2. MongoDB 설치 및 데이터 처리

MongoDB란?

MongoDB 시작과 종료

데이터 처리

1. MongoDB란?

MongoDB란?

- Humongos라는 회사의 제품 명이었으며 10gen으로 회사명이 변경 ->현재 Mongodb.inc로 변경
- JSON Type의 데이터 저장 구조를 제공 {name :"박경미"}
  - (Standard ECMA-262 3rdEdition-1999을 근거로 하는 JavaScript 형태의 데이터 표현방식을 근거로 한다. *[European Computer Manufacturers Association]*)
- Sharding(분산)/Replica(복제) 기능을 제공.
- MapReduce(분산/병렬처리) 기능을제공.
- CRUD(Create, Read, Update, Delete) 위주의 다중 트랜잭션 처리도 가능
- Memory Mapping 기술을 기반으로 Big Data 처리에 탁월한성능을 제공

2. MongoDB 설치

설치환경

1) 설치 가능 플랫폼

- Windows 32 bit / 64 bit
- Linux 32 bit / 64 bit
- Unix Solaris i86pc / 64 bit
- Mac OS X-32bit / 64 bit

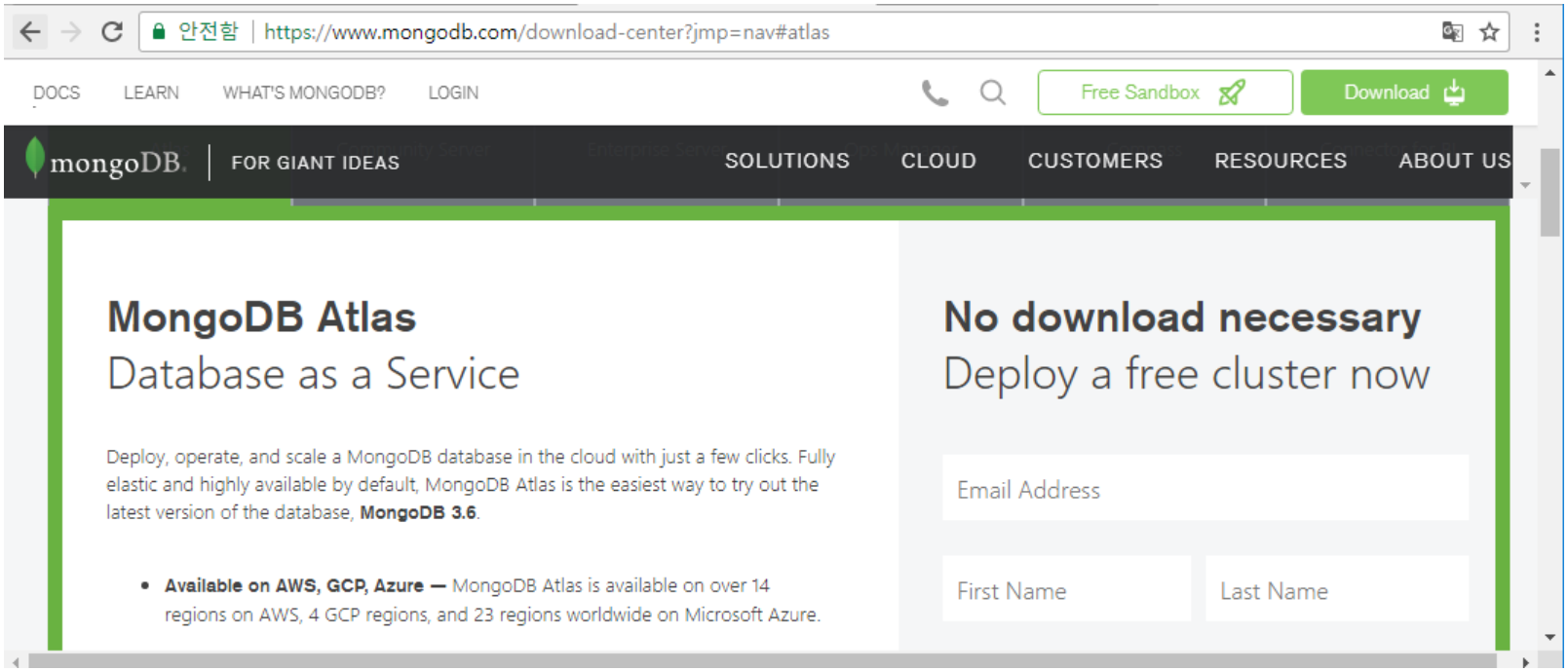
2) 지원 Language Driver

- C / C# / C++
- Java / Java Script
- Perl / PHP / Python
- Ruby / Erlang / Haskell / Scala

mongoDB

2. MongoDB 설치

- download url: https://www.mongodb.com



- zip 파일 다운로드 후 압축해제
- path 설정(d:₩mongodb₩bin)

mongoDB

3. MongoDB 시작과 종료

- 데이터베이스 폴더 작성
  - d:₩mongodb₩text 작성
- 버전학인 및 도움말 보기

```
c:₩>mongod –help
c:₩>mongod --version
```

