

# 게임서버프로그래밍

2019년 2학기

한국산업기술대학교  
게임공학부

정내훈

## 8장 NoSQL 기초

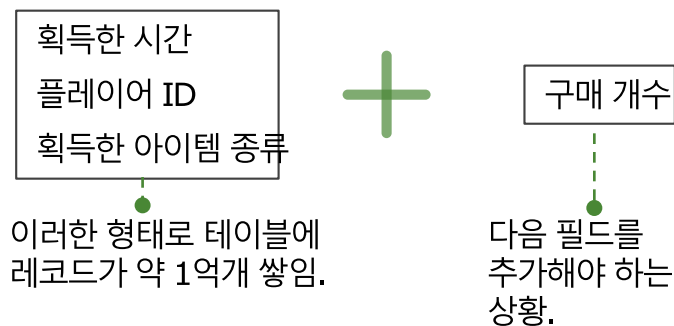
---

- 1 | 관계형 데이터베이스와 NoSQL
- 2 | 관계형 데이터베이스에서 확장성
- 3 | 관계형 데이터베이스에서 고가용성
- 4 | MongoDB를 위한 JSON 이해
- 5 | MongoDB 시작
- 6 | MongoDB에 데이터 액세스
- 7 | 성능 분석 기능
- 8 | MongoDB 수평 확장
- 9 | 게임 서버에서 MongoDB 명령 실행
- 10 | 요약 및 더 알아보기

## 8.1 | 관계형 데이터베이스와 NoSQL

### • 관계형 데이터베이스의 부족한 점

플레이어가 아이템을 구매한 로그를 담는 테이블을 다음과 같이 정의했다고 가정.



기존에 쌓여 있던 레코드 1억 개에도 '구매 개수'라는 필드를 반드시 추가해야 함.

(한 테이블의 모든 레코드는 필드 구성이 모두 같아야 하기 때문)

기존 레코드에는 새로 추가된 필드 값이 모두 null이 되게 함으로써 문제를 해결할 수 있다.

기존 테이블에 레코드 1억 개가 이미 들어 있다고 가정.

필드 하나를 추가하면 데이터베이스 엔진은 기존에 있는 레코드 1억 개 전체에 필드를 추가해야 한다.

프로그램 구조가 복잡해질수록 테이블 구조도 변경, 유지보수 하면서 점점 힘들어짐.

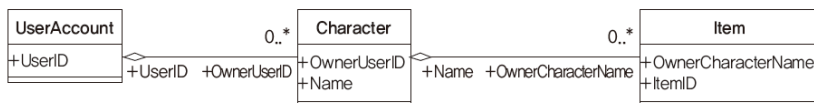


그림 8-1  
사용자, 플레이어 캐릭터, 아이템의  
테이블 관계

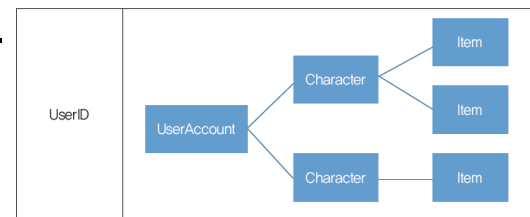


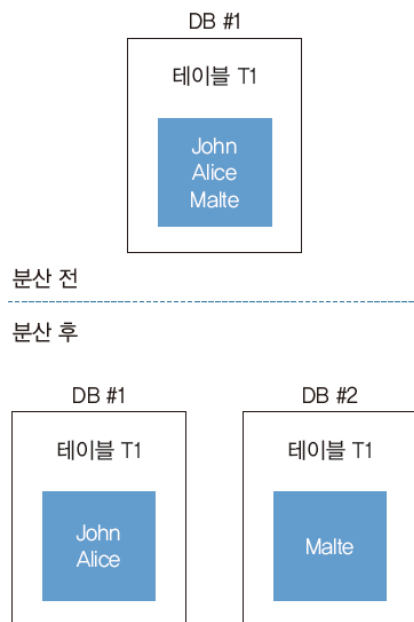
그림 8-2  
레코드의 필드 하나에 JSON 도큐먼트를  
넣는 과격한 방법

## 8.2 | 관계형 데이터베이스에서 확장성

### • 데이터베이스의 수직분산과 수평 분산

데이터베이스의 수직(vertical) 분산 : 여러 테이블을 각각 여러 데이터베이스 컴퓨터에 나눈다.

데이터베이스의 수평(horizontal) 분산 : 테이블 하나가 레코드를 1억 개 가졌다고 가정하고 데이터베이스 컴퓨터가 100대 주어졌다면, 각 컴퓨터에 데이터베이스를 설치하고 1억을 100으로 나눈 수인 100만 개씩 레코드를 분배한다. 이렇게 수평으로 분산된 데이터베이스에서는 각 컴퓨터가 큰 테이블 1개를 조각조각 가진 셈으로, 이를 샤드(shard, 조각)라고 한다.



### • 이 상태에서 데이터베이스를 액세스하려면

John의 레코드를 얻고자 할 때는 먼저 John이 어느 샤드에 있는지 파악해야 한다.

- 해시 함수 사용: John이라는 문자열을 입력 값으로 하여 해시 함수를 실행한다. 연산 후 얻은 정수 값을 샤드 넘버로 사용한다.
- 로케이터(locator) DB 사용: John이 어느 샤드에 저장되었는지를 담고 있는 별도의 테이블에 액세스하여 John이 어느 샤드에 있는지 파악한다.

