MongoDB 특징과 개념

1.MongoDB 개요

1.1 RDBMS와 NoSQL

1. RDBSM & NoSQL

NoSQL이란?

- 관계형 데이터베이스 기술은 무려 30년 동안 널리 사용되어온 대표적인 데이터저장 기술
- 최근에 클라우드 컴퓨팅 환경에서 발생하는 빅 데이터를 효과적으로 저장, 관리하는 데는 여러 가지 문제점이 발생하고 있는 것이 현실
- 문제점을 개선, 보완하기 위한 새로운 데이터 저장 기술이 요구되었는데 이것을 NoSQL이라 함

1. RDBSM & NoSQL

NoSQL의 장점

1.클라우드 컴퓨팅 환경에 적합하다

1) Open Source이다.
2) 하드웨어 확장에 유연한 대처가 가능하다
3) 무엇보다 RDBMS에 비해 저렴한 비용으로 분산처리와 병렬처리가 가능하다.

2.유연한 데이터 모델이다.

1) 비정형 데이터 구조 설계로 설계 비용 감소
2) 관계형 데이터베이스의 Relationship과 Join구조를 Linking과 Embedded로 구현하여 성능이 빠르다.

3. Big Data 처리에 효과적이다.

1) Memory Mapping 기능을 통해 Read/Write가 빠르다.
2) 전형적인 OS와 Hardware에 구축할 수 있다.
3) 기존 RDB와 동일하게 데이터 처리가 가능하다.

1. RDBSM & NoSQL

NoSQL의 제품군

1.Key-Value Database	3.Document Database
1) Amazon's Dynamo Paper 2) Data Model : Collection of K-V Pairs 3) 제품유형 : Riak, Voldemort, Tokyo*	1) Lotus Notes 2) DataModel : Collection of K-V collection 3) 제품유형 : MongoDB, CoughDB
2.BigTable Database	**4.Graph Database**
1) Google's BigTable paper 2) Data Model : Column Families 3) 제품유형 : Hbase, Casandra, Hypertable	1) Euler & Graph Theory 2) Data Model : nodes, reis, K-V on both 3) 제품유형 : AllegroGraph, Sones

* Availablity(유용성) , Consistency(일관성) , Partitioning(지속성)에 따른 제품군 구분

https://db-engines.com/en/

1.MongoDB 개요

1.2 NoSQL 적용 사례

2.NoSQL 적용 사례

사례 - Disney

- Disney Interactive Media Group
- 다양한 Game, Media Data 관리 시스템에 적용
- MySQL 바이너리 데이터 저장 한계 및 성능 문제
- ReplicaSets & Auto Sharding 유연성과 확장성 활용

2.NoSQL 적용 사례

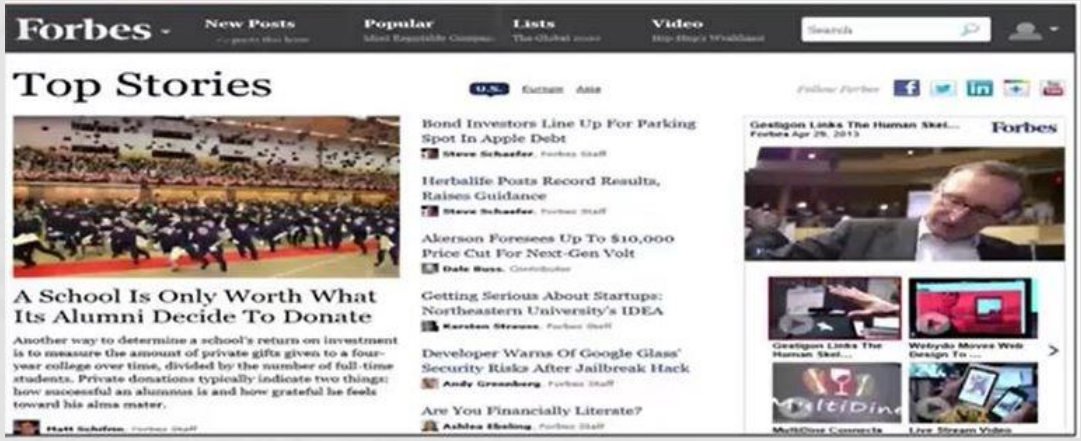
사례 - GYT

- Music Television
- 비디오/오디오 Content Management System에 적용
- MySQL을 NoSQL로 전환
- MTV의 계층적 데이터 구조에 적합한 데이터 모델 활용
- 쉬운 Query와 Index를 이용한 빠른 검색 기능 활용

2.NoSQL 적용 사례

사례 - Forbes

- Business Media Company
- 원고 자동 수집 및 발생 시스템에 적용
- Oracle DB를 NoSQL로 전환
- 전형적인 Static Data 관리에서 Dynamic Data 관리로 전환하면서 발생하는 재 설계 및 구축 비용 절감 목적으로 활용



2.NoSQL 적용 사례

사례 - Shutterfly

- 인터넷기반 사진 정보 및 개인 출판 서비스 사이트
- Oracle DB를 NoSQL로 전환(20TB)
- 100만명 고객/60억개의 이미지/초당 10,000개 트랜잭션 처리에서 발생하는 구축/관리 비용 및 성능 문제가 이슈

2.NoSQL 적용 사례

사례 - foursquare

- 위치 기반 Social Network 사이트
- RDB 기반의 시스템 확장 비용 및 관리 문제가 이슈
- GeoSpatial Index 기능 활용
- ReplicaSets & Auto Sharding System 활용

2.NoSQL 적용 사례

사례 – National Archives

- 미국 국가 기록원
- 콘텐츠 정보 관리 시스템에 적용
- SQL Server에서 NoSQL로 전환(117GB)
- 쉽고 간편한 GridFs 기능 활용

