2. MongoDB CRUD

차례

1. Collection과 Database 생성, 조회, 수정, 삭제

- Database, Collection

```
>use testDB  # Database 생성
>show dbs  # Database 목록보기

>db.createCollection('emp') # Collection 생성
>db.myCollection.insert({x:1, y:10}) # Document 입력으로 Collection 생성
>show collections  #collection 목록보기
>show dbs   # database 목록 보기

>db.dropDatabase()  # 현재 사용중인 database 삭제, 이 작업 중에는 글로벌 쓰기 Lock이 걸림
>db.emp.renameCollection("바꿀이름") #collection 이름 바꾸기
>db.employee.drop()  #collection 삭제
```

1. Collection과 Database 생성, 조회, 수정, 삭제

- Capped Collection

db.createCollection(<컬렉션 이름>,{capped:true, size: <제한할 크기>});

```
>db.createCollection("emp",{capped:true, size:2100000000});
# capped: 해당공간이 모두 사용되면 자동으로 가장 오래된 데이터를 삭제
# size : Collection 제한 사이즈
```

1. Collection과 Database 생성, 조회, 수정, 삭제

- mongoDB 관리 툴
  - MongoDB Ops : 기업용 서버에 사용 가능
  - 무료 클라우드 모니터링

>db.enableFreeMonitoring()

```
> db.enableFreeMonitoring()
{
        "state" : "enabled",
        "message" : "To see your monitoring data, navigate to the unique URL below. Anyone you share the URL with will also be
 able to view this page. You can disable monitoring at any time by running db.disableFreeMonitoring().",
        "url" : "https://cloud.mongodb.com/freemonitoring/cluster/NA5DELNSBRBULHKSX7PDMWM4KZHOCCI6",
        "userReminder" : "",
        "ok" : 1
}
```

- 명령이 수행되는 시간
- 메모리 사용률
- CPU 사용를
- 수행된 명령 수

1. Collection과 Database 생성, 조회, 수정, 삭제

- 데이터베이스 상태 정보 조회

```
>db.getCollectionInfos()
#현재 데이터베이스 collection 정보를 리스트로 반환
>db.serverStatus()
# 호스트, 프로세스 Id, Lock 옵션, 스토리지 엔진 이름, 스토리지 엔진 통계 정보 제공
>db.stats()
# 데이터베이스 내 collection, view, object의 개수와 크기에 대한 통계를 제공
```

1. Collection과 Database 생성, 조회, 수정, 삭제

- Collection 상태 정보 조회

```
>db.collection.isCapped()  #캡드 컬렉션이면 true 반환
>db.collection.latencyStats()  # 컬렉션의 지연 시간 통계를 보여줌
>db.collection.stats()   # 컬렉션의 크기, 도큐먼트 개수, 스토리지 엔지 통계제공
>db.collection.storageSize() # 컬렉션 스토리지 크기 반환
>db.collection.totalIndexSize() # 컬렉션의 인덱스 크기 반환
>db.collection.totalSzie()  # 컬렉션의 스토리지 인덱스 크기의 합 반환
```

2. Document 생성

- Single Document 생성
  - Document 구조

```
{
    field1:value1;
    field2:value2;
    .
    .
    fieldN:valueN;
}
```

```
{
    name: "Kim",
    age: 23,
    place: "Seoul",
    hobbies: ["Singing", "Reading Books"]
}
```

```
{  // 중첩 Document 예시
    name: "Kim",
    age: 23,
    place: "Seoul",
    hobbies: ["Singing", "Reading Books"]
    spouse: {
        name: "Lee",
        age: 21
    }
}
```

- 단일 Document 생성

db.collection.insertOne()

```
> db.user.insertOne({userID: "kimikimi", username: "Kim", password: 1111})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f50a274237701f054a0e52e")
}
> db.user.find().pretty()
{
        "_id" : ObjectId("5f50a274237701f054a0e52e"),
        "userID" : "kimikimi",
        "username" : "Kim",
        "password" : 1111
}
```

