트의 기준으로 find()에서 사용하는 쿼리와 동일
- replacement: 도큐먼트를 대체할 내용
- upsert: 기본값은 false로 이 값이 true 이면 도큐먼트가 없을 경우 새로운 도큐먼트를 추가
- writeConcern: WriteConcern을 사용하려면 이 파라미터를 사용
- collation: 언어에 대한 설정을 할 수 있는데 대소문자 관계에 대한 순서를 다룸

CRUD

- 데이터 수정
  - ✓ replaceOne

```
> db.user.replaceOne({username: "karoid"},
{
  username: "Karpoid" , status: "Sleep" , points: 100, password: 2222
});

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

CRUD

❖ 데이터 수정

✓ replaceOne

```
> db.user.find()

{ _id: ObjectId("62799dc7c174fa3271682bda"),
  username: 'Karpoid',
  status: 'Sleep',
  points: 100,
  password: 2222 }
{ _id: ObjectId("6279a1b1c174fa3271682bdb"),
  username: 'John',
  password: 4321 }
{ _id: ObjectId("6279a1b1c174fa3271682bdc"),
  username: 'K',
  password: 4221 }
{ _id: ObjectId("6279a1b1c174fa3271682bdd"),
  username: 'Mark',
  password: 5321 }
```

CRUD

- 데이터 수정
  - ✓ replaceOne

```
> db.myCollection.replaceOne({ item: "abcl23" },{item:"abc123", status:"P",
    points:100}, {upsert:true});

{ acknowledged: true,
  insertedId: ObjectId("627ae443d894745707476afe"),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1 }
```

CRUD

❖ 데이터 수정

✓ update

● 매개변수는 조건, 갱신내용, 옵션 3가지가 있으며 1번째 매개변수는 조건이고 2번째 매개변수는 변경 내용, 3번째 매개변수는 upsert 옵션으로 생략 가능

```
> db.sample.drop()

true

> db.sample.insert({name:"park", score:90})

{ acknowledged: true,
  insertedIds: { '0': ObjectId("627ae4895fc4fd5668ae45ba") } }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  score: 90 }
```

CRUD

❖ 데이터 수정

✓ update

● 옵션없이 수정을 하면 데이터 전체를 변경하기 때문에 score 속성만 남게 됨

```
> db.sample.update( { name: 'park' }, { $set: { score: 100 } } )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  score: 90 }
```

CRUD

- ❖ 데이터 수정
  - ✓ update
    - ● 도큐먼트 수정 연산자
      - ◆ $currentDate:<field〉필드의 값으로 현재 timestamp나 date 값을 갖도록 추가
      - ◆ $inc: <field> 필드의 값을 증가
      - ◆ $min: <field> 필드의 값을 최소값으로 설정
      - ◆ $max: <field> 필드의 값을 최대값으로 설정
      - ◆ $mul: <field> 필드의 값을 곱 한 값으로 수정
      - ◆ $rename: <field> 필드의 이름을 변경
      - ◆ $setOnInsert: Upsert 값이 true 이면서 문서를 생성할 때만 사용되는 연산자로 쿼리와 $set 연산자에 나온 필드와 값 이외의 필드와 값을 함께 설정
      - ◆ $unset: <field> 필드를 제거하는데 $unset: {(field): ...} 로 연산자 안에서 값은 아무 의미가 없음

CRUD

- ❖ 데이터 수정
  - ✓ update
    - ● $inc: 필드의 값을 주어진 만큼 증가시켜서 저장

```
> db.sample.update({name:"park"},{$inc:{score:-5}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  score: 95 }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
```

CRUD

❖ 데이터 수정

✓ update

● $mul: 필드의 값을 주어진 만큼의 배수로 저장

```
> db.sample.update({name:"park"},{$mul:{score:0.5}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  score: 47.5 }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
```

CRUD

- ❖ 데이터 수정
  - ✓ update
    - ● $rename: 필드의 이름을 변경해서 저장

```
> db.sample.update({name:"park"},{$rename:{"score":"점수"}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  '점수': 47.5 }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
```

CRUD

- ❖ 데이터 수정
  - ✓ update
    - ● $unset: 필드를 삭제

```
> db.sample.update({name:"park"},{$unset:{점수:0.5}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"), name: 'park' }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
```

CRUD

❖ 데이터 수정

✓ update

- $setOnInsert: $set과 비슷한데 upsert의 경우에만 작동하는데 만약 upsert가 일어나지 않으면 아무 동작도 하지 않음
- $currentDate: 필드의 값을 현재 시간으로 변경해서 저장