- Mongodb 서버 및 클라이언트 시작

```
c:₩>mongod --dbpath d:₩mongodb₩test          # 서버실행
c:₩>mongo #클라이언트 실행
```

- 서버 및 클라이언트 실행

```
>use admin    # 서버 종료를 위해 반드시 admin을 사용해야함
>db.shutdownServer()    # 서버 종료
>exit                   # ctr+c 클라이언트 종료
```

mongoDB

mongodb 상태, 논리적구조, 호스트 상태 확인

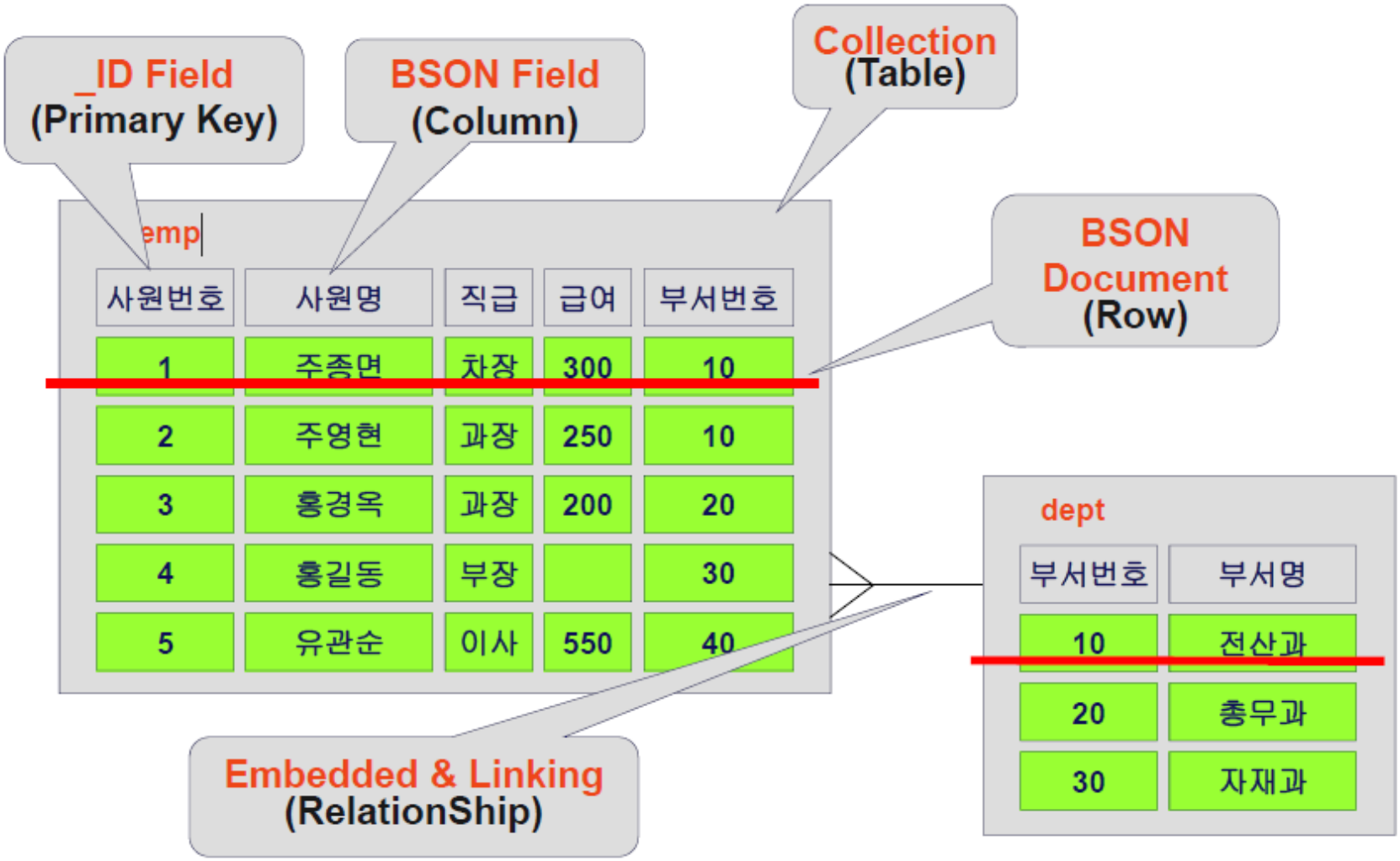
> show dbs #데이터베이스 상태 확인

> use test #데이터베이스 선택

> db.stats() #data 데이터베이스 내에 저장되어 있는 논리적 구조 확인

> db.hostInfo() #data 데이터베이스 호스트 상태 확인

용어


_ID Field
(Primary Key)
BSON Field
(Column)
Collection
(Table)
BSON
Document
(Row)
emp
사원번호
사원명
직급
급여
부서번호
1
주종면
차장
300
10
2
주영현
과장
250
10
3
홍경옥
과장
200
20
4
홍길동
부장
30
5
유관순
이사
550
40
dept
부서번호
부서명
10
전산과
20
총무과
30
자재과
Embedded & Linking
(RelationShip)