2. Document 생성

- 단일 Document 생성 – 실습

```
> use test
> db.emp.insert({ eno : 1101, fname : 'JIMMY'});
> db.emp.insert({ eno : 1102, fname : 'ADAM', lname : 'KROLL'});
> db.emp.insert({ eno : 1103, fname : 'SMITH', job : 'CLERK'});
> db.emp.find()
> db.emp.find().sort({eno:1})

> m={empno: 1014, ename:'smith'}
> db.things.save(m)
> db.things.find()
> db.things.insertOne({empno:1102,ename:'king'})
> for(var n=1103;n<=1120;n++) db.things.save({empno:n, ename:'test'})
> it
# 출력된 결과가 20 개를 초과하면 다음 화면으로 넘어감
```

2. Document 생성

- 단일 Document 생성 :_id 지정하여 입력 시 문제점

```
> db.testCollection.insertOne({_id:1, x:1})
{ "acknowledged" : true, "insertedId" : 1 }
> db.testCollection.insertOne({_id:1, x:2})
WriteError({
        "index" : 0,
        "code" : 11000,
        "errmsg" : "E11000 duplicate key error collection: test.testCollection index: _id_ dup key: { _id: 1.0 }",
        "op" : {
                "_id" : 1,
                "x" : 2
        }
}) :
WriteError({
        "index" : 0,
        "code" : 11000,
        "errmsg" : "E11000 duplicate key error collection: test.testCollection index: _id_ dup key: { _id: 1.0 }",
        "op" : {
                "_id" : 1,
                "x" : 2
        }
})
WriteError@src/mongo/shell/bulk_api.js:465:48
mergeBatchResults@src/mongo/shell/bulk_api.js:871:49
executeBatch@src/mongo/shell/bulk_api.js:940:13
Bulk/this.execute@src/mongo/shell/bulk_api.js:1182:21
DBCollection.prototype.insertOne@src/mongo/shell/crud_api.js:264:9
@(shell):1:1
>
```

2. Document 생성

- 다수 도큐먼트 생성

db.컬렉션이름.insertMany()

```
> db.user.insertMany( [
{username: "Kei", password: 4321 },
{username: "Mijoo", password: 3212 },
{username: "Yein", password: 3123 },
]);
{
   "acknowledged" : true,
   "insertedIds" : [
      ObjectId("5f51894e237701f054a0e52f"),
      ObjectId("5f51894e237701f054a0e530"),
      ObjectId("5f51894e237701f054a0e531")
   ]
}
```

```
> db.user.find().pretty()
{
"_id" : ObjectId("5f50a274237701f054a0e52e"),
"userID" : "kimikimi",
"username" : "Kim",
"password" : 1111
}
{
"_id" : ObjectId("5f51894e237701f054a0e52f"),
"username" : "Kei",
"password" : 4321
}
{
"_id" : ObjectId("5f51894e237701f054a0e530"),
"username" : "Mijoo",
"password" : 3212
}
{
"_id" : ObjectId("5f51894e237701f054a0e531"),
"username" : "Yein",
"password" : 3123
}
```

2. Document 생성

- 다수 도큐먼트 생성 : _id 중첩

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

2. Document 생성– 실습

1. board 데이터베이스 생성
2. 자유 게시판과 비밀 게시판 생성
3. 자유게시판에 아무 글 3개 작성, 글 하나에 댓글하나가 달린 상태로 생성해보자
4. 비밀게시판에 작성자가 'noname'값을 가지는 글하나 작성해보다.

```
use board
freeboard_result=db.board.insertOne({name:'자유게시판'})
freeboard_id=freeboard_result.insertedId

db.article.insertMany([
{board_id : freeboard_id, title:'hello', content : 'Hi, hello', author:'Karoid'},
{board_id : freeboard_id, title:'hello', content : 'Hi, hello', author:'Karoid'},
{board_id : freeboard_id, title:'hello', content : 'Hi, hello', author:'Karoid',comments:[
{author :'Kim', content:'hello Hong'}]},
])

secretboard_result=db.board.insertOne({name:'비밀게시판'})
secretboard_id=freeboard_result.insertedId

db.article.insertOne({board_id : secretboard_id, title: 'my Secret title', content : 'Hi, hello', author:'noname'})
```