```
> db.sample.update({name:"park"},{$currentDate:{lastModified:true}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  lastModified: 2022-05-10T22:34:16.280Z }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
```

CRUD

❖ 데이터 수정

✓ update

- multi: 조건에 맞는 데이터 모두 변경
- upsert 옵션: true로 설정하면 일치하는 조건의 데이터가 없으면 삽입

```
> db.sample.update({_id:1}, {$set:{score:90}, $setOnInsert:{age:30}},
    {upsert:true})

{ acknowledged: true,
  insertedId: 1,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  lastModified: 2022-05-10T22:34:16.280Z }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
{ _id: 1, age: 30, score: 90 }
```

CRUD

❖ 데이터 수정

✓ update

● upsert 옵션: true로 설정하면 일치하는 조건의 데이터가 없으면 삽입

```
> db.users.update({name:"ryu"}, {$set:{score:90}}, {upsert:false})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0 }

> db.users.update({name:"ryu"}, {$set:{score:90}}, {upsert:true})

{ acknowledged: true,
  insertedId: ObjectId("627ae8fdd894745707476d06"),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1 }
```

CRUD

- 데이터 수정
  - ✓ 배열 수정
    - $addToSet: 배열 안에 해당 값이 없다면 추가하고 있다면 추가하지 않음
    - $pop: 배열의 첫 번째 혹은 마지막 요소를 삭제
    - $pull: 쿼리에 해당하는 요소 하나를 제거
    - $push: 해당 요소를 배열에 추가
    - $pullAll: 해당하는 값을 가지는 요소 전부를 제거

CRUD

❖ 데이터 수정

✓ 배열 수정

```
> db.board.insert( {"board_num":1, "board_id":"ggangpae1", "board_title":"가입인사",
    "board_content":"안녕하세요 반갑습니다.", "reply":[]})

{ acknowledged: true,
  insertedIds: { '0': ObjectId("627aea175fc4fd5668ae45bc") } }

> db.board.update({"board_num":1},{"$push":{"reply":{"reply_num":1,
    "reply_content":"반갑습니다", "reply_id":"root", "reply_time":ISODate("2019-12-
    31T09:00:00")}}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

CRUD

❖ 데이터 수정

✓ 배열 수정

```
> db.board.update({"board_num":1},{"$push":{"reply":{"reply_num":2,
    "reply_content":"정말로 반가워", "reply_id":"system", "reply_time":ISODate("2019-
    12-31T09:00:00")}}})
```

```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

CRUD

❖ 데이터 수정

✓ 배열 수정

```
> db.board.find()

{ _id: ObjectId("627aea175fc4fd5668ae45bc"),
  board_num: 1,
  board_id: 'ggangpae1',
  board_title: '가입인사',
  board_content: '안녕하세요 반갑습니다.',
  reply:
   [ { reply_num: 1,
       reply_content: '반갑습니다',
       reply_id: 'root',
       reply_time: 2019-12-31T09:00:00.000Z },
     { reply_num: 2,
       reply_content: '정말로 반가워',
       reply_id: 'system',
       reply_time: 2019-12-31T09:00:00.000Z } ] }
```

CRUD

❖ 데이터 수정

✓ 배열 수정

```
> db.board.update({"board_num":1,
    "reply.reply_num":1},{"$set":{"reply.$":{"reply_num":1,
    "reply_time":ISODate("2020-12-31T10:00:00")}}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

CRUD

❖ 데이터 수정

✓ 배열 수정

```
> db.board.find()

{ _id: ObjectId("627aea175fc4fd5668ae45bc"),
  board_num: 1,
  board_id: 'ggangpae1',
  board_title: '가입인사',
  board_content: '안녕하세요 반갑습니다.',
  reply:
   [ { reply_num: 1, reply_time: 2020-12-31T10:00:00.000Z },
     { reply_num: 2,
       reply_content: '정말로 반가워',
       reply_id: 'system',
       reply_time: 2019-12-31T09:00:00.000Z } ] }
```

CRUD

❖ 데이터 수정

✓ 배열 수정

```
> db.board.update({"board_num":1},{"$pull":{"reply":{"reply_num":1}}})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }

> db.board.find()

{ _id: ObjectId("627aea175fc4fd5668ae45bc"),
  board_num: 1,
  board_id: 'ggangpae1',
  board_title: '가입인사',
  board_content: '안녕하세요 반갑습니다.',
  reply:
   [ { reply_num: 2,
       reply_content: '정말로 반가워',
       reply_id: 'system',
       reply_time: 2019-12-31T09:00:00.000Z } ] }
```

CRUD

❖ 데이터 수정

✓ updateOne

```
db.collection.updateOne(
    <query>,
    <update>,
    {
        upsert: <boolean>,
        writeConcern: <document>,
        collation: <document>,
        arrayFilters: [ <filterdocument1>, ... ]
    }
)
```