2.NoSQL 적용 사례

사례 – 국내

2.NoSQL 적용 사례

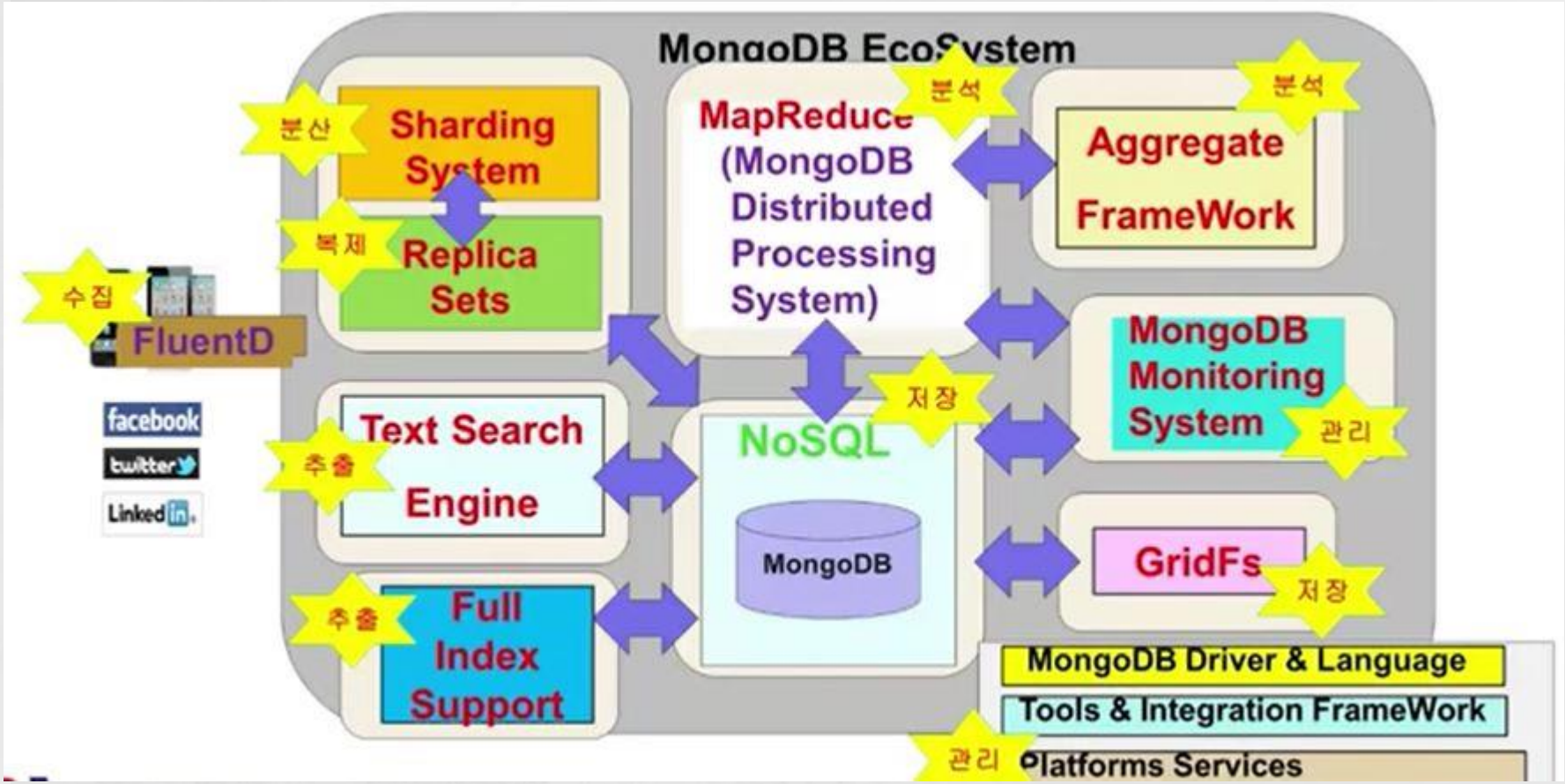
사례 – 향후

1.MongoDB 개요

1.3 MongoDB

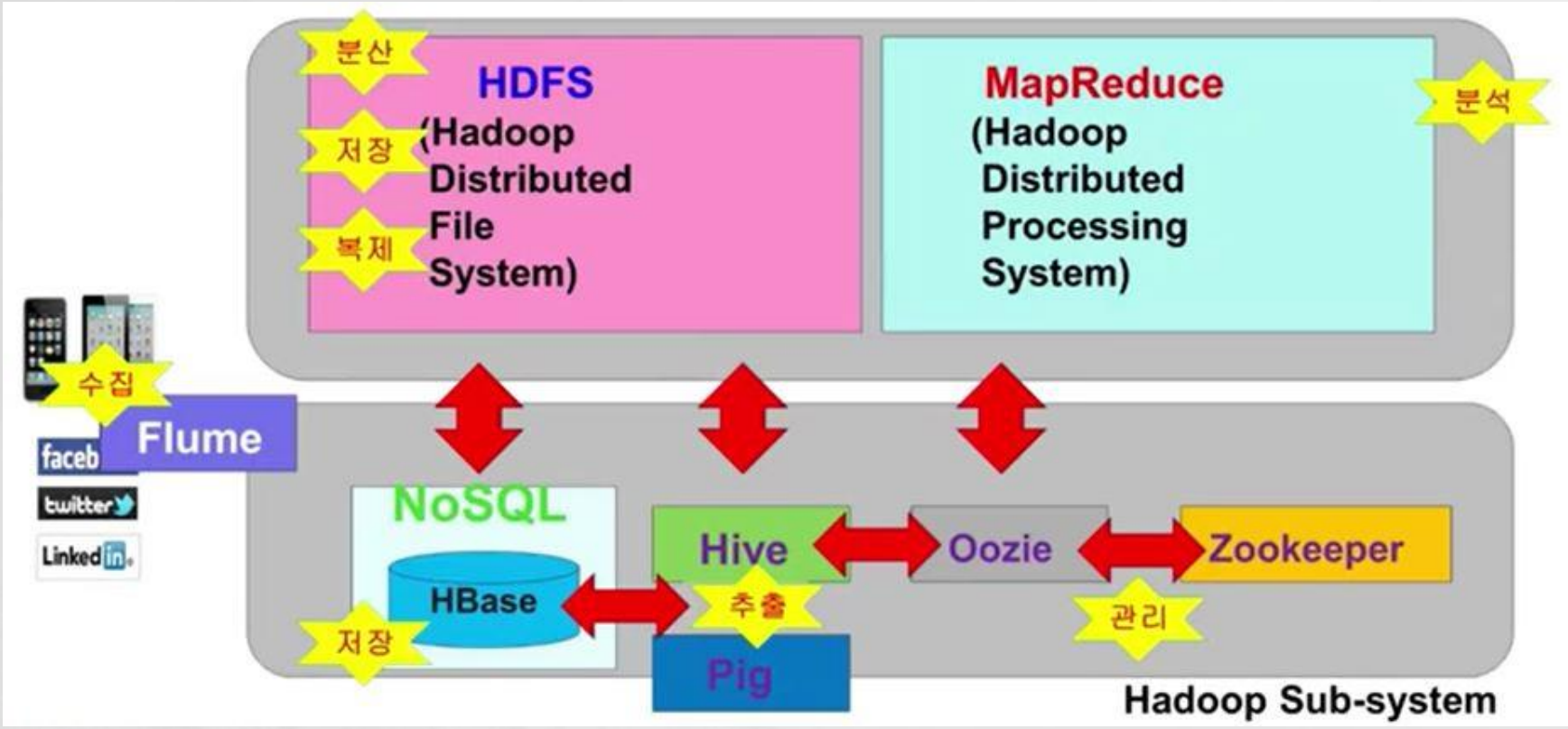
3. MongoDB & HBASE

MongoDB EcoSystem



3. MongoDB & HBASE

Hadoop EcoSystem



3. MongoDB & HBASE

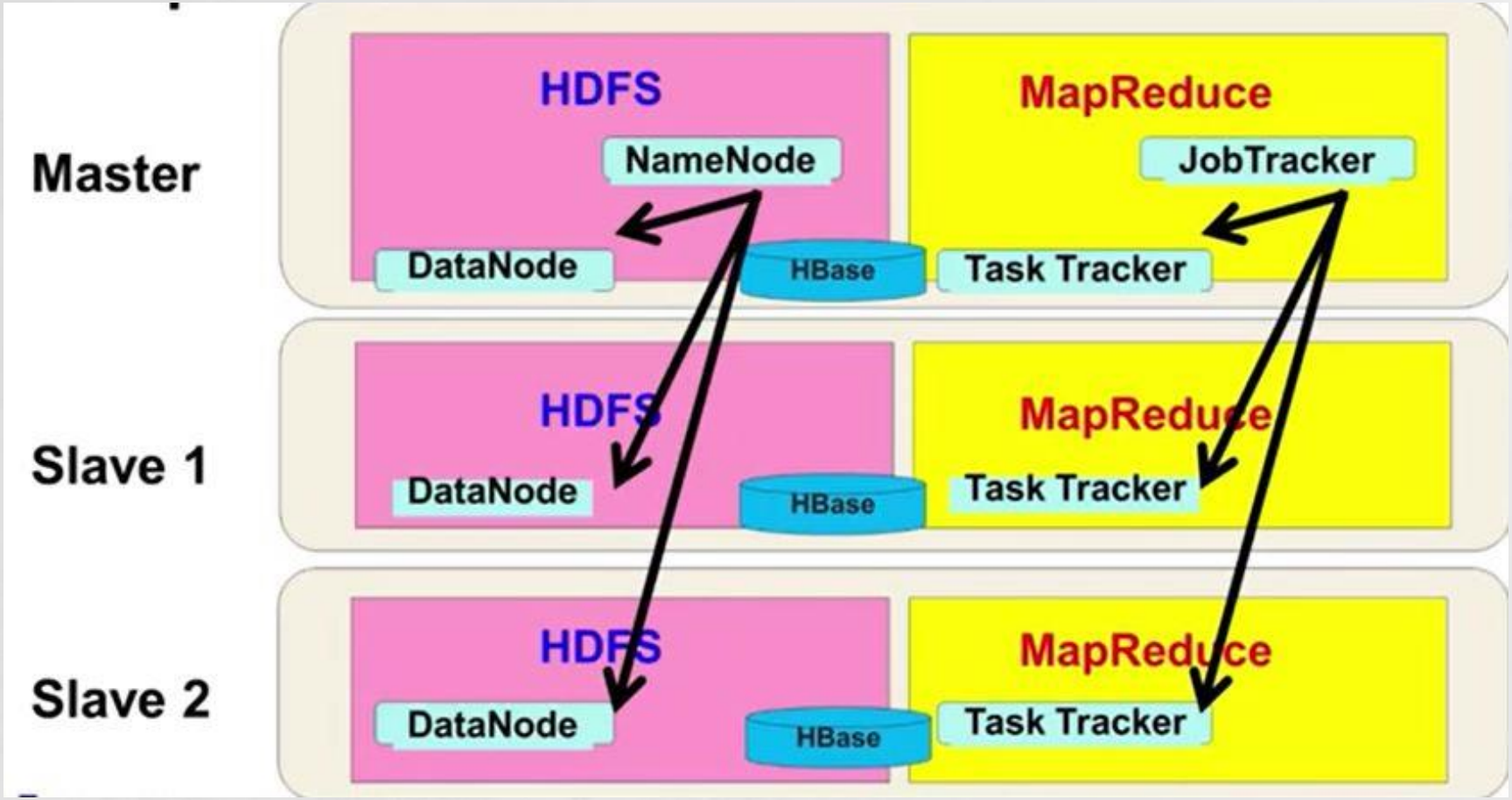
HDFS(Hadoop Distributed File System)

http://hadoop.apache.org/

- 저 비용 하드웨어를 이용한 빅데이터의 효율적인 처리를 위한 분산 파일 시스템을 의미
- 아파치 너츠(Apache Nutch) 웹 검색 엔진 프로젝트를 위한 하부 구조로 만들어졌으며 아파치 루씬(Apache Lucene)의 일 부분인 아파치 하둡(Apache Hadoop) 프로젝트에 의해 시작(Mr. Doug Cutting에 의해 개발 됨)
- 수평적 확장을 통한 시스템의 가용성을 극대화시킬 수 있으며, 이 기종 간의 하드웨어와 소프트웨어 플랫폼의 이식성이 뛰어 남
- https://projects.apache.org/project.html?hadoop