그림 8-3  
테이블을 샤드 2개로 분산

## 8.2 | 관계형 데이터베이스에서 확장성

- 해당 샤드에 데이터베이스 질의를 실행하기

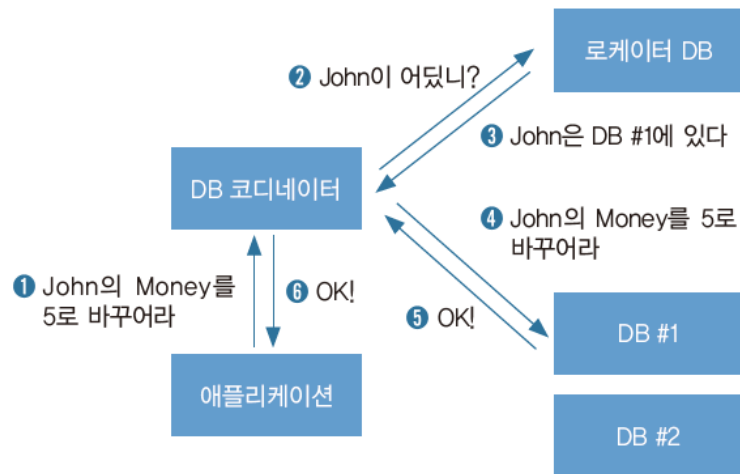


그림 8-4 DB 코디네이터가 로케이터 DB, DB 샤드를 이용해서 DB 분산

- 1 앱(게임 서버)이 데이터베이스에 질의를 던진다.
- 2 데이터베이스 질의를 수행하는 코디네이터(coordinator)는 질의에 관련된 레코드가 있는 위치를 어떤 데이터베이스에 묻는다.
- 3 해당 샤드 위치를 파악한다.
- 4 해당 샤드에서 레코드 액세스를 처리한다.
- 5, 6 결과를 받는다.

## 8.2 | 관계형 데이터베이스에서 확장성

### • 분산 락(distributed lock)

기기 둘 이상에 걸쳐 저장된 데이터를 액세스하는 동안 다른 스레드나 프로세스에서 액세스를 블로킹시켜 주는 것

기기 두 대 이상에 걸쳐 발견되어야 하는 데이터는 내부적으로 분산 락으로 보호되면서 액세스 되고, 이렇게 함으로써 일관성을 지킬 수 있다.

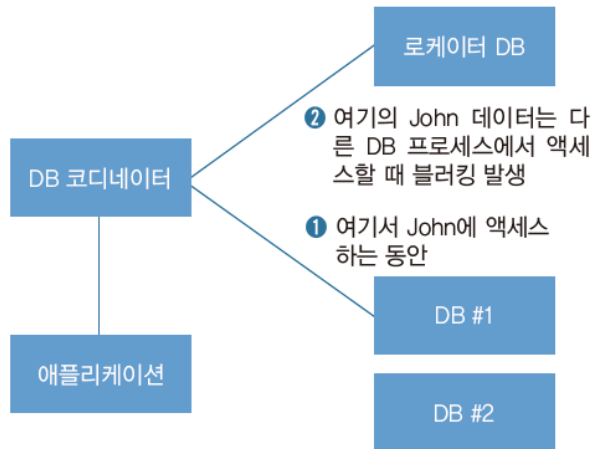


그림 8-5  
일관성을 중요시 할 때의 처리 과정

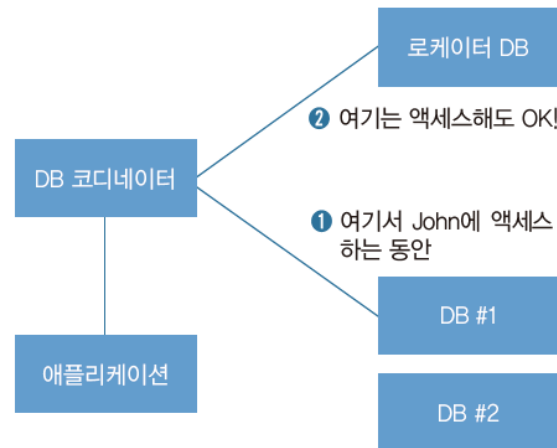


그림 8-6  
일관성을 중요시하지 않을 때의 처리 과정

- 1 샤드에 든 레코드를 액세스하는 동안 관련된 다른 기기에서 락의 강도를 낮추면
- 2 락이 그만큼 줄기 때문에 수평 분산의 효과를 제대로 볼 수 있다.

액세스하는 데이터가 항상 같은 결과를 입출력하지 않는 결과가 발생할 수 있다.

“더 이상 관계형 데이터베이스로서 역할을 하지 못한다”

작업 : 1등 플레이어 찾기

시나리오 : 만일 어떤 서버에서 1등과 2등을 차례로 다른 DB#1과 DB#2에 추가한 경우 다른 서버가 DB#1을 먼저 보고 DB#2를 보았다면? => 2등이 1등!

## 8.3 | 관계형 데이터베이스에서 고가용성

- 장애 극복(fail over)-레코드의 상시 백업

이중화 혹은 다중화(redundancy : 같은 데이터가 DB 1과 DB 2에 모두 있게 하는 것

DB 1을 액세스하다가 DB 1이 죽으면 DB 2가 DB 1 대신 액세스를 받아 처리한다.

이때 DB 1을 액티브(active) 또는 마스터(master)라고 하며, DB 2를 패시브(passive) 또는 슬레이브(slave)라고 한다.

이렇게 구성된 데이터베이스는 한쪽이 죽더라도 나머지가 대체 역할을 하기 때문에 항상 이용 가능한 상태가 된다.  
즉, 고가용성을 만족시킴.

- 데이터 일관성

데이터를 변경할 때 액티브와 패시브를 모두 변경한 후에야 '처리 완료'를 선언하고, 그때까지 질의 실행 결과는 블로킹 된다.

게임 서버가 질의 구문을 수행할 때는 ❶, ❷뿐만 아니라 ❸, ❹이 끝날 때까지 대기.

→ 가용성을 높이려고 성능을 희생시킨 것.