3. 도큐먼트 조회

- Find()

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

Parameter	Type	Describtion
query	document	Optional. 쿼리 연산자를 이용해 원하는 도큐먼트를 선택.
projection	document	Optional. return할 필드를 선택할 수 있다.
documents	cursor	데이터베이스 상의 도큐먼트의 커서

```
>show collections
> db.user.find()
> db.user.find().pretty()
```

3. 도큐먼트 조회

- 쿼리 (Query)
  - 데이터베이스 속의 수 많은 정보 속에서 원하는 정보를 찾아내도록 도와주는 필터같은 역할(where 절)

```
> db.user.find({username: "Yein"})
{ "_id" : ObjectId("5f51894e237701f054a0e531"), "username" : "Yein", "password" : 3123 }
```

  - MongoDB에서는 원하는 데이터를 찾기 위해 연산자를 사용
  - 연산자의 종류: 비교(Comparison), 논리(Logical), 요소(Element), 배열(Array) 등

3. 도큐먼트 조회

- 쿼리 (Query)
  - **비교(Comparison)**

Operator	Describtion
$eq	(equals) 주어진 값과 일치하는 값
$gt	(greater than) 주어진 값보다 큰 값
$gte	(greather than or equals) 주어진 값보다 크거나 같은 값
$lt	(less than) 주어진 값보다 작은 값
$lte	(less than or equals) 주어진 값보다 작거나 같은 값
$ne	(not equal) 주어진 값과 일치하지 않는 값
$in	주어진 배열 안에 속하는 값
$nin	주어빈 배열 안에 속하지 않는 값

3. 도큐먼트 조회

- 쿼리 (Query)
  - **논리(Logical)**

Operator	Describtion
$or	주어진 조건중 하나라도 true 일 때 true
$and	주어진 모든 조건이 true 일 때 true
$not	주어진 조건이 false 일 때 true
$nor	주어진 모든 조건이 false 일때 true

  - **$where 연산자**
    - $where 연산자를 통하여 javascript expression 을 사용 할 수 있음.

3. 도큐먼트 조회

- 쿼리
  - **$regex 연산자**

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

Option	Describtion
i	대소문자 무시
m	정규식에서 anchor(^) 를 사용 할 때 값에 \n 이 있다면 무력화
x	정규식 안에있는 whitespace를 모두 무시
s	dot (.) 사용 할 떄 \n 을 포함해서 매치

3. 도큐먼트 조회

- 쿼리 : 실습

```
use testDB
db.containerBox.insertMany([
  {name: "bear", weight: 60, category: "animal"},
  {name: "bear", weight: 30, category: "animal"},
  {name: “bear", weight: 10, category: "animal"},
  {name: “cat", weight: 20, category: "animal"},
  {name: "cat", weight: 2, category: "animal"},
  {name: "phone", weight: 1, category: "electronic"}
])
```

1. Weight가 10 이상인 것을 검색하라.
2. Category가 animal 인 것을 검색하라.
3. Category가 animal 이고 name bear인 것을 검색하라.
4. Weight가 20~30 인 것을 검색하라.
5. Name이 cat, phone 인 것을 검색하라.

3. 도큐먼트 조회

- 점 연산자
  - 오브젝트 내의 값을 불러오기 위해 사용

```
var myVar={hello:'world'}
myVar.hello

var a={name : {firstName: 'Karodi', lastName:'jeong'}}
a.name.firstName

db.A.save(a)

db.A.find({‘name.firstName’:'Karodi'})
```

3. 도큐먼트 조회

- 프로젝션 (Projection)
  - 쿼리의 결과값에서 보여질 field를 선택

```
db.user.find( { } ,{ "_id": false, "username": true, "password": true} )
db.user.find( null, {"_id":false, "username":true} )
```