mongoDB

4. 데이터 처리

Collection 생성
>use data
>db.createCollection("emp",{capped:true, size:2100000000, max:500000});

capped: 해당공간이 모두 사용되면 다시 처음부터 재사용 할 수 있는 데이터 구조를 생성
size : 해당 Collection의 최초 새성 크기 지정 가능

#collection 보기
>show collections

emp collection 현재 상태 정보 분석
>db.emp.stats()

#emp collection 이름 바꾸기
>db.emp.renameCollection("employee")

#employee collection 삭제
>db.employee.drop()

4.1 Collection 생성과 관리

Collection 생성/삭제

```
> db.createCollection (“emp”, { capped : false, size:8192 });
{ "ok" : 1 }
> show collections
emp
>
> db.emp.validate()     #Collection의 현재 상태 및 정보 분석
 {
"ns" : "test.emp",
"firstExtent" : "0:61000 ns:test.emp",
"lastExtent" : "0:61000 ns:test.emp",
"extentCount" : 1,
"datasize" : 0,
"nrecords" : 0,
"lastExtentSize" : 8192,
> db.emp.renameCollection( “employees" ) #해당 Collection 이름 변경
> db.employees.drop();    # 해당 Collection 삭제
true
```

capped : 해당 공간이 모두 사용되면 다시 처음부터 재 사용할수 있는데이터 구조를생성할 때
size : 해당 **Collection**의 최초 생성 크기 지정 가능

4.2 Insert & update&remove

SQL	mongoDB
careate table emp(no number(4), ename varchar2(10))	db.createCollection({"emp"})
insert into emp values(1, 'aaa')	db.emp.insert({no:1, ename:"aaa"})
update emp set ename='bbb' where no=1	db.emp.update({no:1},{$set: {ename:"bbb"})
delete from emp where no=1	db.emp.remove({no:1})

collection에 데이터 입력

```
> 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={ename='smith'}
> n={empno=1101}

> db.things.save(m)
> db.things.save(n)
> db.things.find()
> db.things.insert({empno:1102,ename:'king'})

> for(var n=1103;n<=1120;n++) db.things.save({n:n,m:'test'})

> it # 출력된 결과가 20개를 초과하면 다음 화면으로 넘어감
```

데이터 수정

```
> db.emp.update ({ eno:1101 }, { $set: { fname : “JOO” } } );
> db.emp.update ({ eno:1102 }, { $set: { job : “CHIEF” } } );
> db.emp.update ({ eno:1103 }, { $set: { lname : “STANFORD” } } );
> db.emo.find()
> db.emp.find().sort ({eno:-1});

> db.things.update({n:1103}, {$set: {ename: "standford", dept:"research"}})
> db.things.update({n:1104}, {$set: {ename: 'john', dept:'inventory'}})
> db.things.update({n:1105}, {$set: {ename:‘ joe', dept:'accounting'}})
> db.things.update({n:1106}, {$set:{ ename: 'king', dept:'research'}})
> db.things.update({n:1107}, {$set: {ename: 'adams', dept:'personel'}})
> db.things.update({n:1108}, {$set: {ename: 'blake', dept:'research'}})
> db.things.update({n:1109}, {$set: {ename: 'smith', dept:'inventory'}})
> db.things.update({n:1110}, {$set: {ename: 'allen', dept:'accounting'}})
> db.things.update({n:1111}, {$set: {ename: 'clief', dept:'research'}})
>db.things.update({n:1120}, {$set: {ename: 'miller', dept:'personel'}})

>db.things.save({empno:1101,ename:'Blake', dept:'account'})
```

