MongoDB 기본 명령어 익히기

기본적인 명령어 익히기

서버 시작

```
mongod <option list> -f config.conf
```

서버 종료

```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
```

서버 상태

```
> db.serverStatus()
{
        "host" : "tlog-mongo01",
        "version" : "4.4.3",
        "process" : "mongod",
        "pid" : NumberLong(7034),
        "uptime" : 192,
        "uptimeMillis" : NumberLong(191855),
        "uptimeEstimate" : NumberLong(191),
        "localTime" : ISODate("2021-02-02T06:06:07.393Z"),
        "asserts" : {
                "regular" : 0,
                "warning" : 0,
                "msg" : 0,
                "user" : 25,
                "rollovers" : 0
        },
        "connections" : {
                "current" : 1,
                "available" : 818,
                "totalCreated" : 1,
                "active" : 1,
                "exhaustIsMaster" : 0,
                "exhaustHello" : 0,
                "awaitingTopologyChanges" : 0
        },
... ...
```

데이터베이스, 컬렉션, 도큐먼트에 대한 개념 익히기

mongodb 는 데이터 레코드를 bson 형태의 도큐먼트로 저장하면 이 도큐먼트 들은 컬렉션에 함께 보장된다.
데이터베이스는 이러한 컬렉션들의 개념적인 컨테이너 역할을 한다.

데이터베이스

데이터베이스 생성

최소 하나의 도큐먼트가 생성되야 상위 컬렉션과 함께 데이터베이스도 생성된다.

```
> use myDB
switched to db myDB
> use myDB
switched to db myDB
> db.newCollection.insertOne( {x:1} )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6018efa69d338b940f4f5183")
}
```

데이터베이스 조회

```
> use myDB
switched to db myDB
```

데이터베이스 삭제

```
> use myDB
switched to db myDB
> db.dropDatabase()
{ "dropped" : "myDB", "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

데이터베이스 상태조회

```
> db.stats()
```

컬렉션 명령어

컬렉션 생성

```
> db.newCollection.insertOne( {x:1} )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6018efa69d338b940f4f5183")
}
```

컬렉션 이름변경

```
> db.newCollection.renameCollection("renamedCollection");
{ "ok" : 1 }
```

컬렉션 조회

```
> show collections
renamedCollection
>
```

컬렉션 삭제

```
> db.renamedCollection.drop()
true
> show collections
```

컬렉션 상태 조회

```
> db.newCollection.stats()
{
        "ns" : "myDB.newCollection",
        "size" : 33,
        "count" : 1,
        "avgObjSize" : 33,
        "storageSize" : 4096,
        "freeStorageSize" : 0,
        "capped" : false,
        "wiredTiger" : {
                "metadata" : {
                        "formatVersion" : 1
```

```
                },
... ...
```

capped 컬렉션

- 데이터 입력과 조회 동작의 높은 throughput 목적으로한 컬렉션
- 순서를 보장하기 때문에 데이터 반환시 이를 위한 인덱스가 불필요한다.
- capped 사이즈 만큼 고정되면 오래된 데이터는 fifo 방식으로 자동 삭제 된다.
- 휘발성 로그 statistic 데이터 등을 저장하는데 유용할 수 있다.
- 임의로 삭제하거나 갱신과 같은 연산을 할 수 없다.
- oplog.rs 등

```

```

도큐먼트 생성

https://docs.mongodb.com/manual/mongo/

도큐먼트 쿼리 와 SQL 비교

MongoDB bson 쿼리	**SQL**
db.collection.insert()	INSERT
db.collection.bulkWrite()	Batched DML(INSERT,UPDATE, DELETE)
db.collection.update() \n db.collection.update({}, {$set: {}}, {upsert:true})	UPDATE \n REPLACE \n (INSERT.. ON DUPLICATE KEY UPDATE...)
db.collection.remove()	DELETE
db.collection.find()	SELECT
db.collection.aggregte() \n MapReduce	SELECT .. GROUP BY ..

단일 도큐먼트 생성, 다수 도큐먼트 생성

db.collection.insertOne()

단일 도큐먼트를 생성한다. 결과에 따라 true false 를 반환한다.

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

db.collection.insertMany()

다수의 도큐먼트를 생성한다. 결과에 따라 true false 를 반환한다.

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

db.collection.insert()

단일 혹은 다수의 도큐먼트를 생성한다.