3. 도큐먼트 조회

- 커서 (cursor)
  - 쿼리 결과에 대한 포인터
  - find 명령은 결과로 도큐먼트를 직접 반환하지 않고 커서를 반환
  - 해당 도큐먼트의 위치 정보만을 반환하여 작업을 효율적으로 만들 수 있음.
  - 데이터를 전부 불러올 도큐먼트만 선별적으로 조회 가능
  - 커서는 결과값을 읽는 작업을 위해 필요한 것이기 때문에 10분이 지나면 비활성 상태로 전환.

```
> var cursor = db.user.find()
> cursor
> cursor.hasNext()
> cursor.next()
> It
> var arr=db.user.find().toArray()
> Arr[5]
```

```
> db.cappedCollection.find().forEach(function(document){
작업
...})
```

4. 도큐먼트 수정

- 도큐먼트 교체

db.collection.replaceOne(filter, replacement, options)

```
db.collection.replaceOne(
<filter>,
<replacement>,
{
upsert: <boolean>,  #insert 역할
writeConcern: <document>,
collation: <document>,
hint: <document|string> // Available starting in 4.2.1
}
)
```

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

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

4. 도큐먼트 수정

- 도큐먼트 수정

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

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

4. 도큐먼트 수정

- 업데이터 연산자 : 필드

이름	설명
$currentDate	필드 값을 날짜 또는 타임스탬프 형식으로, 현재 날짜로 설정한다.
$inc	필드 값을 지정된 양만큼 증가시킨다.
$min	지정된 값이 기존 필드 값보다 작은 경우에만 필드를 업데이트한다.
$max	지정된 값이 기존 필드 값보다 큰 경우에만 필드를 업데이트한다.
$mul	필드 값을 지정된 양만큼 곱한다.
$rename	필드명을 수정한다.
$set	도큐먼트의 필드 값을 설정한다.
$setOnInsert	업데이트로 도큐먼트가 삽입되는 경우 필드 값을 설정한다. 기존 도큐먼트를 수정하는 경우에는 영향을 미치지 않는다.
$unset	도큐먼트에서 지정된 필드를 제거한다.

4. 도큐먼트 수정

- 업데이트 연산자 : 배열 연산자

이름	설명
$	쿼리 조건과 일치하는 첫 번째 요소를 업데이트하라는 자리 표시자 역할을 한다.
$[]	쿼리 조건과 일치하는 도큐먼트에 대해 배열의 모든 요소를 업데이트하라는 자리 표시자 역할을 한다.
$[<identifier>]	쿼리 조건과 일치하는 도큐먼트에 대해 arrayFilters 조건과 일치하는 모든 요소를 업데이트하는 자리 표시자 역할을 한다.
$addToSet	Set에 이미 존재하지 않는 경우에만 배열에 요소를 추가한다.
$pop	배열의 첫 번째 또는 마지막 항목을 제거한다.
$pull	지정된 쿼리와 일치하는 모든 배열의 요소를 제거한다.
$push	배열에 항목을 추가한다.
$pullAll	배열에서 일치하는 모든 값을 제거한다.

4. 도큐먼트 수정

- 업데이트 연산자: 배열 –한정어

이름	설명
$each	$push 및 $addToSet 연산자가 배열 업데이트를 위해 여러 항목을 추가할 수 있도록 한정한다.
$position	$push 연산자가 배열에 요소를 추가하기 위해 배열 안의 위치를 지정할 수 있도록 한정한다.
$slice	$push 연산자가 업데이트된 배열의 크기를 제한하도록 한정한다.
$sort	$push 연산자가 배열에 저장된 도큐먼트의 순서를 변경하도록 한정한다.

4. 도큐먼트 수정

- 수정 배열 실습

```
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}}
])
```

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

```
db.character.update({inventory:'pen'}
{
$set:{"inventory.$":"pencil"}
})
```

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

```
db.character.updateMany(
{inventory:'pen'},
 { $set:{"inventory.$":"pencil"}
})
```

5. 도큐먼트 삭제

- db.collection.deleteOne({<query>)
- db.collection.deleteMany({<query>)