데이터 삭제

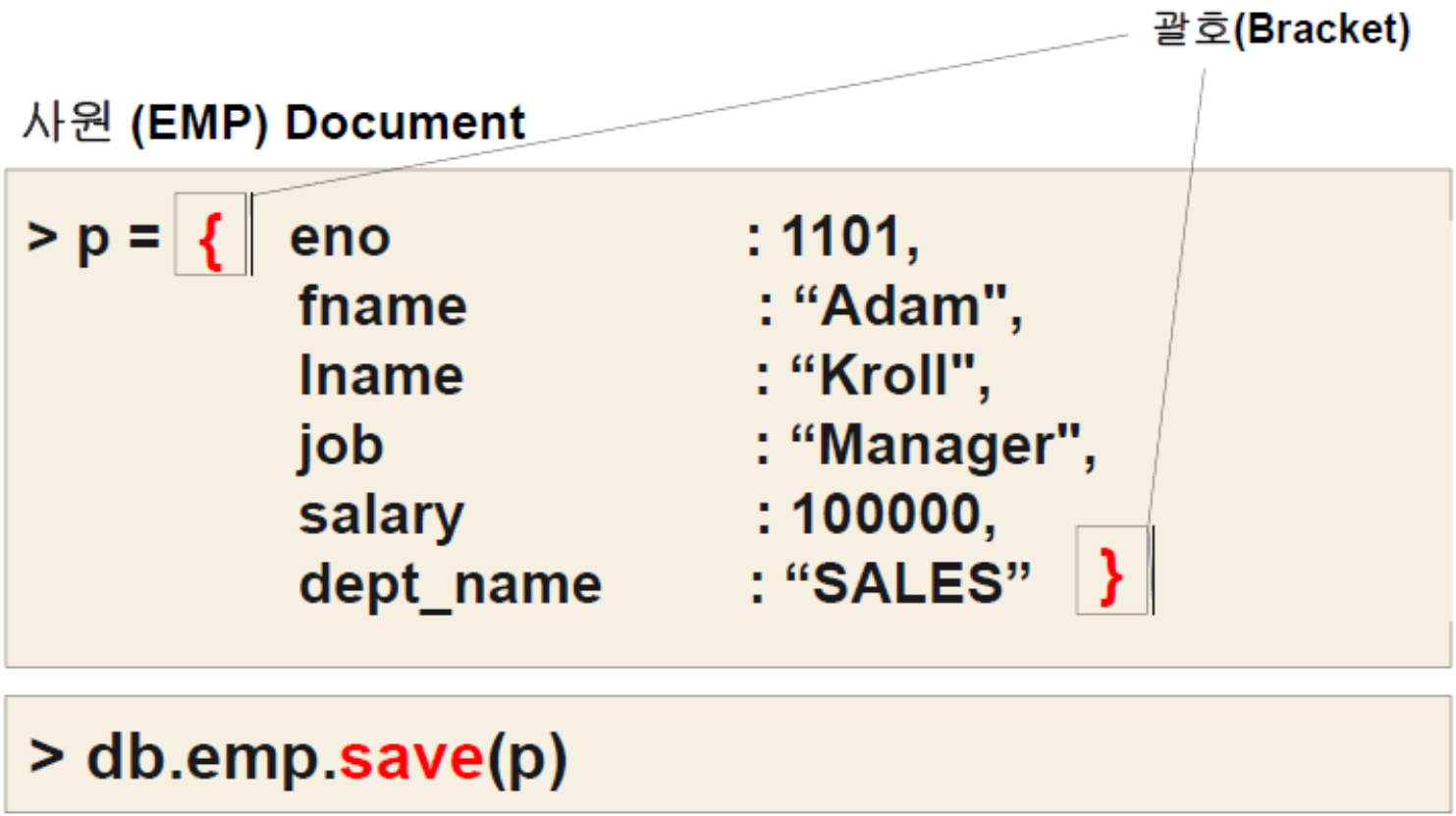
```
> db.emp.remove ({ eno: 1101});

> db.things.remove({m:'test'})
> db.things.remove({})
> db.things.find()
> db.things.drop()
```

insert, update, save 차이점

- **insert** : Collection에 하나의 Document를 최초로 저장할 때 사용
- **update**: 하나의 Document에서 특정 필드의 값을 변경할 때 사용
- **save**: 하나의 Document에서 특정 필드의 값을 변경하더라도 Document 단위로 변경하는 방법