3. MongoDB & HBASE

Hadoop Architecture



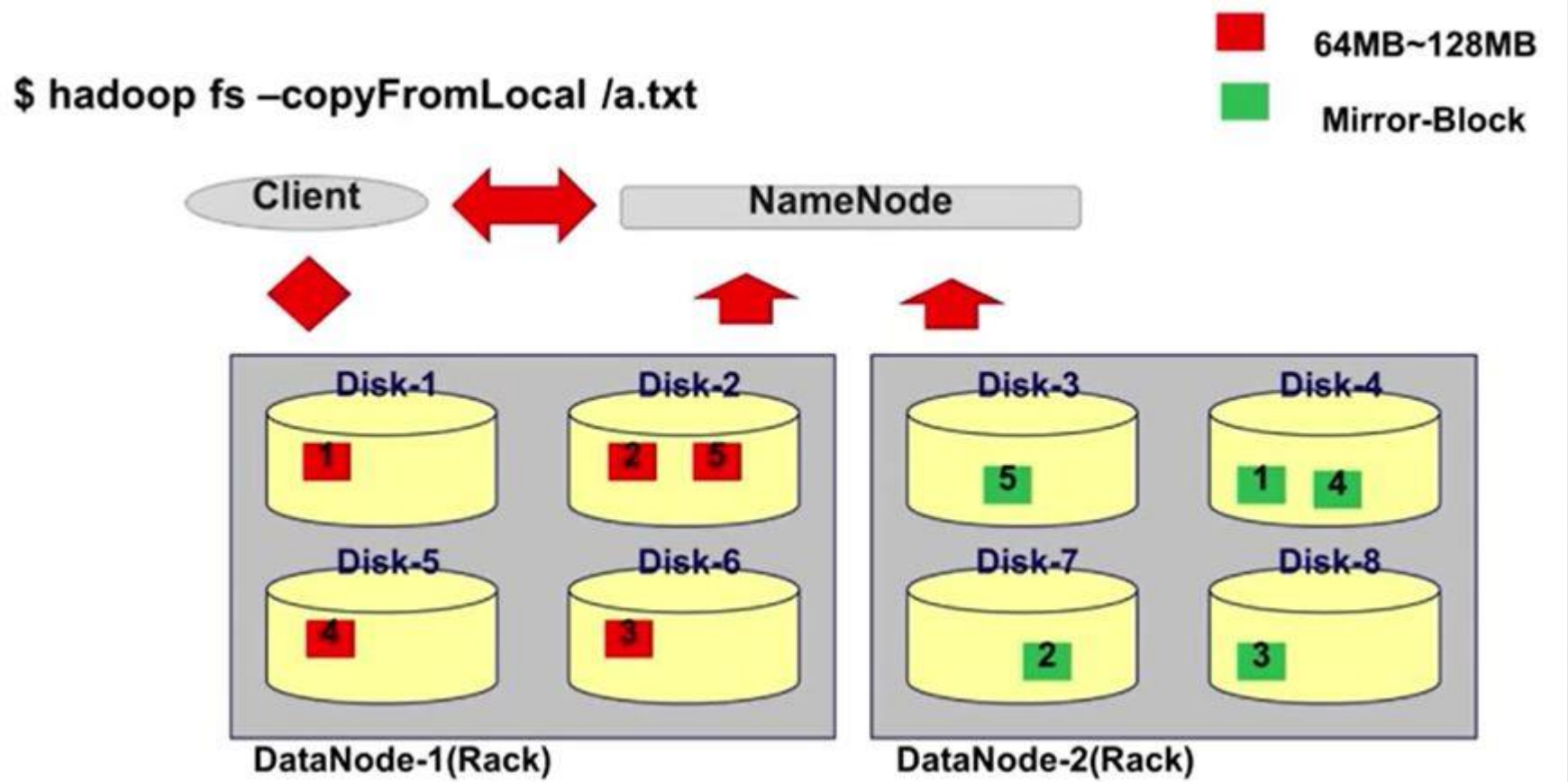
3. MongoDB & HBASE

Hadoop Shell command

```
bin/hadoop fs -ls
bin/hadoop fs -mkdir
bin/hadoop fs -copyFromLocal
bin/hadoop fs -copyToLocal
bin/hadoop fs -moveToLocal
bin/hadoop fs -rm
bin/hadoop fs -chmod
bin/hadoop fs -setrep –w 4 –r /dir1/s-dir/
```

3. MongoDB & HBASE

HDFS(Hadoop Distributed File System)



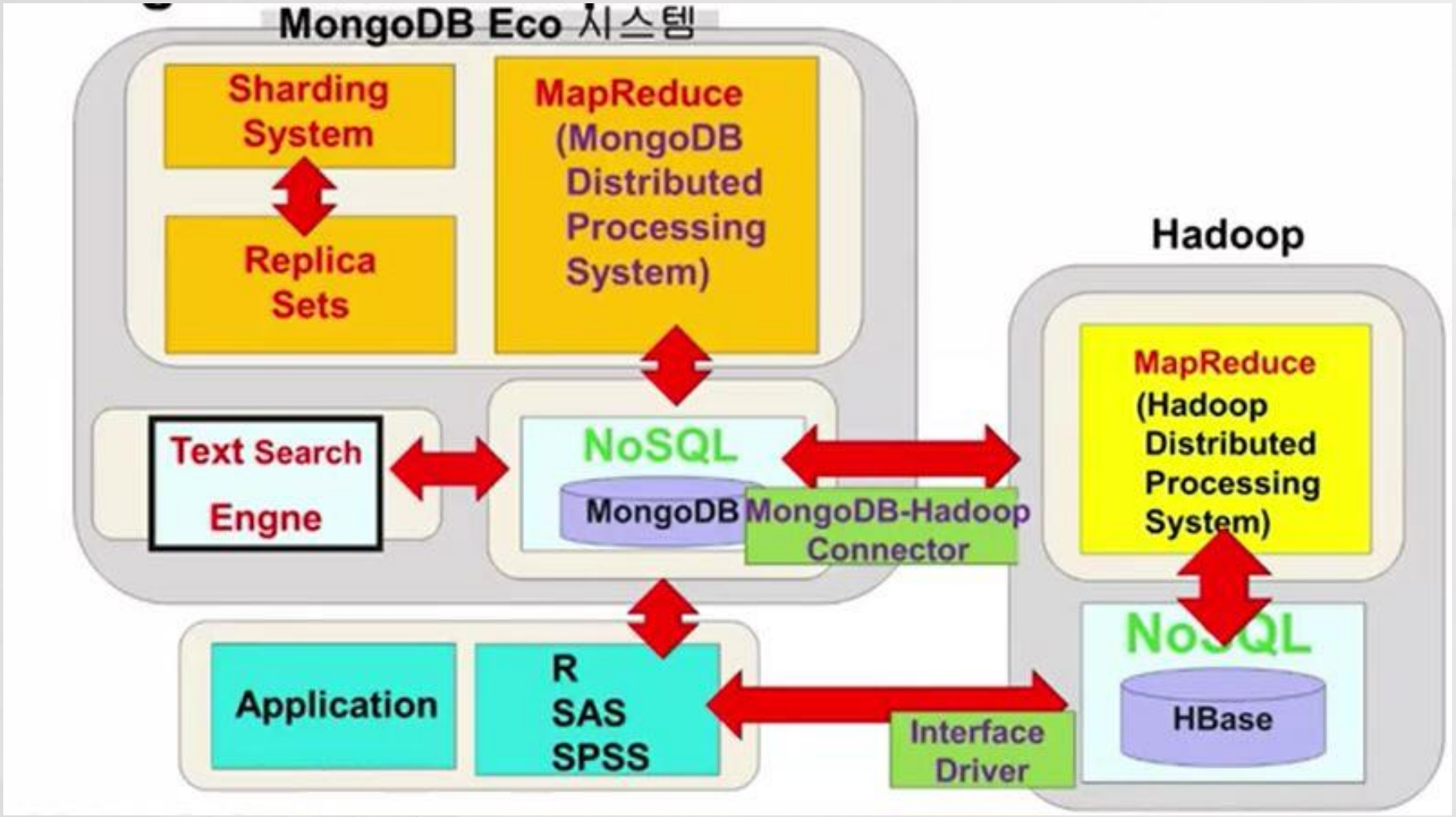
3. MongoDB & HBASE

MapReduce

- 구글에서 분산 컴퓨팅을 지원하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크
- 이 프레임워크는 페타 데이터 이상의 클러스터 환경에서 병렬처리 하기 위해 개발되었으며 MAP과 REDUECE 함수로 구성
- MAP 함수를 통해 Input 데이터(Key, Value)를 여러 개의 작은 Split-Peace로 분산 입력하고, REDUCE 함수를 통해 중복 데이터를 제거한 후 사용자가 원하는 형태로 데이터를 집계
- 구글 MapReduce를 기반으로 Hadoop MapReduce 가 설계 되었으며 Hadoop HDFS를 통해 수집 저장된 빅데이터의 효과적인 분석 처리 위한 프로그램 모델을 의미