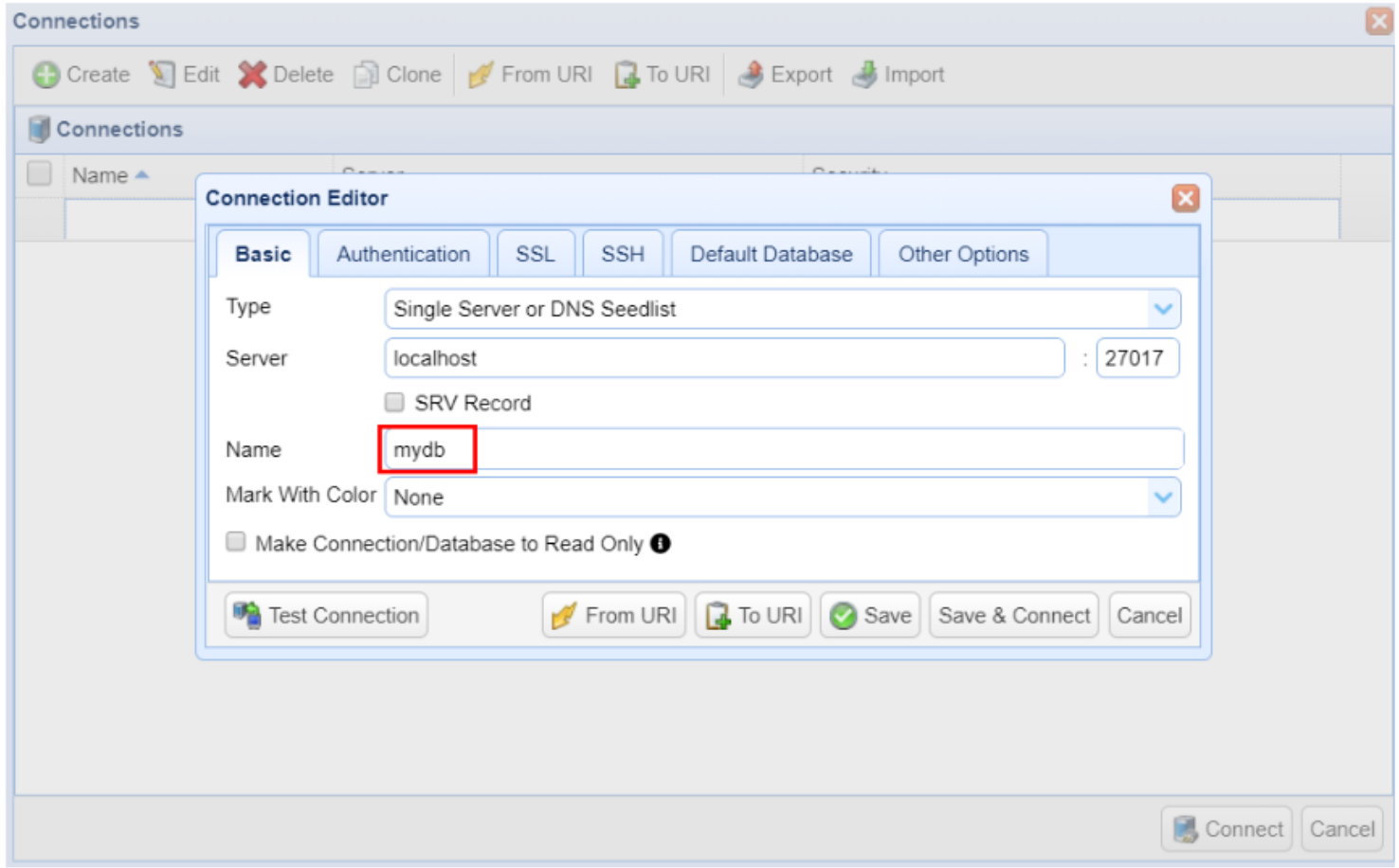
```
db.character.deleteOne({name:'x'})
db.character.deleteMany({name:'x'})
db.character.deleteMany({})
db.character.drop()
db.dropDatabase()
```