CRUD

- 데이터 수정
  - ✓ updateMany

```
db.collection.updateMany(
    <query>,
    <update>,
    {
        upsert: <boolean>,
        writeConcern: <document>,
        collation: <document>,
        arrayFilters: [ <filterdocument1>, ... ]
    }
)
```

CRUD

❖ 데이터 수정

✓ updateMany

{

"acknowledged" : <Boolean>,

"matchedCount" : <integer>,

"modifiedCount" : <integer>

}

- query: 업데이트할 도큐먼트의 조건
- update: 변경 사항
- upsert: true로 설정하면 쿼리한 도큐먼트가 없을 경우 새로운 도큐먼트를 추가
- writeConcern
- collation
- arrayFilters: 배열 필드에서 업데이트 작업을 위해 수정할 배열 요소를 결정하는 필터 문서 배열

CRUD

❖ 데이터 수정

✓ updateMany

```
> db.containerBox.find(null, {_id:false})

{ name: 'bear', weight: 60, category: 'animal' }
{ name: 'bear', weight: 10, category: 'animal' }
{ name: 'cat', weight: 2, category: 'animal' }
{ name: 'phone', weight: 1, category: 'electronic' }

> db.containerBox.updateMany({ name: "bear"},
{$set: {name: "teddy bear", category: "toy"}});

{ acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0 }
```

CRUD

❖ 데이터 수정

✓ updateMany

```
> db.containerBox.find(null, {_id:false})

{ name: 'teddy bear', weight: 60, category: 'toy' }
{ name: 'teddy bear', weight: 10, category: 'toy' }
{ name: 'cat', weight: 2, category: 'animal' }
{ name: 'phone', weight: 1, category: 'electronic' }
```

CRUD

❖ 데이터 수정

✓ updateMany

```
> db.character.insertMany([
{name: 'x', inventory: ['pen', 'cloth', 'pen']},
{name: 'y', inventory: ['book', 'cloth'], position:{x:1, y:5}},
{name: 'z', inventory: ['wood', 'pen'], position:{x:0, y:9}}])

{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("627af0d75fc4fd5668ae45bd"),
     '1': ObjectId("627af0d75fc4fd5668ae45be"),
     '2': ObjectId("627af0d75fc4fd5668ae45bf") } }
```

CRUD

❖ 데이터 수정

✓ updateMany

```
> db.character.updateMany({}, {$set: {"inventory.$[penElem]": "pencil"}},
    {arrayFilters: [{penElem: 'pen'}]})

{ acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 2,
  upsertedCount: 0 }
```

CRUD

❖ 데이터 수정

✓ updateMany

```
> db.character.find({"inventory": "pencil"})

{ _id: ObjectId("627af0d75fc4fd5668ae45bd"),
  name: 'x',
  inventory: [ 'pencil', 'cloth', 'pencil' ] }
{ _id: ObjectId("627af0d75fc4fd5668ae45bf"),
  name: 'z',
  inventory: [ 'wood', 'pencil' ],
  position: { x: 0, y: 9 } }
```

CRUD

- 데이터 삭제
  - ✓ deleteOne 과 deleteMany

```
db.collection.deleteOne(
    <query>,
    {
        writeConcern: <document>,
        collation: <document>
    }
)

db.collection.deleteMany(
    <query>,
    {
        writeConcern: <document>,
        collation: <document>
    }
)
```

CRUD

❖ 데이터 삭제

✓ deleteOne 과 deleteMany

{

“acknowledged”: <boolean>,

“deletedCount” : <integer>

}

- query: 업데이트할 도큐먼트의 조건
- writeConcern
- collation

CRUD

❖ 데이터 삭제

✓ deleteOne 과 deleteMany

```
> db.character.deleteMany({})

{ acknowledged: true, deletedCount: 3 }

> db.containerBox.deleteMany({category: 'animal'})

{ acknowledged: true, deletedCount: 1 }
```

CRUD

❖ 데이터 삭제

✓ remove

```
> db.sample.remove({_id:1}, {justOne:true})

{ acknowledged: true, deletedCount: 1 }

> db.sample.find()

{ _id: ObjectId("627ae4895fc4fd5668ae45ba"),
  name: 'park',
  lastModified: 2022-05-10T22:34:16.280Z }
{ _id: ObjectId("627ae58b5fc4fd5668ae45bb"),
  name: 'park',
  score: 90 }
```