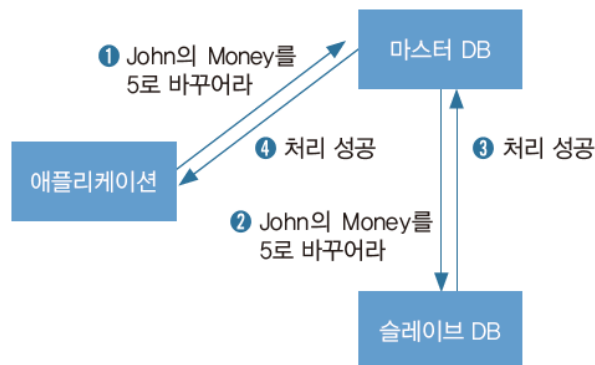
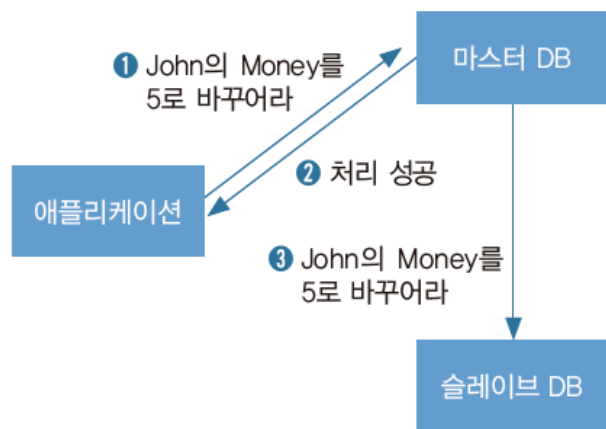


그림 8-7  
일관성을 중요시할 때의 데이터 변경 처리

## 8.3 | 관계형 데이터베이스에서 고가용성

- 고가용성을 유지하면서 데이터베이스 처리 속도를 유지하려면

슬레이브가 없을 때처럼 속도가 잘 나오게 하려면, 슬레이브의 변경 전파 과정을 건너뛰고 바로 게임서버에 응답한다.



- 1 게임 서버가 마스터 DB에 변경하는 질의를 보낸다.
- 2 마스터 DB는 변경 요청을 바로 처리하고 결과를 반환한다.
- 3 마스터 DB는 슬레이브 DB에 변경된 것을 알려 준다.  
슬레이브 DB는 변경 사항을 자기 자신에게 적용한다.

그림 8-8 신속한 응답을 중요시할 때의 데이터 변경 처리

최신이 아닌 데이터인 스테일 데이터(stale data)를 가져오는 문제가 있다.

관계형 데이터베이스는 ACID(원자성, 일관성, 고립성, 지속성)를 추구하는 것이 원칙이다.

ACID 중에서 C(일관성)와 I(고립성)를 어느 정도 포기하면 '만족스러운 수준'의 스케일 아웃(기기를 많이 두어 문제를 해결할 수 있음)과 고가용성을 확보할 수 있다.



## 8.3 | 관계형 데이터베이스에서 고가용성

- BASE( Basically Available), '기본적으로 가용성'

그리고 그들의 데이터베이스는 소프트 스테이트( `soft state`)로 일시적으로 데이터 상태가 변화 중일 수 있다, "데이터 상태가 유연하다."라는 의미. 그리고 그들의 일관성을 결과적 일관성( `eventual consistency`)이라고 한다.

- NoSQL

1. ACID 대신 BASE를 지향한다.
2. 일관성을 일부 희생하더라도 높은 확장성과 고가용성을 실현한다.
3. 탈(脫) 테이블 구조의 유연한 저장 방식을 이용하여 더 효율적인 프로그래밍을 추구한다.

## 8.4 | MongoDB를 위한 JSON 이해


- JSON(JavaScript Object Notif ication)

그리고 그들의 데이터베이스는 소프트 스테이트( **soft state**)로 일시적으로 데이터 상태가 변화 중일 수 있다, “데이터 상태가 유연하다.”라는 의미. 그리고 그들의 일관성을 결과적 일관성( **eventual consistency**)이라고 한다.

`{"a" : 1}` 

a	1
---	---

그림 8-11 a = 1 내용을 가진 JSON 객체

`{  
 "a" : 1,  
 "b" : "x",  
 "c" : 100  
}` 

a	1
b	x
c	100

그림 8-12 JSON 객체 도식화

값에는 문자열, 숫자 말고도 다음과 같은 종류의 다른 객체를 넣을 수도 있다.

- a = 1
- b = "x"
- c = 100
- d = <또 다른 객체>

`{  
 "a" : 1,  
 "b" : "x",  
 "c" : 100,  
 "d" : {"x" : 50, "y" : "qp"}  
}`



a	1	
b	x	
c	100	
d	x	50
	y	qp

그림 8-13  
JSON 값에는 또 다른 객체가  
들어갈 수 있음

## 8.4 | MongoDB를 위한 JSON 이해

객체가 아닌 배열을 값으로 가질 수도 있다. 이때 배열은 대괄호 [] 구문 블록으로 둘러싼다.

```
{
  "a" : 1,
  "b" : "x",
  "c" : 100,
  "d" : { "x" : 50, "y" : "qp" },
  "e" : [99, 88, 77]
}
```



a	1						
b	x						
c	100						
d	<table> <tr> <td>x</td><td>50</td></tr> <tr> <td>y</td><td>qp</td></tr> </table>	x	50	y	qp		
x	50						
y	qp						
e	<table> <tr> <td>0</td><td>99</td></tr> <tr> <td>1</td><td>88</td></tr> <tr> <td>2</td><td>77</td></tr> </table>	0	99	1	88	2	77
0	99						
1	88						
2	77						