4.3 JSON타입과 BSON타입

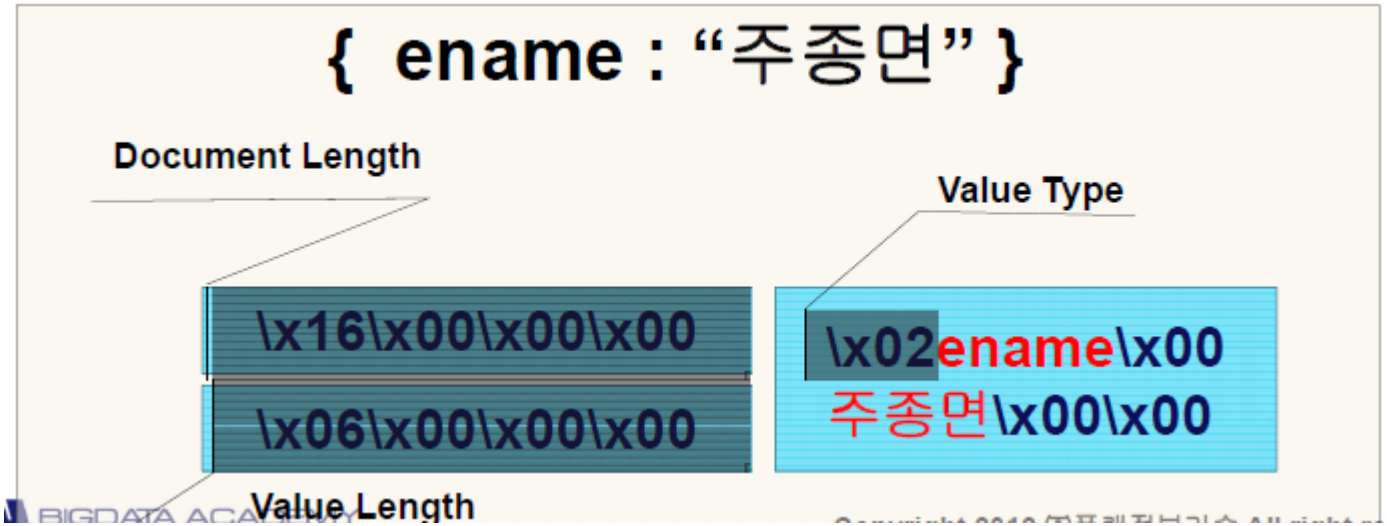
- JSON (Java Script Object Notation)



mongoDB

BSON (Binary Serial Object Notation)

- MongoDB에서 모든 데이터는 JSON 형태로 표시되지만 데이터베이스내에서는 BSON 형태의 Binary형태의 데이터로 변환되어 저장
- 경량의 데이터 교환 형식인 JSON(JavaScript Object Notation) 타입을 근거로 하며 사람이 읽고 쓰기에 용이하며 기계가 분석하고 생성하기에 용이하다. (Name과 Value로 구성됨)
- JavaScript Programming Language와 Standard ECMA-262 3rd Edition-1999을 근거로 한다.



mongoDB

4.4 Data Type 종류

- **Data Type 종류**

종류	내용
JSON	atring, 32/64bit integer, double, array, boolean, binary data, null, symbol, javascript code, object
DATE	날짜 정보 저장 예 : Date()
TIMESTAMP	년, 월, 일, 초 mile scond 단위까지 저장
Internationalized STRINGS	UTF-8 다국적 문자 값을 저장할 수 있는 BSON 데이터 타입
OBJET ID	OBJECT_ID 값을 저장할 수 있는 데이터 타입

4.4 Data Type 종류

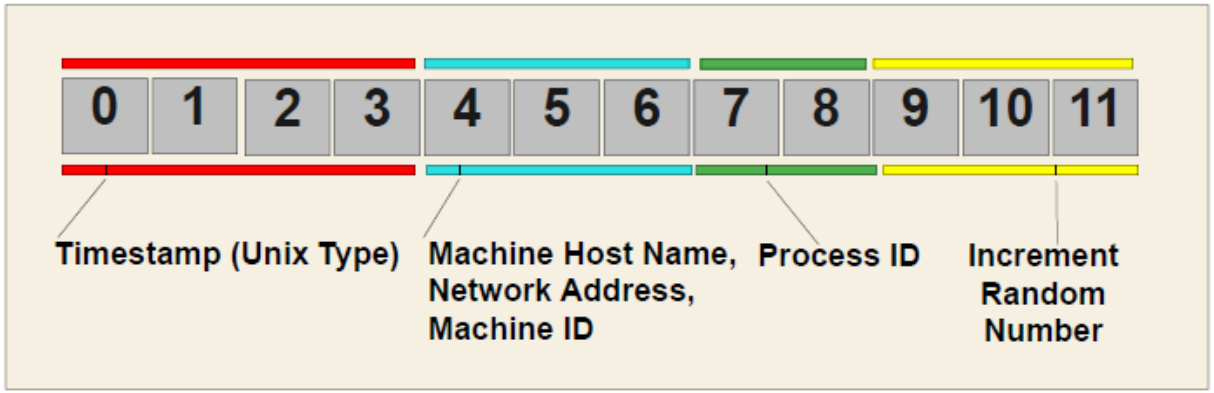
요소명		Data Type	
기능		포멧	설명
null		{"x" : null}	null값과 존재하지 않는 Field를 표현하는데 사용.
불린		{"x" : true}	참과 거짓을 구분할 때 사용.
숫자	실수	{"x" : 3.14}	숫자는 8바이트 부동소수점이 기본 형
	정수	{"x" : 3}	일반적인 정수도 8바이트 부동소수점을 사용함.
		{"x" : NumberInt("3")}	4바이트 정수 표현
		{"x" : NumberLong("3")}	8바이트 정수 표현
문자열		{"x" : "foobar"}	UTF-8 문자열을 표현할 때 사용.
날짜		{"x" : new Date()}	1970년 1월 1일부터의 시간을 1/1000 초 단위로 저장
정규표현식		{"x" :/foobar/i}	자바스크립트 정규표현식 문법사용이 가능함.
배열		{"x" : ["a", "b", "c"]}	값의 셋트나 리스트를 배열로 표현
내장 문서		{"x" : {"foo" : "bar"}}	문서는 다른 문서를 포함할 수도 있음.
객체 ID		{"x" : ObjectID()}	문서용 12바이트 ID. RDB의 PK와 같은 개념
코드		{"x" : function() { /* ..... */ }}	임의의 코드를 포함 할 수도 있음.
코드		{"x" : function() { /* ..... */ }}	임의의 코드를 포함 할 수도 있음.