CRUD

- 데이터 삭제
  - ✓ 수정과 삭제가 될 때 완료되기 전에는 다른 곳에서 데이터를 조회할 수 없도록 하기 위해서는 {{삭제 수정 조건}, $isolated:1} 옵션을 추가
  - ✓ Collection 삭제는 db.Collection이름.drop()

CRUD

- Bulk
  - ✓ 대량의 작업을 모아서 처리하는 것
  - ✓ bulkWrite: 명령을 모아서 한번에 수행할 수 있는 함수
    - ordered 옵션에 true 와 false 를 이용해서 병렬로 처리할 지 순차적으로 처리를 설정한 수 있음
    - 사용 가능한 작업: insertOne, updateOne, updateMany, deleteOne, deleteMany

CRUD

- Bulk
  - ✓ bulkWrite

```
> db.board.bulkWrite([{insertOne:{"document":{"title":"hi", "content":"hello"}}},
    {updateOne:{"filter":{"title":"hi"}, "update":{$set:{"content":"안녕하세요"}}}}])

{ acknowledged: true,
  insertedCount: 1,
  insertedIds: { '0': ObjectId("627affcf5fc4fd5668ae45c0") },
  matchedCount: 1,
  modifiedCount: 1,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {} }
```

CRUD

- Bulk
  - ✓ bulkWrite

```
> db.board.find()

{ _id: ObjectId("627aea175fc4fd5668ae45bc"),
  board_num: 1,
  board_id: 'ggangpae1',
  board_title: '가입인사',
  board_content: '안녕하세요 반갑습니다.',
  reply:
   [ { reply_num: 2,
       reply_content: '정말로 반가워',
       reply_id: 'system',
       reply_time: 2019-12-31T09:00:00.000Z } ] }
{ _id: ObjectId("627affcf5fc4fd5668ae45c0"),
  title: 'hi',
  content: '안녕하세요' }
```

CRUD

- Bulk
  - ✓ bulkWrite
    - 다중 쓰기 동작을 수행하는 API
    - Bulk 동작 빌더 초기화

      var bulk = db.inventory.initializeUnorderedBulkOp();
    - 동작 추가

      bulk.동작(매개변수);
    - 동작 실행

      bulk.execute()

CRUD

- Bulk
  - ✓ Bulk

```
> var bulk = db.inventory.initializeUnorderedBulkOp();

> bulk.insert({item:"iPadPro", details:{RAM:"3GB", SSD:"128GB"}, category:"Apple"});

> bulk.insert({item:"MacBookPro", details:{RAM:"16GB",SSD:"256GB"},
    category:"Apple"});

> bulk.execute();

{ acknowledged: true,
  insertedCount: 2,
  insertedIds:
   [ { index: 0, _id: ObjectId("627b00b35fc4fd5668ae45c1") },
     { index: 1, _id: ObjectId("627b00b35fc4fd5668ae45c2") } ],
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: [] }
```

CRUD

- ❖ Bulk
  - ✓ Bulk

```
> db.inventory.find()

{ _id: ObjectId("627998e6c174fa3271682bca"),
  item: 'ABC2',
  details: { model: '14Q3', manufacturer: 'ABC Company' },
  stock: [ { size: 'M', qty: 50 } ],
  category: 'clothing' }
{ _id: ObjectId("627998e6c174fa3271682bcb"),
  item: 'ABC3',
  details: { model: '14Q3', manufacturer: 'CCC Company' },
  stock:
   [ { size: 's', qty: 25 },
     { size: 'M', qty: 50 },
     { size: 'L', qty: 50 } ],
  category: 'clothing' }
```

CRUD

- Bulk
  - ✓ Bulk

```
> db.inventory.find()

{ _id: ObjectId("627998e6c174fa3271682bcc"),
  item: 'ABC3',
  details: { model: '14Q3', manufacturer: 'AAA Company' },
  stock: [ { size: 's', qty: 30 } ],
  category: 'clothing' }
{ _id: ObjectId("627b00b35fc4fd5668ae45c1"),
  item: 'iPadPro',
  details: { RAM: '3GB', SSD: '128GB' },
  category: 'Apple' }
{ _id: ObjectId("627b00b35fc4fd5668ae45c2"),
  item: 'MacBookPro',
  details: { RAM: '16GB', SSD: '256GB' },
  category: 'Apple' }
```

CRUD

- Transaction 처리
  - ✓ Mongo DB에서는 Session을 가져와서 트랜잭션을 설정

```
session = db.getMongo().startSession()
session.startTransaction({
    readConcern: { level: "snapshot" },
    writeConcern: { w: "majority" }
});

<원하는 작업을 수행>

session.commitTransaction();
session.endSession();
```

  - ✓ 트랜잭션 취소

```
session.abortTransaction();
```