그림 8-14

객체가 아닌 배열을 값으로 가질 수도 있음

배열 안에 들어가는 값은 문자열이나 숫자로, 중괄호 {} 구문 블록을 사용해서 객체를 넣어도 된다.

```
{
  "a" : 1,
  "b" : "x",
  "c" : 100,
  "d" : { "x" : 50, "y" : "qp" },
  "e" : [99, 88, 77, { "pi" : 3.14, "r" : 1 }]
}
```



a	1												
b	x												
c	100												
d	<table> <tr> <td>x</td><td>50</td></tr> <tr> <td>y</td><td>qp</td></tr> </table>	x	50	y	qp								
x	50												
y	qp												
e	<table> <tr> <td>0</td><td>99</td></tr> <tr> <td>1</td><td>88</td></tr> <tr> <td>2</td><td>77</td></tr> <tr> <td>3</td><td> <table> <tr> <td>pi</td><td>3.14</td></tr> <tr> <td>r</td><td>1</td></tr> </table> </td></tr> </table>	0	99	1	88	2	77	3	<table> <tr> <td>pi</td><td>3.14</td></tr> <tr> <td>r</td><td>1</td></tr> </table>	pi	3.14	r	1
0	99												
1	88												
2	77												
3	<table> <tr> <td>pi</td><td>3.14</td></tr> <tr> <td>r</td><td>1</td></tr> </table>	pi	3.14	r	1								
pi	3.14												
r	1												

그림 8-15

배열 안에 객체를 가질 수도 있음

## 8.5 | MongoDB 시작

- MongoDB에서 다루는 데이터의 집합 관계와 관계형 데이터베이스(RDBMS)의 집합관계의 차이

RDBMS에서 DB 인스턴스라고 부르는 것은 MongoDB에서도 똑같이 부른다.

RDBMS의 DB 인스턴스 안에는 테이블이 있다. MongoDB의 DB 인스턴스 안에는 컬렉션(collection)이 있다.

RDBMS의 테이블 안에는 레코드가 있다. MongoDB에서는 컬렉션 안에 레코드 대신 문서(document)가 들어간다.

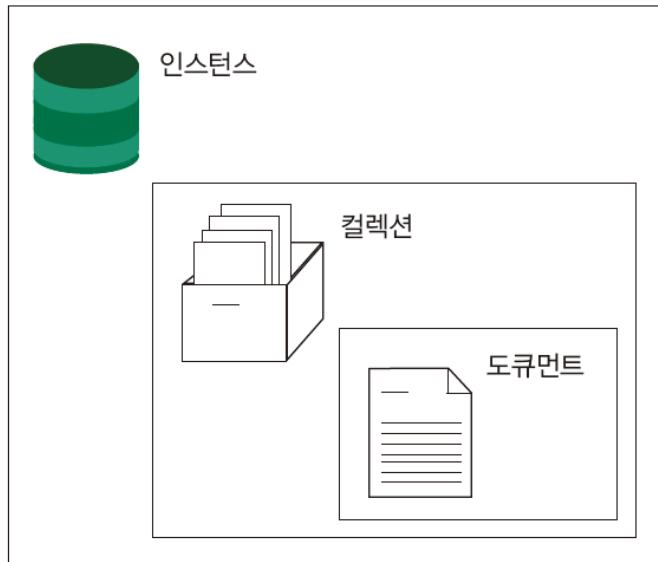


그림 8-16 테이블=컬렉션, 레코드=도큐먼트

The screenshot shows the Robo 3T interface. The top bar indicates the current database is 'db-01' and the collection is 'coll1'. The command bar shows the query: `db.getCollection('coll1').find({})`. The results table displays 5 documents with their keys and values.

Key	Value	Type
(1) ObjectId("574083739ff656498d3a7e50")	{ 2 fields }	Object
_id	ObjectId("574083739ff656498d3a7e50")	ObjectId
x	1.0	Double
(2) ObjectId("5741a0f6fdcabb5a8c1489a")	{ 3 fields }	Object
_id	ObjectId("5741a0f6fdcabb5a8c1489a")	ObjectId
x	1	Int32
y	2	Int32
(3) ObjectId("5741a10afdcabb5a8c1489b")	{ 4 fields }	Object
_id	ObjectId("5741a10afdcabb5a8c1489b")	ObjectId
x	1	Int32
y	2	Int32
z	John	String
(4) ObjectId("5741a118fdcabb5a8c1489c")	{ 5 fields }	Object
_id	ObjectId("5741a118fdcabb5a8c1489c")	ObjectId
x	1	Int32
y	2	Int32
z	John	String
w	[ 3 elements ]	Array
(5) ObjectId("5741a12ffdcabb5a8c1489d")	{ 6 fields }	Object
_id	ObjectId("5741a12ffdcabb5a8c1489d")	ObjectId
x	1	Int32
y	2	Int32
z	John	String
w	[ 3 elements ]	Array
[0]	1	Int32
[1]	2	Int32
[2]	3	Int32
q	{ 2 fields }	Object
f1	3.14	Double
f2	5.99	Double

그림 8-17 Robo 3T(MongoDB의 GUI 도구)

## 8.5 | MongoDB 시작

- MongoDB 설치하기

1. <https://www.mongodb.com>에서 운영체제에 맞는 파일을 내려받아 설치.

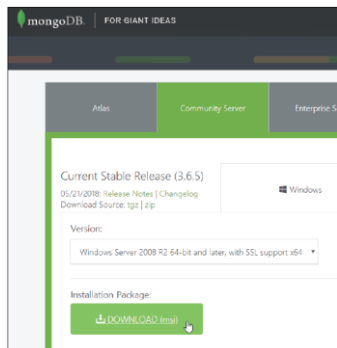


그림 8-18  
MongoDB 웹 사이트에서  
내려받기

2. 설치한 후에는 mongod.exe를 실행한다.  
단 그냥 실행하면 안 되고 dbpath를 지정해야 한다. dbpath에 폴더를 하나 만들어 거기에 MongoDB가 저장할 데이터베이스 파일을 생성하면 된다.