OBJECT ID type

- RDB에서 rowid와 유사한 데이터 속성, 테이블에서 Document를 유일하게 식별할 수 있는 값, RDBMS의 primary key와 같이 유일한 값으로 구성

Object ID type의 특징

- BSON Object ID는 12 Byte의 Binary 값으로 구성되어 있다.
- 하나의 Collection에 하나의 Document를 입력하면 반드시 하나의 유일한 값의 Object ID가 부여된다.
- MongoDB에서 제공하는 Default Object ID와 별도로 사용자 직접 유일한 값을 Object ID로 부여할 수 있다.



mongoDB

BSON ObjectID 데이터 타입

```
> use test
> p={eno:1101,fname:"adam",lname:"kroll",
        job:"manager",salary:1000,dept_name:"sales"}
> db.emp.save(p)
> p

> db.emp.findOne({_id:ObjectId("5a6359b63bb0517a8f26f0a0")})

> db.emp.findOne({_id:new ObjectId("5a6359b63bb0517a8f26f0a1")})
```

mongoDB

JSON Type

- 숫자, 문자, Binary 타입 데이터를 저장할 수 있는 타입

```
> p={
"_id" : ObjectId("201301010000000000000001"),
"v_date" : ISODate("2018-01-01T14:22:46.777Z"),
"v_bin": BinData(0,"2feaces232csdceq2424"),
"v_char" : "park gyeong mi",
"v_num" : 12456755,
"v_arr" : ["sddd@naver.com","ccc@kkk.com"],
"v_bignum" : NumberLong(1254000)}

> db.data_att.save(p)
> db.data_att.find()
> db.data_att.drop()
```

mongoDB

array type

- Mongodb는 c/c++/java언어에서 제공하는 배열 타입 데이터속성 제공

```
# array type
> for(var n=1103;n<1120;n++)
            db.things.save({empno:n, ename:"test", sal:1000})
> db.things.find()

> var cursor=db.things.find()

> var cursor=db.things.find()
> while(cursor.hasNext()) printjson(cursor.next())

> var cursor=db.things.find()
> printjson(cursor[17])

-index : 0부터 시작
var arr=db.things.find().toArray()
arr[10]
```

Date와 Timestamp type

- **Date type**

```
#date type
x=new Date()
x.toString()

d=new ISODate()
d.getMonth()
d.getYear()
d.getDate()
```

- **Timestamp**

```
db.foo.insert({x:1, y:new Timestamp()})
db.foo.find()
db.foo.drop()

db.foo.insert({y:new Timestamp, x:3})
db,foo.find({},{_id:0})

for(var i=0; i<=10; i++) db.foo.insert({v:new Timestamp(), x:i})
db.foo.find()
```

Sequenc Number Type

- **oracle 의 sequence와 유사**

```
function seq_no(name){
        var ret=db.seq_no.findAndModify({
        query:{_id:name},
        update:{$inc:{next:1}},
        "new":true, upsert:true
        });
        return ret.next
}

db.order_no.insert({_id:seq_no("order_no"),name:"jimmy"})
db.order_no.insert({_id:seq_no("order_no"),name:"chad"})
db.order_no.find()
db.order_no.drop()
db.seq_no.drop()
```

4.5. 연산자의 종류

비교 연산자와 논리 연산자

종류	유형	
비교 연산자	$cmp	두 값을 비교하여 앞의 값이 크면 양수, 작으면 음수, 같으면 0을 return 함
	$eq	두 값이 같으면 True, 다르면 False
	$gt	앞의 값이 크면 True, 작거나 같으면 False
	$gte	앞의 값이 크거나 같으면 True, 작으면 False
	$lt	앞의 값이 작으면 True, 크거나 같으면 False
	$lte	앞의 값이 작거나 같으면 True, 크면 False
	$ne	두 값이 다르면 True, 같으면 False
Boolean 연산자	$and	모두 True이면 True
	$not	하나라도 True이면 True
	$or	해당 결과값의 반대