3. MongoDB & HBASE

MongoDB와 Hadoop Architecture

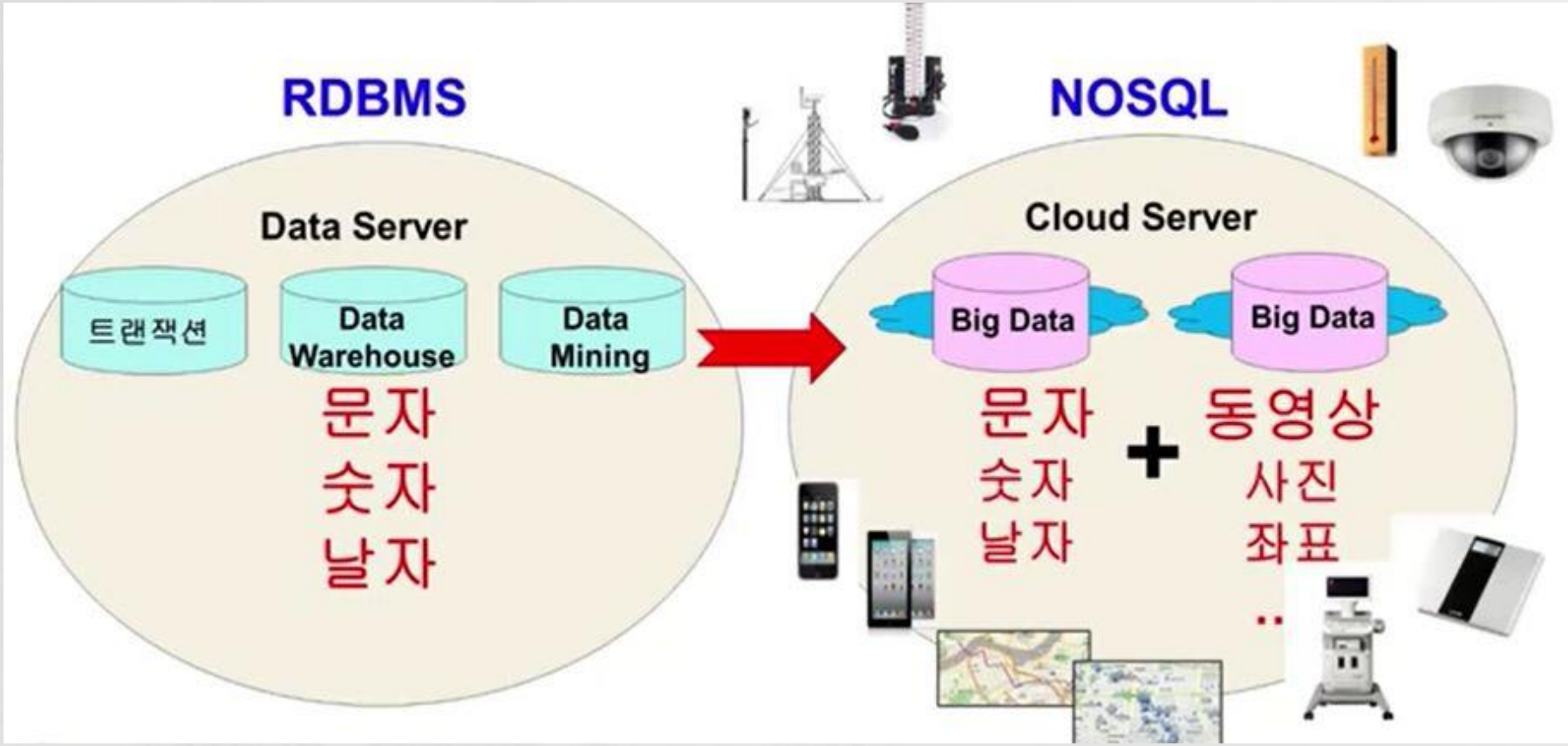


2.MongoDB 데이터처리

2.1 MongoDB

1.MongoDB

RDBMS와 NoSQL



1.MongoDB

MongoDB 란?

- Humongo라는 회사의 제품명이었으며, 현재 MongoDB inc로 회사명이 변경

- JSON Type의 데이터 저장 구조를 제공

  (Standard ECMA-2623rd Edition-1999을 근거로 하는 JavaScript 형태의 데이터 표현 방식을 근거 [European Computer Manufacturers Association])

- Sharding(분산)/Replica(복제) 기능을 제공

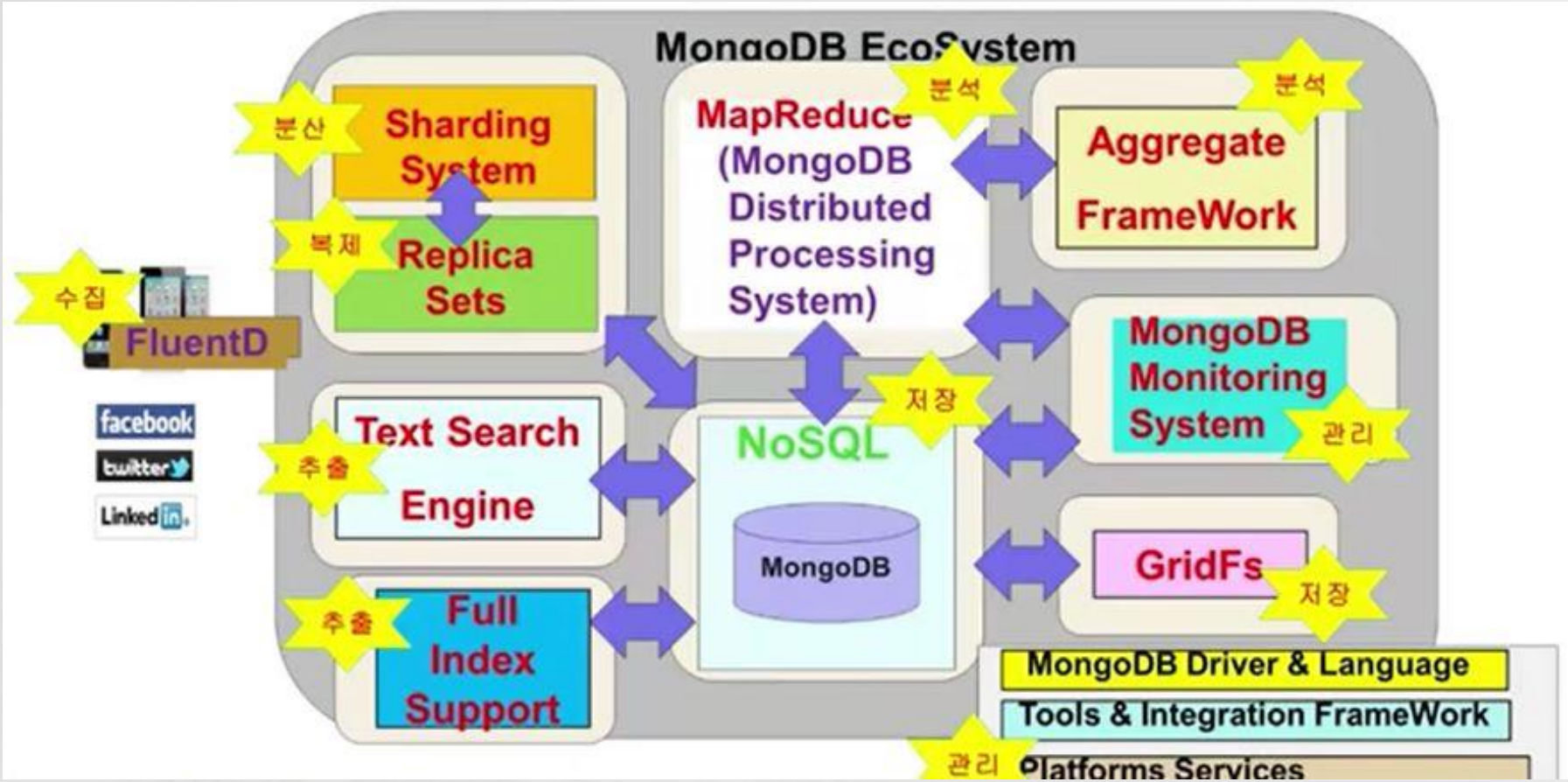
- MapReduce(분산/병렬처리) 기능을 제공

- CRUD(Create, Read, Update, Delete) 위주의 다중 트랜잭션 처리도 가능

- Memory Mapping 기술을 기반으로 Big Data 처리에 탁월한 성능을 제공

1.MongoDB

MongoDB EcoSystem



1.MongoDB

설치 환경 및 지원 드라이버

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

주요 개념

RDBMS와 차이

- MongoDB는 NoSQL(Not only SQL)로 기존 RDBMS와 다른 개념
- RDBMS 용어와 비교

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple / Row	Document
Column	Key / Field
Table Join	Embedded Documents
Primary Key	Primary Key (_id)

- MongoDB는 Join 연산을 지원하지 않음
- Embedded Documents는 Join이 아니고, 유사한 결과를 얻기 위한 패턴

Document

Document

- 기존 RDBMS는 Row 단위의 레코드 기반이라면, MongoDB는 Document 기반의 데이터베이스
- Document는 RDBMS의 Row를 의미
- MongoDB는 Key-Value기반으로 데이터를 관리하며, 입출력에는 JSON, 저장에는 BSON(Binary Json)를 사용
- NoSQL에서 지향하는 Schemaless에 부합
- JSON과 BSON 포맷을 사용하므로 배열 형태 또는 계층적(Nested) 구조의 데이터를 쉽게 다룰 수 있음

ObjectID

ObjectID

- Document에는 _id라는 ObjectID 타입의 값을 가짐
- ObjectID는 RDBMS의 Primary Key와 동일한 개념으로 유일함을 보장하는 12 byte binary data
- MongoDB는 분산 환경 지원을 위해 Sharding을 하기 때문에 서버가 아닌 클라이언트에서 생성
  - 4 Bytes – Unix Timestamp
  - 5 Bytes – Random Value(MAC or IP 주소 3바이트+Process ID 2바이트)
  - 3 Bytes – 임의 값을 시작하는 증가하는 카운터