`mongod --dbpath d:\data\mongodb\data`

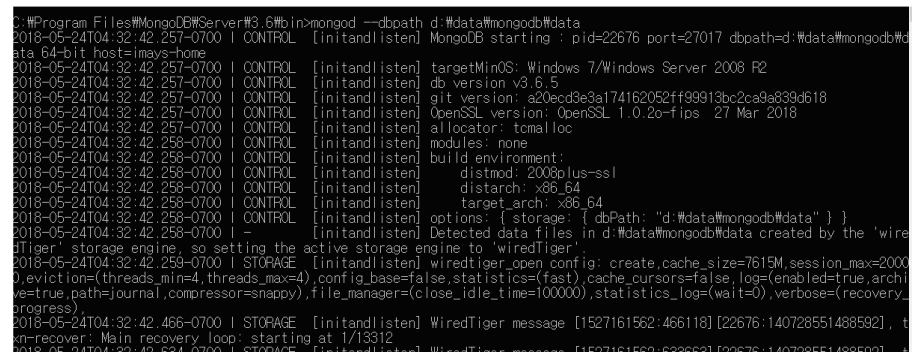


그림 8-19 dbpath를 지정한 후 mongod.exe 실행 예

3. <https://robomongo.org>에 접속해서 Download Robo 3T 버튼을 클릭하여 설치를 진행.

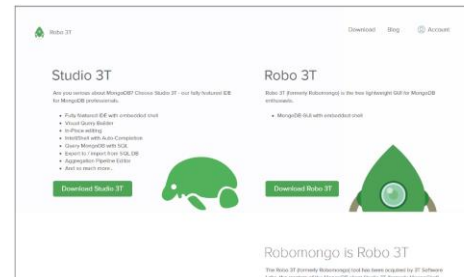


그림 8-20  
Robo 3T 내려받기

## 8.5 | MongoDB 시작

4. 설치를 마쳤으면 Robo 3T 첫 화면에서 MongoDB 서버에 접속한 후 Open Shell을 실행하여 명령어를 입력할 준비를 한다.

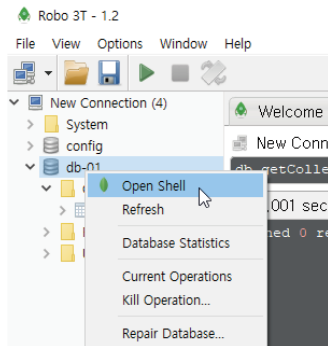


그림 8-21  
Open Shell 실행

5. DB 인스턴스 생성하기 및 들어가기

use <DBNAME>을 입력한다. 예를 들어 use test01을 입력하면 test01 이름의

DB 안에서 여러 행동을 할 수 있다

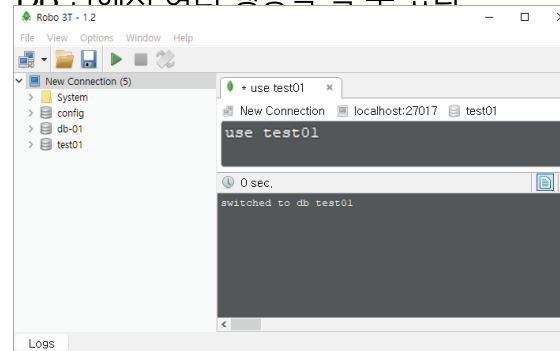


그림 8-22  
지금부터 test01 이름의 인스턴스 안에서 활동

## 8.6 | MongoDB에 데이터 액세스

### • 8.6.1 생성(create)

`db.<COLLNAME>.insert(<OBJECT>)`

컬렉션과 도큐먼트를 만들기.

<COLLNAME>에는 컬렉션 이름을 입력하고,  
<OBJECT>에는 JSON 형식으로 도큐먼트를 입력  
한다.

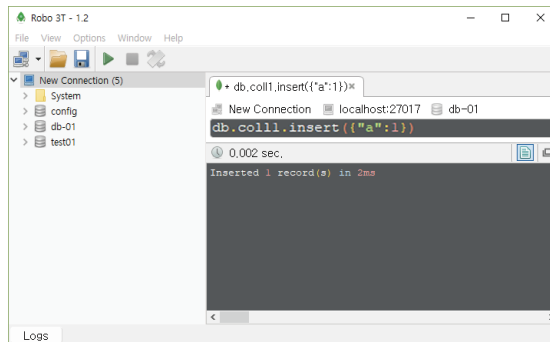


그림 8-23 도큐먼트를 컬렉션에 추가

## 8.6 | MongoDB에 데이터 액세스

### 8.6.2 읽기(read)

`db.<COLLNAME>.find(<COND>)`

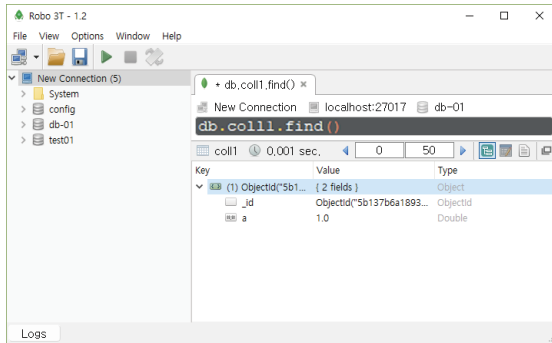


그림 8-24 컬렉션 안에 있는 문서 나열

원하는 문서 찾기

<COND>에는 찾기 조건문을 넣는다.  
 {} 혹은 아무것도 안 넣으면 "모두를 찾아라."라는 의미.  
 <COLLNAME>은 찾고자 하는 컬렉션 이름.  
 find 대신 findOne을 사용하면 첫 번째로 나오는 하나만 출력한다.

이를 SQL 구문으로 바꾸면 다음과 비슷하다.