mongoDB

SQL vs Mongodb 문법

SQL	Mongodb 문법
SELECT * FROM SCOTT.EMP;	db.employees.find({},{_id:0})
SELECT * FROM SCOTT.EMP WHERE EMPNO = 7369	db.employees.find({empno:7369}) .forEach(printjson)
SELECT ROWID, ENAME FROM SCOTT.EMP WHERE EMPNO = 7900	db.employees.find({empno:7900}, {ename:""}) .forEach(printjson)
CREATE INDEX SCOTT.EMP_ENAME ON SCOTT.EMP (ENAME); SELECT empno, ename FROM SCOTT.EMP WHERE ENAME >= 'ALLEN' AND ENAME < 'SCOTT'	db.employees.ensureIndex({ename:1}) db.employees.find({},{_id:0, empno:1, ename:1}).min({ename:"ALLEN"}) .max({ename:"SCOTT"})
CREATE INDEX SCOTT.EMP_DEPTNO ON SCOTT.EMP (DEPTNO); SELECT EMPNO, ENAME, HIREDATE FROM SCOTT.EMP WHERE DEPTNO BETWEEN 20 AND 30;	db.employees.ensureIndex({deptno:1}) db.employees.find({$min:{deptno:20}, $max:{deptno:30}, $query:{}}, {_id:0, empno:1, ename:1, hiredate:1})
SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE EMPNO > 7500 AND EMPNO <= 7600;	db.employees.find({empno:{$gt:7500, $lte:7600}}, {_id:0,empno:1,ename:1})
SELECT COUNT(*) FROM SCOTT.EMP; SELECT COUNT(*) FROM SCOTT.EMP WHERE EMPNO > 7900;	db.employees.count() db.employees.find({empno: {'$gt':7900}}).count()
SELECT DISTICT DEPTNO FROM SCOTT.EMP;	db.employees.distinct("deptno")

SQL vs Mongodb 문법

SQL	Mongodb 문법
SELECT DEPTNO, SUM(SAL) CSUM FROM SCOTT.EMP WHERE HIREDATE >= '01-01-1980' AND HIREDATE <= '31-12-1981' GROUP BY DEPTNO;	db.employees.group({key: {deptno:true}, cond: {"hiredate": {$gte: "01-01-1980", $lt: "31-12-1981"}}, reduce: function(obj, prev) {prev.csum += obj.sal; }, initial: { csum:0}})
SELECT ENAME, JOB FROM SCOTT.EMP WHERE DEPTNO = 10 ORDER BY ENAME DESC;	db.employees.find({deptno:10},{_id:0, ename:1, job:1}) .sort({ename:-1})
SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE SAL <= 1000;	db.employees.find({sal:{$lte:1000}}, {_id:0, empno:"", ename:""})
SELECT EMPNO, ENAME, SAL FROM SCOTT.EMP WHERE SAL >= 1500 AND SAL <= 5000;	db.employees.find({sal:{$gte:1500, $lte:5000}}, {_id:0, empno:1, ename:1, sal:1})
SELECT EMPNO, DEPTNO FROM SCOTT.EMP WHERE DEPTNO IN (10, 30);	db.employees.find({deptno:{$in:[10,30]}}, {_id:0, empno:1, deptno:1})
SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE DEPTNO = 10 OR DEPTNO = 30;	db.employees.find({$or:[{deptno:10}, {deptno:30}]}, {_id:0, empno:1, ename:1})
SELECT EMPNO, DEPTNO FROM SCOTT.EMP WHERE DEPTNO = 10 AND SAL > 1000;	db.employees.find({$and:[{deptno:10}, {sal:{$gt:1000}}]}, {_id:0, empno:1, deptno:1})
SELECT EMPNO, COMM FROM SCOTT.EMP WHERE COMM IS NOT NULL;	db.employees.find({comm:{$exists:true}}, {_id:0, empno:1, comm:1})
SELECT EMPNO, COMM FROM SCOTT.EMP WHERE COMM IS NULL;	db.employees.find({comm:{$exists:false}},{_id:0, empno:1, comm:1})

exists = 필드가 존재하는지

산술 연산자와 문자 연산자

종류	유형	설명
산술 연산자	$add	두 값을 합칩니다.
	$devide	앞의 값에서 뒤에서 값을 나눕니다.
	$mod	앞의 값에서 뒤의 값을 나눈 나머지 값을 return 합니다.
	$multiply	두 값을 곱합니다.
	$subtract	앞의 값에서 뒤의 값을 뺍니다.
문자 연산자	$strcasecmp	앞의 문자열이 뒤의 문자열보다 크면 양수, 작으면 음수, 같으면 0을 return
	$substr	앞의 문자열에서 첫번째 숫자 위치에서 두번째 숫자 길이만큼의 sub-string을 return
	$toUpper	대문자로 return 합니다
	$toLower	소문자로 return 합니다.