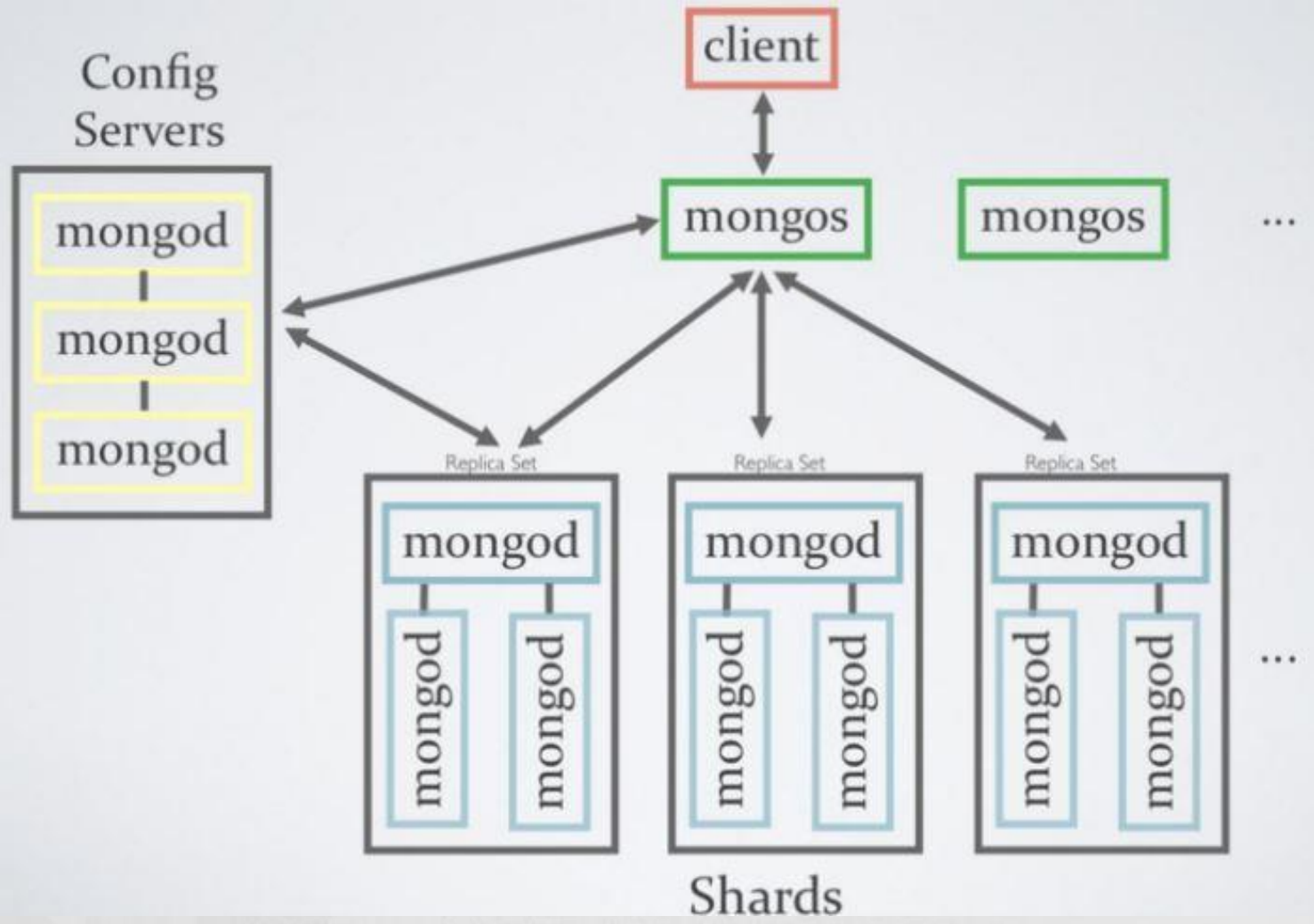
Sharding

Sharding

- MongoDB는 대용량 데이터를 분산 처리할 수 있고 Scale-out이 편리
- Sharding은 여러 장비에 걸쳐 데이터를 분할하는 과정을 의미하고 Partioning으로 표현
- Mongos라는 MongoDB의 라우터(Router)는 ObjectID로 데이터가 존재하는 Shard에만 데이터를 요청
  - 여러 개의 노드(클러스터)들을 단일 장비로 표현 가능
  - 트래픽 부하의 분산 효과
  - 병렬 작업을 통한 빠른 처리 성능

Sharding

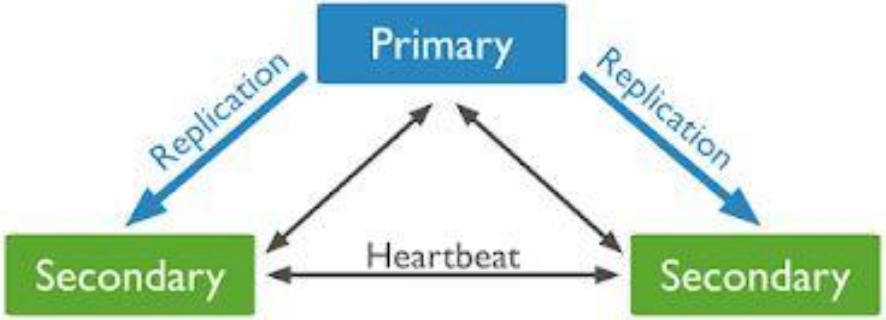
Sharding



Sharding

Sharding

- 분산 환경이므로 사용성을 유지하기 위해 Primary와 Secondary로 구분되는 ReplicaSet를 가짐
- Primary에 장애 발생하거나 사용이 불가능해지면, 과반수 투표를 통해 새로운 Primary를 선출

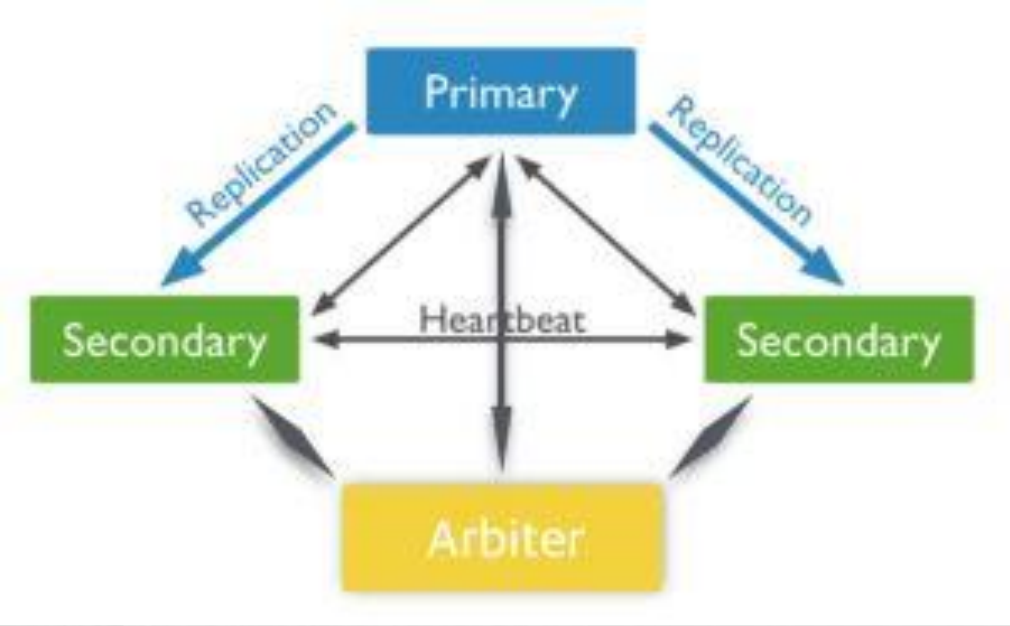


복제 셋 내 멤버의 수	복제 셋의 과반수
1	1
2	2
3	2
4	3
5	3
6	4
7	4

Sharding

Sharding

- 기본적으로 3개의 ReplicaSet을 유지하지만, 소규모 환경에서는 2개로 운영 가능
- 대신 Primary 선출 투표에만 참여하는 Arbitor라는 특수 맴버를 포함



Join 불가능

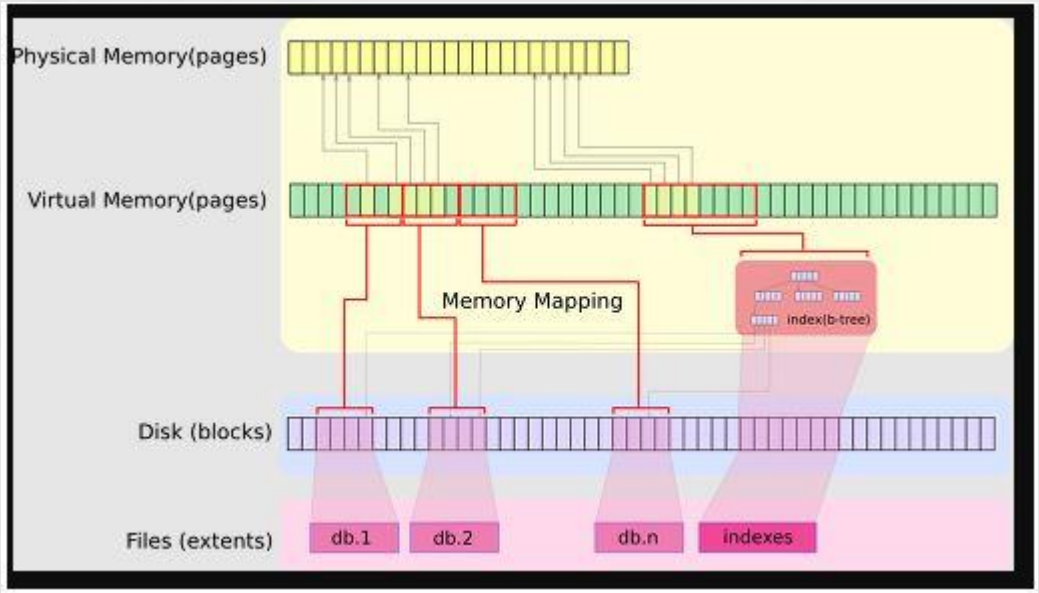
Join 불가능

- MongoDB는 Join이 불가능하므로 Join과 같은 결과를 만들 수 있음
- 참조하는 Collection이 많아지면서 성능이 느려지기 때문에 권장 안함
- Embedded sub-document를 사용