`select * from <COLLNAME> where <COND>`

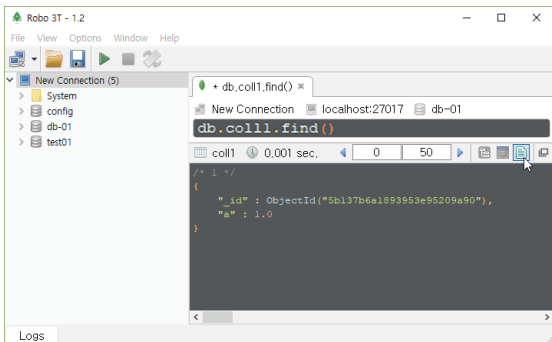


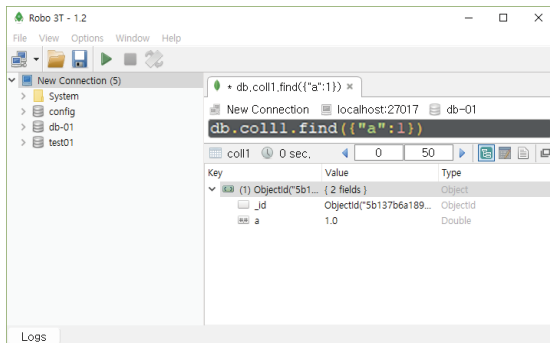
그림 8-25 JSON 형식으로 문서 보기

\_id

오브젝트 ID라고 한다. 오브젝트 ID는 추가된 문서의 고유 식별자로 키 역할을 한다. 관계형 데이터베이스와 달리 MongoDB에서는 추가되는 모든 문서에 이처럼 오브젝트 ID가 추가됩니다



## 8.6 | MongoDB에 데이터 액세스



특정 조건을 만족하는 것 찾기

<COND> 안에 '키:값' 형식으로 구문을 넣는다.  
컬렉션의 각 문서에 대해 최상위에 a가 있으며,  
그것의 값이 1인 문서들을 찾고 있음을 알 수 있다.

그림 8-26 키 a의 값이 1인 문서 찾기

a가 1보다 큰 것을 찾는 방법

```
db.coll1.insert({"a": 2})
db.coll1.insert({"a": 3})
```

insert 명령어로 a = 2, a = 3인 문서를 추가.

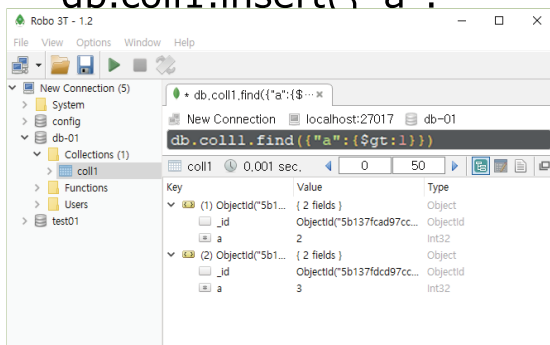


그림 8-27 a가 1보다 큰 문서들을 찾는 구문과 실행 결과

## 8.6 | MongoDB에 데이터 액세스

a가 여러 값 중 하나인 것을 찾기

```
select * from coll1 where a=1 or  
a=2
```

-----● SQL 은 다음과 같이 입력.

```
{"a":1}
```

-----● MongoDB에서는 JSON 형태로 검색 조건을 만들어야 한다.

"a = 1이다."를 정의

```
[{"a":1}, {"a":2}]
```

-----● "a = 1이다, a = 2다."를 배열 객체로 정의한다.  
배열은 대괄호 [] 구문 블록 안에 넣는다.

```
{$or:[{"a":1}, {"a":2}]}
```

-----● 이 배열 객체는 or 연산에 쓰일 것이므로 다음과 같이 표시함.

a = 1 or a = 2 형태의 구문보다는 가독성이 떨어지지만 MongoDB에서는 이렇게 써야한다.

```
db.coll1.find({$or:[{"a":1}, {"a":2}]});
```

-----● 이 구문을 find 함수에 넣는다.

## 8.6 | MongoDB에 데이터 액세스

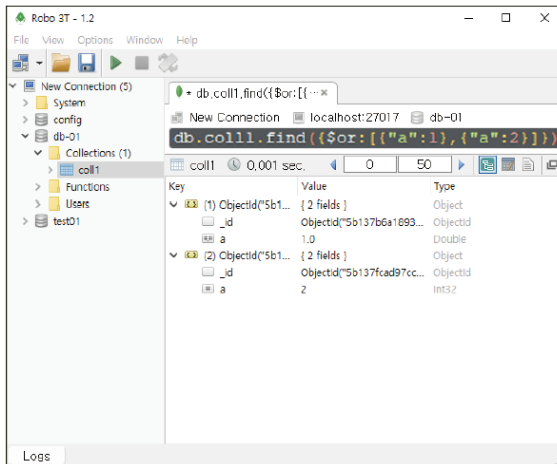
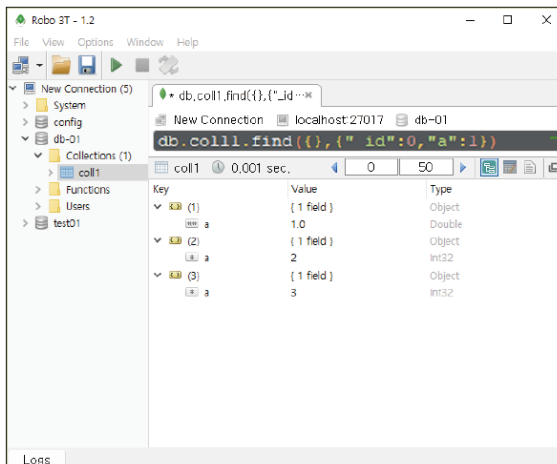


그림 8-28 a = 1 또는 a = 2를 만족하는 도큐먼트 찾기

우리가 원하는 일부만 출력하기



● {"\_id":0,"a":1}

"0이면 출력하지 마라, 1이면 출력하라."라는 의미로, 이렇게 두 번째 인자에 JSON 구문을 넣는다.

도큐먼트 안의 속성, 즉 필드에 대응하는 이름을 먼저 입력하고 0이나 1을 넣어서 출력할지 말지를 지정하면 된다.