6. NoSQLBooster

- NoSQLBooster 설치
  - 다운로드 주소 : https://www.nosqlbooster.com/downloads

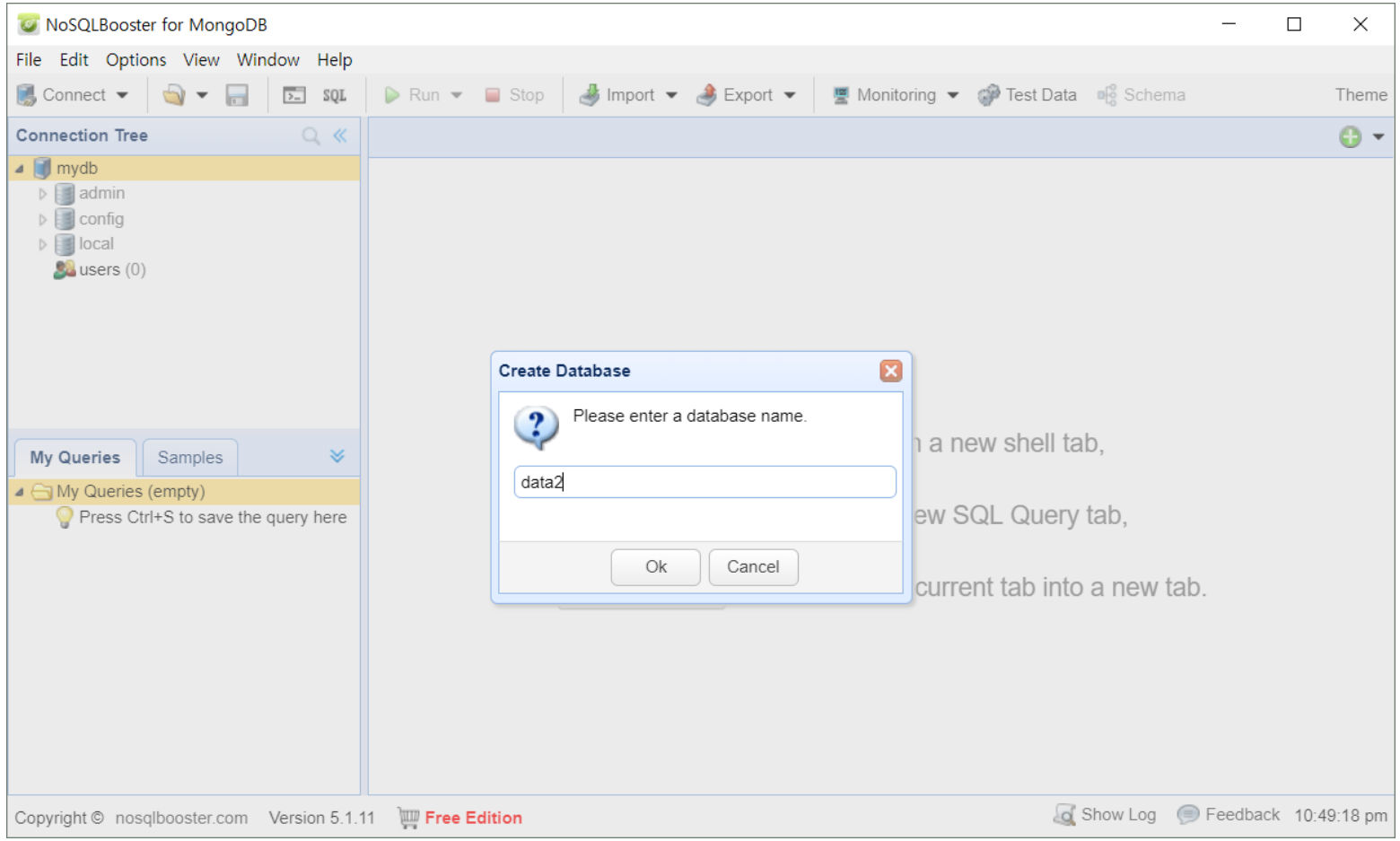
6. NoSQLBooster

- NoSQLBooster DB연결
  - connection -> create 메뉴 선택



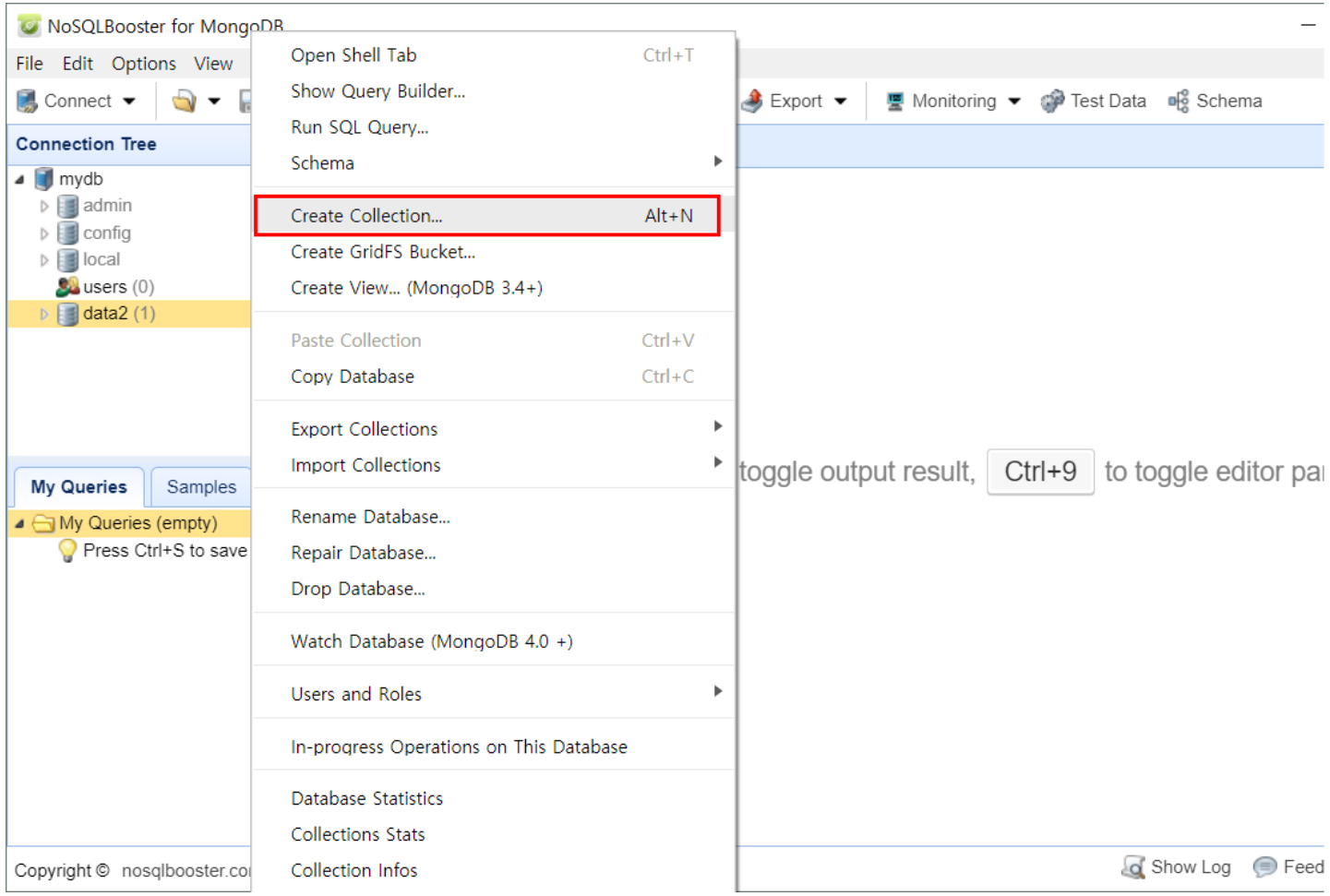
6. NoSQLBooster

- 데이터베이스 생성
  - 새로 생성된 connect에서 우클릭 -> create Database 선택



6. NoSQLBooster

- collection 생성
  - db에서 우클릭 create Collection 선택



6. NoSQLBooster

- Document CRUD
  - collectio에서 더블클릭선택