```
db.collection.insert(
   <document or array of documents>,
   {
     writeConcern: <document>,
     ordered: <boolean>
   }
)
```

도큐먼트 조회, 점 연산자, 프로젝션

db.collection.find()

도큐먼트를 검색하고 검색된 도큐먼트의 커서를 반환한다.

```
db.collection.find( query, projection )
db.collection.find()
db.collection.find( { _id: 5 } )
db.collection.find(
        { _id: { $in: [ 5, ObjectId("507c35dd8fada716c89d0013") ] } }
)
db.collection.find(
        { birth: { $gt: new Date('1950-01-01') } }
)
db.collection.find(
        birth: { $gt: new Date('1920-01-01') },
        death: { $exists: false }
)
```

프로젝션

find() 메소드에서 query 는 SQL 의 condition 에 해당한다. 즉 조건에 맞는 도큐먼트의 모든 filed 를 반환하는데 프로젝션을 이용하여 각 도큐먼트에서 필드를 선택하거나 연산을 할 수 있다. SQL 의 SELECT 절에 해당한다.

점 연산자

점 연산자를 통해 embedded 도큐먼트내 필드에 접근할 수 있다.

```
> db.users.insert({name: {first: "john", last: "kennedy"}})
WriteResult({ "nInserted" : 1 })

> db.users.find( { "name.first" : "john" , "name.last" : "kennedy" } )
{ "_id" : ObjectId("601a1e98fb91390a854cf027"), "name" : { "first" : "john",
"last" : "kennedy" } }
```

커서와 커서를 이용한 도큐먼트 반환

RDBMS 에서 제공하는 api 에서 SELECT 는 resultset 을 반환하는데 이를 제어하는 것을 커서라고 한다.
mongodb 에서고 커서를 제공하면 find() 와 같은 메소드에서 query 의 결과가 있는 경우 커서를 반환한다.

```
> var cursor = db.users.find()
> cursor.sort( {"name" : -1} ).allowDiskUse()
{ "_id" : ObjectId("601a3c1afb91390a854cf029"), "name" : { "first" : "john3",
"last" : "kennedy3" } }
{ "_id" : ObjectId("601a3c13fb91390a854cf028"), "name" : { "first" : "john2",
"last" : "kennedy2" } }
{ "_id" : ObjectId("601a1e98fb91390a854cf027"), "name" : { "first" : "john",
"last" : "kennedy" } }
> cursor.isClosed()
true
>
> var cursor = db.users.find()
> cursor.size()
3
> cursor.skip(1)
{ "_id" : ObjectId("601a3c13fb91390a854cf028"), "name" : { "first" : "john2",
"last" : "kennedy2" } }
{ "_id" : ObjectId("601a3c1afb91390a854cf029"), "name" : { "first" : "john3",
"last" : "kennedy3" } }
>
> var cursor = db.users.find()
> cursor.forEach( function(myDoc) { print( "user: " + myDoc.name.first ); } );
user: john
user: john2
user: john3
```

Update

샘플 데이터

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },
   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P"
},
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D"
},
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A"
},
   { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A"
},
   { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status:
"A" }
] );
```

db.collection.updateOne()

```
조건에 맞는 최초 한개의 도큐먼트만 업데이트를 수행한다.
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

db.collection.updateMany()

조건에 맞는 모든 도큐먼트를 업데이트를 수행한다.

```
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

db.collection.replaceOne()

필드 일부를 변경하지 않고 해당 도큐먼트를 삭제하고 새로운 도큐먼트로 대체한다.

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty:
40 } ] }
)
```

Delete

샘플 데이터

```
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D"
},
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A"
},
] );
```

db.collection.deleteMany()

컬렉션에서 조건에 맞는 다수혹은 전체 도큐먼트를 삭제한다.

```
db.inventory.deleteMany({ status : "A" })
db.inventory.deleteMany({ })
```

db.collection.deleteOne()

컬렉션에서 조건에 맞는 한개의 도큐먼트를 삭제한다.

```
db.inventory.deleteOne( { status: "D" } )
```