메모리 의존적

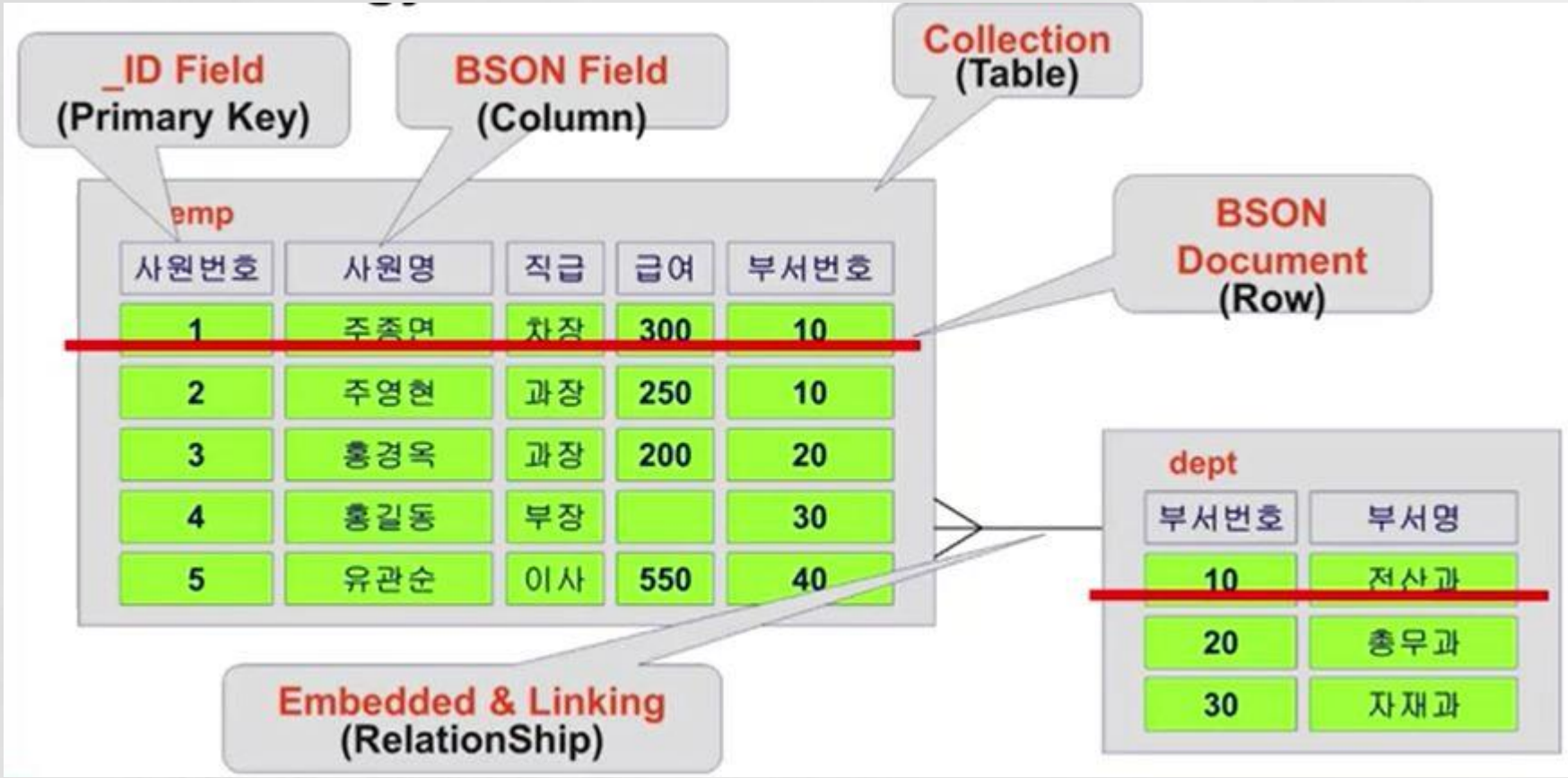
메모리 의존적

- MongoDB는 메모리 맵 형태의 파일엔진 DB이기 때문에 메모리 의존적
- 메모리의 크기가 전체 성능을 결정
- 메모리 크기를 초과하면 하드디스크의 가상메모리와 Page fault로 인한 빈번한 IO로 성능이 저하
- Document의 최대 크기는 16Mbyte
- 초과할 경우 Collection을 분리하여 관리



4.MongoDB 데이터 처리

Terminology



4.MongoDB 데이터 처리

Collection 생성

- 관계형 데이터베이스의 논리적 저장 구조인 테이블(Table) 에 해당되는 데이터 구조를 MongoDB 에서는 컬렉션(Collection) 이라고 표현
- 관계형 데이터베이스의 데이블 구조는 하나의 테이블을 생성하기 위해서는 반드시 구성 요소(컬럼명, 데이터 타입과 길이, 제약 조건)에 대한 정의가 먼저 되어야 하는데 이것을 정형화된 데이터 구조라고 표현
- Collection을 생성할 때 구성 요소(필드명 ,데이터 타입과 길이 등)가 결정되어 있지 않더라도 데이터 저장 구조를 생성

4.MongoDB 데이터 처리

Collection의 생성

- Collection에는 Capped Collection과 Non Capped Collection 2가지 종류
- Non Capped Collection은 관계형 데이터베이스의 테이블처럼 디스크 공간이 허용하는 범위 내에서 데이터를 계속적으로 저장할 수 있는 타입
- Capped Collection은 최초 제한된 크기로 생성된 공간(익스텐트) 내에서만 데이터를 저장할 수 있고 만약, 최초 공간이 모두 사용되면 다시 처음으로 돌아가서 기존 공간을 재사용하는 타입의 Collection
- 로그 데이터처럼 일정한 기간 내에서 만 저장, 관리할 필요가 있는 데이터들도 존재하기 마련인데 이런 경우 적용하면 효과적인 타입

4.MongoDB 데이터 처리

Collection 생성과 삭제

```
> db.createCollection ("emp", { capped : false, size:8192 });
{ "ok" : 1 }                          ← capped : 해당 공간이 모두 사용되면 다시 처음부터
                                                 재 사용할 수 있는 데이터 구조를 생성할 때
                                        size     : 해당 Collection의 최초 생성 크기 지정 가능
> 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
```

4.MongoDB 데이터 처리

Collection의 생성

- Collection 생성과 삭제
  - C:\MongoDB> mongo
  - > use SALES <- SALES DB 생성 및 이동(첫 번째 컬렉션 생성 시 DB 자동 생성)
  - > db.createCollecction("employees", {capped:true, size:100000}) <- 저장공간 재사용
  - > show collections
  - > db.employees.status() <- Collection의 현재 상태 및 정보 분석
  - > db.employees.renameCollection("emp") <- 해당 Collection 이름 변경
  - > db.emp.drop() <- 해당 Collection 삭제