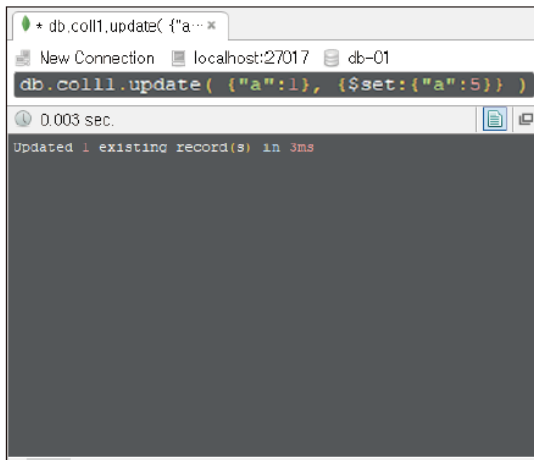
그림 8-29 \_id 속성은 보여 주지 말고, a 속성은 보여 주기

## 8.6 | MongoDB에 데이터 액세스

### • 8.6.3 업데이트(update)

MongoDB에 있는 도큐먼트 내용 일부를 변경하는 방법 : `db.coll1.update()` 구문 사용.

K가 V인 도큐먼트를 찾고, 그 도큐먼트의 K2 를 V2로 바꾸고 싶으면 첫 번째 인자에 `{K:V}`를 넣고 두 번째 인자에 `{$set:{K2:V2}}`를



첫 번째 인자에 검색 조건을 넣는다.

`{"a":1}`, 즉 `a = 1`인 것을 찾겠다는 의미.

두 번째 인자에는 무슨 값을 무엇으로 바꿀 것인지 넣는다.

`{"a":5}`, 즉 `a = 5`로 바꾸라는 의미.

그림 8-30 도큐먼트 내용 일부 변경

`update coll1 set a=5 where a=1`

-----● SQL 은 다음과 같이 입력.

## 8.6 | MongoDB에 데이터 액세스

### • 8.6.4 지우기(delete)

문서를 지울 때는 `db.coll1.remove()` 함수를 사용하며, 인자에는 지울 조건을 넣는다.  
`find()` 함수에 넣었던 조건과 같은 방식으로 수행한다.

a = 3인 문서를 찾아서 지우려면

```
db.coll1.remove({"a":3})
```

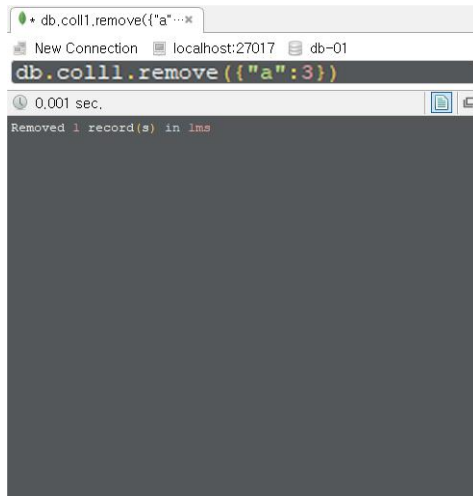


그림 8-32 문서 지우기

CRUD( Create, Read, Update, Delete)

컬렉션에 문서를 추가하고, 읽고, 업데이트하고, 삭제하기

## 8.7 | 성능 분석 기능

### • 성능 분석 기능

내부에서 일어나는 일 중에서 어떤 일이 많은 시간을 차지하는지 분석하는 용도

MongoDB에서는 실행 구문 오른쪽에 `.explain("executionStats")` 구문을 넣으면 성능 통계 정보가 나오며, 이는 성능 문제가 발생했을 때 병목 지점을 찾는 시작점이 된다.

Key	Value	Type
(1)	{ 4 fields }	Object
queryPlanner	{ 6 fields }	Object
plannerVersion	1	Int32
namespace	db-01.coll1	String
indexFilterSet	false	Boolean
parsedQuery	{ 0 fields }	Object
winningPlan	{ 2 fields }	Object
rejectedPlans	[ 0 elements ]	Array
executionStats	{ 6 fields }	Object
executionSuccessful	true	Boolean
nReturned	3	Int32
executionTimeMillis	0	Int32
totalKeysExamined	0	Int32
totalDocsExamined	3	Int32
executionStatistics	{ 13 fields }	Object
serverInfo	{ 4 fields }	Object
host	imays-home	String
port	27017	Int32
version	3.6.5	String
gitVersion	a20ecd3e3a174162052ff9...	String
ok	1.0	Double

그림 8-33 `.explain("executionStats")`로 성능 통계 정보 확인

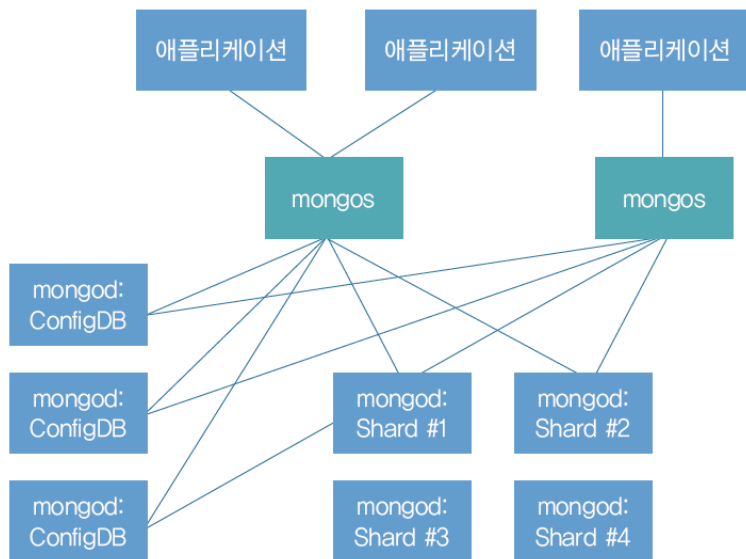
Key	Value	Type
(1)	{ 4 fields }	Object
createdCollectionAutomatically	false	Boolean
numIndexesBefore	1	Int32
numIndexesAfter	2	Int32
ok	1.0	Double

그림 8-34 MongoDB에서는 도큐먼트 안에 있는 '이름:값'에서 '이름'에 인덱스 추가

## 8.8 | MongoDB 수평 확장

- MongoDB로 샤딩을 구성하는 방법.