실습

```
-- comm 의 타입이 double 인 것을 검색
db.employees.find({comm:{$type:1}},{empno:"", comm:""})

-- comm 의 타입이 string 인것을 검색
db.employees.find({comm:{$type:2}},{empno:"", comm:""})

-- ename이 S로 시작해서 H로 끝나는 것을 검색, 'i' : 대소문자 구분 없음
db.employees.find({ename:{$regex:'S.*H', $options: 'i'}},{empno:1, ename:1})

-- ename이 S로 시작해서 H로 끝나는 것을 검색, 'i' : 대소문자 구분 없음
db.employees.find({ename:{$regex:'s.*h', $options: 'i'}},{empno:1, ename:1})

-- 'm' : 대소문자 구분함
b.employees.find({ename:{$regex:'s.*h', $options: 'm'}},{empno:1, ename:1})
db.employees.find({ename:{$regex:'S.*H', $options: 'm'}},{empno:1, ename:1})
```

create & alter

SQL문장	MongoDB 스키마 문장
CREATE TABLE members(mem_no varchar(30), age number, type char(1), PRIMARY KEY (mem_no));	db.members.insert({ mem_no : "T2013001", age : 49, type : "ACE"}) 또는 db.createCollection("members")
ALTER TABLE members ADD regist_date DATE;	
ALTER TABLE members DROP COLUM regist_date;	
CREATE INDEX i_members_type ON members(type);	db.members.ensureIndex({ type:1})
CREATE INDEX i_members_type_no ON members(type, mem_no DESC);	db.members.ensureInde({ type:1, mem_no: -1})
DROP TABLE members;	db.member.drop()

mongoDB

select 문-1

SQL문	MongoDB문
SELECT * FROM members;	db.members.find()
SELECT rowid,mem_no,type FROM members;	db.members.find({}, {mem_no:1, type:1})
SELECT mem_no, type FROM members;	db.members.find({}, { mem_no;1, type:1, _id:0})
SELECT * FROM members WHERE type = "ACE";	db.members.find({type:"ACE"})
SELECT mem_no, type FROM members WHERE type = "ACE";	db.members.find({type:"ACE"}, { mem_no;1, type:1, _id:0})
SELECT * FROM members WHERE type != "ACE";	db.members.find({type: { $ne: "ACE"} })
SELECT * FROM members WHERE type = "ACE“ AND age = 49;	db.members.find({type:"ACE", age:49})
SELECT * FROM members WHERE type = "ACE“ OR age = 49;	db.members.find({$or: [{type:"ACE"}, {age:49}] })

mongoDB

select 문-2

SQL	Mongodb
SELECT * FROM members WHERE age > 45;	db.members.find({age: { $gt: 45} })
SELECT * FROM members WHERE age < 55;	db.members.find({age: { $lt: 55} })
SELECT * FROM members WHERE age > 45 AND age <= 55;	db.members.find({age: { $gt: 45 , $lte:55}})
SELECT * FROM members WHERE mem_no like "%2013%";	db.members.find({ mem_no: /2013/})
SELECT * FROM members WHERE mem_no like "T%";	db.members.find({ mem_no: /^T/})
SELECT * FROM members WHERE type = "ACE“ ORDER BY mem_no ASC;	db.members.find({ type: "ACE" }).sort({mem_no:1})
SELECT * FROM members WHERE type = "ACE“ ORDER BY mem_no DES;	db.members.find({ type: "ACE" }) .sort({mem_no:-1})

mongoDB

select 문 -3

SQL	Mongodb
SELECT COUNT(*) FROM members;	db.members.count() 또는 db.memers.find().count()
SELECT COUNT(mem_no) FROM members;	db.members.count({ mem_no: {$exists:true} }) db.members.find({ mem_no: {$exists:true} }).count()
SELECT COUNT(*) FROM members WHERE age > 45;	db.members.count({ age: {$gt:45}}) db.members.find({ age: {$gt:45}}).count()
SELECT DISTINCT type FROM members;	db.members.distinct("type")
SELECT * FROM members WHERE rownum =1;	db.members.findOne() 또는 db.members.find().limit(1)
EXPLAIN PLAN SELECT * FROM members WHERE type = "ACE";	db.members.find({ type:"ACE"}).explain()

mongoDB

update Record 문

SQL문	MongoDB문
UPDATE members SET type = "GOLD" WHERE age > 45	db.members.update({ age: { $gt:45 } }, { $set: { type: "GOLD" } }, { multi: true})
UPDATE members SET age = age + 3 WHERE type = "ACE"	db.members.update({ type: "ACE" }, { $inc: { age:3 } }, { multi: true})

mongoDB

Delete Records 문

SQL문	MongoDB문
DELETE FROM members WHERE type = "ACE"	db.members.remove({type: "ACE"})
DELETE FROM members	db.members.remove()