4.MongoDB 데이터 처리

데이터의 입력 수정 삭제

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

```
> db.emp.find().sort ({eno:-1});

{ "_id" : ObjectId("4fe6852f5642c534a77fbdb1"),
      "eno" : 1103, "fname" : "SMITH", "job" : “CERK", "lname" : "STANFORD" }
{ "_id" : ObjectId("4fe685195642c534a77fbdb0"),
      "eno" : 1102, "fname" : "ADAM", "job" : "CHIEF", "lname" : "KROLL" }

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

4.MongoDB 데이터 처리

데이터의 입력 수정 삭제

- 데이터 Collection에 데이터 입력
  - > use test <- test DB를 생성 하고 이동
  - > m={ename : “smith”} <- MongoDB에서는 JSON 타입으로 데이터 표현
  - > n={empno : 1101}
  - > db.thing.save(m) <- 데이터를 저장할 때 SAVE 사용
  - > db.thing.save(n)
  - > db.thing.find() <- Collection에 저장된 데이터를 검색할 때 FIND 를 사용
  - > db.thing.insert({empno : 1102, ename : “king”})

4.MongoDB 데이터 처리

데이터의 입력 수정 삭제

- 데이터 Collection에 데이터 UPDATE
  - > db.thing.update({n:1103}, { $set : { ename : “standford”}, dept : “research”}})
  - > db.thing.update({n:1104}, { $set : { ename : “John”}, dept : “inventory”}})
  - > db.thing.update({n:1105}, { $set : { ename : “Joe”}, dept : “accounting”}})
  - > db.thing.update({n:1106}, { $set : { ename : “King”}, dept : “research”}})
  - > db.thing.update({n:1107}, { $set : { ename : “adams”}, dept : “personel”}})
  - > db.thing.save({empno : 1108, ename : “Blake”, dept : “account”})

4.MongoDB 데이터 처리

데이터의 입력 수정 삭제

- 데이터 Collection에 데이터 REMOVE
  - > db.thing.remove({m : “test”}) <- 조건을 만족하는 데이터를 검색 후 삭제
  - > db.thing.find();
  - > db.thing.remove({})
  - > db.thing.find();
  - > db.thing.drop();

4.MongoDB 데이터 처리

SQL과 Mongo Query 비교

SQL Statement	Mongo Query Statement
CREATE TABLE emp (empno Number, ename Number)	db.createCollection("emp")
INSERT INTO emp VALUES(3,5)	db.emp.insert({empno:3, ename:5})
SELECT * FROM emp	db.emp.find()
SELECT empno, ename FROM emp	db.emp.find({}, {empno:1, ename:1})
SELECT * FROM emp WHERE empno=3	db.emp.find({empno:3})
SELECT empno, ename FROM emp WHERE empno=3	db.emp.find({empno:3}, {empno:1, ename:1})
SELECT * FROM emp WHERE empno=3 ORDER BY ename	db.emp.find({empno:3}).sort({ename:1})

4.MongoDB 데이터 처리

SQL과 Mongo Query 비교

SQL Statement	NoSQL Statement
SELECT * FROM emp WHERE empno > 3	db.emp.find({empno:{$gt:3}})
SELECT * FROM emp WHERE empno != 3	db.emp.find({empno:{$ne:3}})
SELECT * FROM emp WHERE ename LIKE "%Joe%"	db.emp.find({ename:/Joe/})
SELECT * FROM emp WHERE ename LIKE "Joe%"	db.emp.find({ename:/^Joe/})
SELECT * FROM emp WHERE empno>1 AND empno <=4	db.emp.find({empno:{$gt:1,$lte:4}})
SELECT * FROM emp ORDER BY ename DESC	db.emp.find().sort({ename:-1})
SELECT * FROM emp WHERE empno=1 and ename='Joe'	db.emp.find({empno:1,ename:'Joe'})
SELECT * FROM emp WHERE empno=1 or empno=3	db.emp.find({ $or : [{ empno : 1 } , { empno : 3 }] })
SELECT * FROM emp WHERE rownum = 1	db.emp.findOne()

4.MongoDB 데이터 처리

SQL과 Mongo Query 비교

SQL Statement	NoSQL Statement
SELECT empno FROM emp o, dept d WHERE d.deptno=o.deptno AND d.deptno=10	o = db.emp.findOne({empno:1}); name = db.dept.findOne({deptno:o.deptno})
SELECT DISTINCT ename FROM emp	db.emp.distinct('ename')
SELECT COUNT(*) FROM emp	db.emp.count()
SELECT COUNT(*) FROM emp where deptno > 10	db.emp.find({deptno: {'$gt': 10}}).count()
SELECT COUNT(sal) from emp	db.emp.find({sal: {'$exists': true}}).count()
CREATE INDEX i_emp_ename ON emp(ename)	db.emp.ensureIndex({ename:1})
CREATE INDEX i_emp_no ON emp(deptno ASC, ename DESC)	db.emp.ensureIndex({deptno:1,ename:-1})
UPDATE emp SET ename='test' WHERE empno=1	db.emp.update({empno:1}, {$set:{ename:' test'}})
DELETE FROM emp WHERE deptno=10	db.emp.remove({deptno:10});

4.MongoDB 데이터 처리

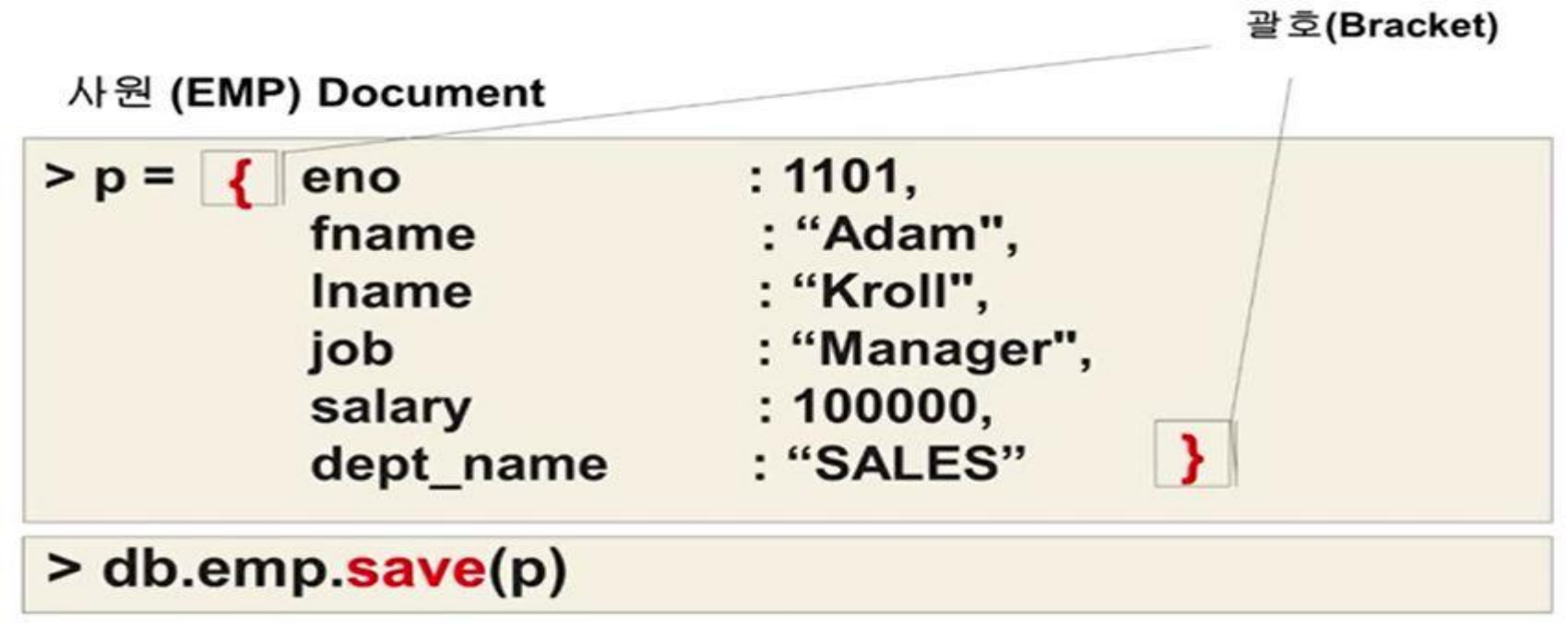
SAVE, INSERT, UPDATE 문의 차이점

- INSERT
  - Collection에 하나의 Document를 최초 저장할 때 일반적으로 사용되는 메소드
- UPDATE
  - 하나의 Document에서 특정 필드 만을 수정하는 경우 사용되는 메소드
  - 하나의 Document가 여러 개의 필드로 구성되어 있더라도 해당 필드만 수정하기 때문에 빠른 시간 내에 효율적으로 데이터를 변경
  - 빅데이터의 빠른 수정이 요구되는 경우 가장 적합한 방법
- SAVE
  - 하나의 Document에서 특정 필드 만 변경하더라도 Document 단위로 데이터를 변경하는 방법
  - Document 단위로 데이터를 변경하는 경우에는 효율적이지만 필드 단위로 변경하는 경우에는 UPDATE문을 실행하는 것이 더 효율적

4.MongoDB 데이터 처리

JSON(Java Script Object Notation) 타입과 BSON 타입

- MongoDB는 Document 형태의 데이터 저장 기술로 구현
- Document는 데이터를 저장하는 단위, { ~ } 표현, 표현방법을 JSON 타입

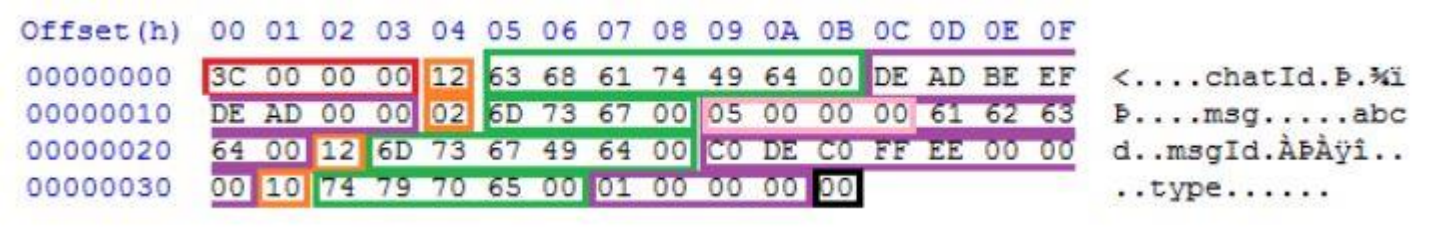


https://www.json.org/json-ko.html

4.MongoDB 데이터 처리

JSON타입과 BSON(Binary Serial Object Notation) 타입

- MongoDB에서 모든 데이터는 반드시 JSON 타입으로 표현되지만 데이터베이스 내에 저장될 때는 BSON 타입의 바이너리 (Binary) 형태의 데이터로 변환되어 저장
  - 경량의 데이터 교환 형식인 JSON(JavaScript Object Notation) 타입을 근거로 하며 사람이 읽고 쓰기에 용이하며 기계가 분석하고 생성하기에 용이
  - JavaScript Programming Language와 Standard ECMA-262 3rd Edition-1999을 근거
  - BSON 구조는 [데이터 길이] -([데이터 타입] - [키값] - [데이터])*n - [종료문자]의 형태로 구성



http://bsonspec.org/

4.MongoDB 데이터 처리

JSON타입과 BSON(Binary Serial Object Notation) 타입