MongoDB 샤드 클러스터( shard cluster)를 구성할 수 있다.



mongos는 App, 즉 서버에서 명령을 받는다. 그리고 적절한 샤드로 명령을 보내 결과를 받는 중계자 역할을 한다.

ConfigDB는 이러한 MongoDB 샤드 클러스터에 관한 정보를 담고 있다.

기존 MongoDB 데이터베이스 서버, 즉 mongod.exe가 그대로 사용된다.

각 샤드는 샤드 키( shard key)로 구분되는 도큐먼트를 분배해서 가진다. 마찬가지로 mongod.exe가 그대로 사용된다.

그림 8-35 MongoDB 샤드 클러스터 구성

MongoDB 웹 사이트: <http://bit.ly/2rTOUTi>

MongoDB 샤드 세팅: <http://bit.ly/2RkRHRW>

샤딩 관련 명령: <http://bit.ly/2AfMJfP>

## 8.9 | 게임 서버에서 MongoDB 명령 실행

- MongoDB는 C++, C#, 자바, 루비, 파이썬 등 다양한 언어에서 실행할 수 있는 클라이언트를 제공

MongoDB 샤드 클러스터( shard cluster)를 구성할 수 있다.

```
client = new  
client("mongodb://localhost:27017");
```

MongoDB와 연결 객체를 만든다.  
입력 매개변수로는 MongoDB의  
포인트를 문자열 형태로 입력한다.

```
db =  
client["mydb"];
```

연결 객체에서 원하는 DB  
인스턴스를 액세스할 객체를 얻어  
온다.  
입력 매개 변수로 DB 인스턴스의  
이름을 넣어 준다.

```
coll =  
db["mycollection"];
```

DB 인스턴스 객체에서 컬렉션을  
액세스할 객체를 얻어 온다.  
매개변수로 컬렉션 이름을 넣는다.



## 8.9 | 게임 서버에서 MongoDB 명령 실행

새 문서를 추가한다.

여러 MongoDB API는 BSON(Binary JSON) 형식의 객체를 이용해서 문서를 액세스한다.  
BSON은 JSON과 같은 역할을 하나, 데이터 포맷으로 텍스트대신 BSON 객체의 트리 구조를 사용한다.

```
doc = new BsonDocument // BSON 객체를 생성한다. 단 아래 매개변수로 생성한다.
{
    { "name", "John" },           // 최종 하위 노드
    { "address", new BsonDocument // 하위 노드. 단 value가 또 다른
    BSON 객체다.
        {
            { "City", "Seoul" }, // 최종 하위 노드
            { "Street", "Nambu-street" } // 최종 하위 노드
        }
    }
};
```

```
coll.insert(doc);
```

BSON 객체를 만든 후에는 이를 컬렉션에 추가하는 함수를 실행한다. 매개변수로 BSON 객체를 넣는다.

## 8.10 | 요약 및 더 알아보기

- NoSQL과 관계형 데이터베이스의 차이(요약)

표 8-1 관계형 데이터베이스와 NoSQL 비교

구분	관계형 데이터베이스	NoSQL
레코드 구조	테이블 안의 모든 레코드가 동일한 데이터 구조(필드)를 가져야 한다.	컬렉션 안의 모든 문서가 서로 다른 데이터 구조(트리)를 가져도 된다.
중요하게 여기는 것	데이터의 원자성과 일관성을 중요하게 여긴다.	데이터의 가용성(데이터 액세스 요청을 신속하게 응답)과 수평 확장성을 중요하게 여긴다.
질의 기능	질의에서 할 수 있는 기능이 풍부하며 문법이 표준화되어 있다.	질의 기능이 상대적으로 적고 표준이 없다.

NoSQL은 게임 플레이어의 데이터를 저장하는 용도로 간혹 쓰이지만, 게임 플레이어 데이터보다는 주로 로그 분석이나 통계 분석 등 용도로 활용된다. 어떤 개발 프로젝트에서는 데이터베이스 설계를 확정하기 전까지 NoSQL을 사용하여 개발을 더 빠르게 유도하기도 한다.

# 8.10 | 요약 및 더 알아보기

- MongoDB와 관련해서 알아야 할 추가적인 특징과 활용법.
  - 오브젝트 ID는 문서를 추가할 때마다 자동 할당되지만, 직접 수동으로 할당해도 된다.
  - 한 번 설정된 샤드 키는 변경할 수 없다. 변경하려면 샤드를 재구성해야 한다.
  - MongoDB는 인덱스 비용이 관계형 데이터베이스보다 큰 경향이 있다. 이는 MongoDB버전이 계속 올라가면서 변경될 여지도 있으니, 관심을 갖고 지켜보자.
  - 결과적 일관성 때문에 일시적으로 잘못된 데이터가 얻어질 수 있음을 인지하고 프로그래밍을 해야 한다.
  - MongoDB에는 insert, update 말고도 upsert라는 기능도 있다. upsert는 “없으면 넣고 있으면 있는 것을 수정하라.”라는 의미.  
예를 들어 “A = 1인 문서의 X를 2로 바꾸어라.”를 실행할 때, A = 1인 문서가 없으면 A = 1인 새 문서를 삽입하면서 그것의 X를 2로 세팅한다.
  - MongoDB에는 MMAP나 WiredTiger 같은 내부 엔진들이 있다. 장단점에 따라 골라서 사용하면 된다.
  - MongoDB에 대한 더 다양한 자료는 <https://docs.mongodb.com>이나 여러 인터넷 자료 및 도서를 살펴보자.
  - MongoDB뿐만 아니라 Couchbase나 Cassandra, Riak 등 다른 NoSQL 데이터베이스도 있다.