```
> p= {
        "_id" : OjbectID("20130110000000001"),      <- ID 타입(Unique 값으로 사용자가 직접 정의 가능)
        "v_date" : ISODate("2017-01-21T14:22:46.7772"),   <- Data 타입
        "v_bin" : Bindata(0,"2facesf232csged2323"),       <- Binary 타입
        "v_char" : "Hong Gi Dong",                        <- 문자 데이터 타입
        "v_num" : 1038641858,                             <- 숫자 데이터 타입(32bit integer)
        "v_arr" : ["gdh11@daum.net", "dgh11@param.com" ],   <- 배열 데이터 타입
        "v_bigum" : NumberLong(125700)}                   <- 큰 숫자 데이터 타입(64bit integer)
>
> db.data_att.save(p)
> db.data_att.find()
> db.data_att.drop()
```

4.MongoDB 데이터 처리

연산자(Operator)의 종류

종류	유형	설명
비교연산자	$cmp	두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 작으면 음수, 크면 양수, 같으면 0을 리턴
	$eq	두 개의 값을 비교하여 동일하면 True, 동일하지 않으면 False를 리턴
	$gt	두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 크면 True, 작으면 False를 리턴
	$gte	두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 크거나 같으면 True, 작으면 False를 리턴
	$lt	두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 작으면 True, 크거나 같으면 False를 리턴
	$lte	두 개의 값을 비교하여 첫 번째 값이 두 번째 값보다 작거나 같으면 True, 크면 False를 리턴
	$ne	두 개의 값을 비교하여 같지 않으면 True, 같으면 False를 리턴
Boolean 연산자	$and	여러 개의 조건이 모두 True인 조건을 검색
	$not	검색 조건이 아닌 조건을 검색
	$or	여러 개의 조건 중에서 하나라도 만족되는 조건을 검색

4.MongoDB 데이터 처리

연산자(Operator)의 종류

종류	유형	설명
산술연산자	$add	두 개의 값을 합산한 결과를 리턴
	$devide	두 개의 값을 나눈 결과를 리턴
	$mod	첫 번째 값을 두 번째 값으로 나눈 후 나머지 값을 리턴
	$multiply	첫 번째 값과 두 번 째 값을 곱한 결과를 리턴
	$subtract	첫 번째 값에서 두 번째 값을 뺀 결과를 리턴
문자연산자	$strcasecmp	Long 타입의 긴 문자열 2개를 비교하여 첫 번째 문자열이 두 번째 문자열보다 크면 양수 값을 리턴 하고, 작으면 0값을 리턴
	$substr	해당문자열에서 첫 번째 정의된 숫자 만큼을 Skip하고 두 번째 정의된 숫자 만큼의 길이 데이터를 사용자에게 리턴
	$toUpper	해당문자열의 값을 소문자로 변환
	$toLower	해당 문자열의 값을 대문자로 변환

4.MongoDB 데이터 처리

빅데이터의 추출과 분석

- MongoDB 의 Aggregation Framework 함수를 이용한 빅데이터의 추출
  - 과다한 프로그래밍을 통한 비용과 시간 낭비문제를 최소화시키고 최소한의 코딩과 빠른 읽기 작업을 가능
- MongoDB 의 Map/Reduce 기능을 이용한 빅데이터의 추출
  - Map 함수와 Reduce 함수를 이용하여 JavaScript 형태로 제공되는 문법을 통해 컬렉션 내의 데이터 를 빠르게 읽고 가공 처리할 수 있는 기능
- MongoDB 와 Hadoop 의 Map-Reduce를 연동한 빅데이터의 추출
  - MongoDB로부터 데이터를 읽은 다음 Hadoop MapReduce를 이용하여 데이터를 분석, 가공 처리하는 방법

4.MongoDB 데이터 처리

Aggregation FrameWork

- 실행하면 내부적으로 MongoDB의 MapReduce를 사용하게 되며 빠른 성능을 보장, MapReduce를 이 용한 JavaScript로 생성, 외부 데이터 처리에는 제한적
- $project, $match, $group, $sort, $limit, $skip 6개의 연결 연산자로 구성되며 관계형 데이터베이스의 SELECT, WHERE, GROUP BY, ORDER BY 등과 같은 형태로 문장 작성을 할 수 있도록 만들어져서 사용하기 매우 편리

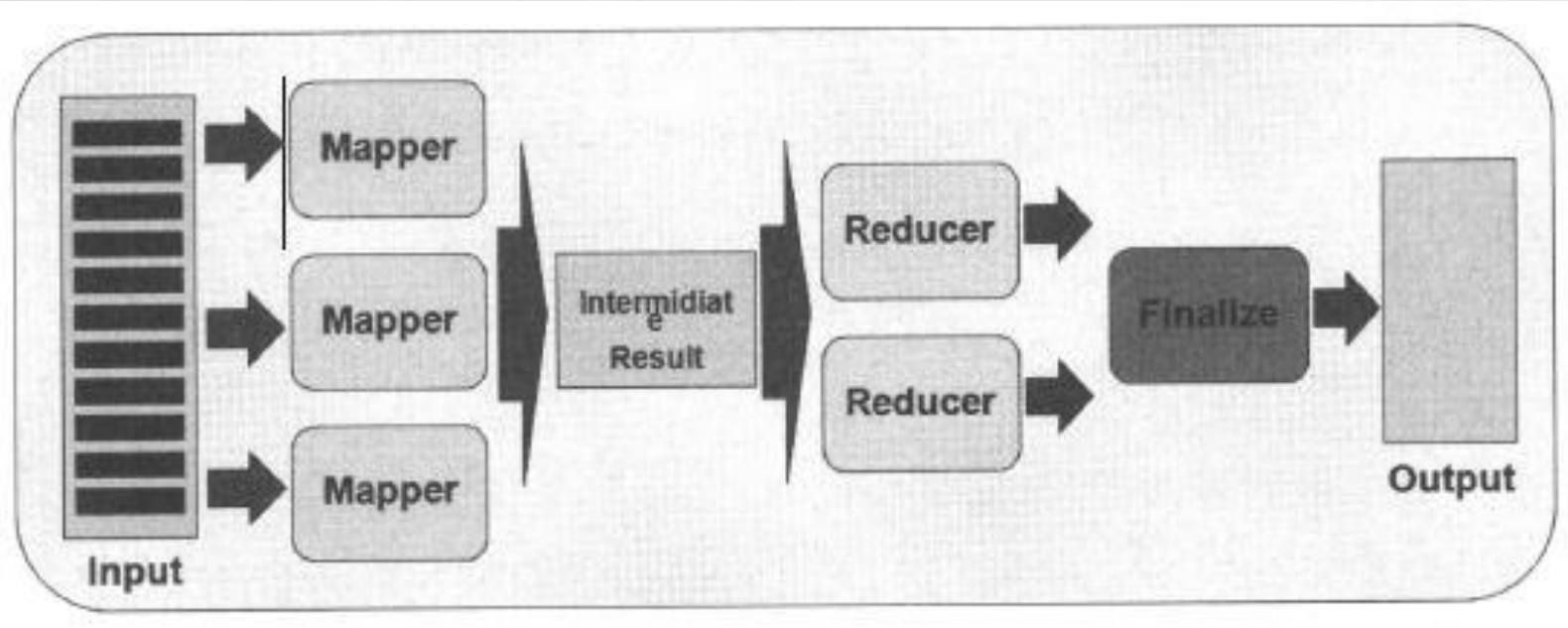
```
db.weather_history.aggregate(
{ $match :
  {co2 : { $gte : 0.00, $lte : 9.99 } },
{ $project :
  { _id : 1,  co2 : 1 } } ,
{ $group :
  { _id  :  { $year : "$_id" },
    value : { $avg : "$co2" } } } ,
{ $sort :
  { _id : 1 } } )
```

```
SELECT      year, avg(co2)
FROM        weather_history
WHERE       co2 >= 0.00
            and co2 <= 9.99
GROUP BY    year
ORDER BY    id asc;
```

4.MongoDB 데이터 처리

MapReduce 기능

- 구글에서 대용량 데이터 처리 (Batch Processing) 및 집합(Aggregation)을 위해 만들어 졌으며 비 공유 구조로 연결된 여러 개의 노드에서 병렬 처리 방식으로 대용량 데이터를 처리할 수 있음
- MAP과 REDUCE 함수 만으로 병렬 프로그래밍이 가능



4.MongoDB 데이터 처리

MapReduce 기능

- Map Function : 해당 컬렉션에서 분석 대상 필드를 emit 함수를 이용하여 정의하는 함수
- Reduce Function : 컬렉션에서 데이터를 분석 및 통계 작업을 수행하는 함수
- Finalize : 처리된 결과를 집계하는 함수
- MapReduce 함수 : Map Function과 Reduce Function에 의해 리턴 된 데이터를 이용해 결과(Output)를 출력하는 함수

```
db.collection_name.mapReduce(
                    <map>,          ← map 함수 이름
                    <reduce>,       ← reduce 함수 이름
                    {
                      <out>,        ← 실행결과를 저장할 collection명
                      <query>,      ← 검색 조건
                      <sort>,       ← sorting 조건
                      <limit>,      ← 데이터 검색조건
                      <finalize>,   ← 실행결과를 집계하게될 함수이름
                      <scope>,
                      <jsMode>,
                      <verbose> } )
```

4.MongoDB 데이터 처리

JavaScript 함수

- 비 저장형(Non-Storing JavaScript Function)
  - > function del_document() { db.test.find();
  - db.test.remove({})
  - > del_document()
  - > db.eval(del_document)
- 저장형(Storing JavaScript Function)
  - > db.system.js.save({_id "calculate_function" , value function(x , y, z)
  - { return x + y * z; }
  - })
  - > db .eval("calculate_function(10, 20, 3)")
  - > db.system.js.find()
  - " id" "calculate_function" , "value" function _cf_l_f_anonymous_
  - function(x , y, z) {
  - return x + y * z;
  - } }