

BigData & NoSQL

주 종 면 (jina6678@daum.net/010-3864-1858)



. (주)플랜정보기술 / 대표 컨설턴트

- 한국 데이터베이스 진흥원 기술 위원 및 겸임 교수
- 한국 SW 기술협회 겸임 강사
- **MongoDB Master** 공인 전문가
- **Oracle ACE** 공인 전문가
- DB 설계/튜닝/컨설팅 /MongoDB 기술서적 외 20권 출판

. MongoDB 공식 한국 사용자 그룹 운영자

- www.mongodb-korea.org
- 현재 회원 수 약 3,500 명
- 초/중급 스터디 그룹 및 심화 스터디 그룹 운영 중
- 10gen co 엔지니어 초빙 기술 컨퍼런스 개최

. 미국 MongoDB inc. 한국 공식 파트너 (www.pitmongo.co.kr)

- MongoDB 개발자 과정 및 DBA 과정 운영
- MongoDB 모니터링 및 설계 툴 판매

- 학습 목표 & 학습 방법-

1. **BigData 처리를 위한 기술적 개념 파악**
2. **NoSQL 저장 기술에 대한 이해**
3. 활용을 위한 설치 및 구현 방법 이해(난이도 : 초중급)
 - 1) 각 차시별 평균 **20분~30분** 학습
 - 2) 주요 기술에 대한 **Demonstration** 진행

* 본 강의에 사용되는 PPT 자료의 저작권은 (주)플랜정보기술에 있음을 알려 드립니다.
무단 복제 및 재사용을 금지합니다.

- 목 차 -

- 1 차시. **BigData** 개념**
- 2 차시. **NoSQL** 개념**
- 3 차시. **NoSQL** 유형**
- 4 차시. **MongoDB EcoSystem & Hadoop EcoSystem****
- 5 차시. **MongoDB & Data 처리****
- 6 차시. **Data 추출 및 분석****
- 7 차시. **트랜잭션 처리 및 인덱스****
- 8 차시. **DocumentDB 데이터 모델링****
- 9 차시. **Data 분산 처리****
- 10 차시. **Data 복제****
- 11 차시. **Hbase & 데이터 처리****
- 12 차시. **Redis & 데이터 처리****
- 13 차시. **Neo4J & 데이터 처리****
- 14 차시. **FluentD & 데이터 수집****



1 차시

BigData 개념

IT 분야 10대 키워드



2011년 10대 키워드

커머셜 클라우드
컨슈머 클라우드&
N 스크린 UX
비즈니스 플랫폼 NS
스마트 워크
상황인식 컴퓨팅
보안/프라이버시 360
마켓플레이스 에코시스템
비즈니스 분석기술
멀티플랫폼으로의 웹표준
응용 프로그램 수명 주기관리

2012년 10대 키워드

정보 보호 및 보안
클라우드 서비스
소셜 네트워크 서비스
모바일 애플리케이션
위치기반 서비스
스마트 워크
소셜 비지니스
스마트 디바이스
오픈 플랫폼
빅 데이터

2013년 10대 키워드

빅 데이터 도입&활용
신종 보안 위협
스마트 홈&가전 서비스
특허&지재권 중요도
클라우드 컴퓨팅 확산
HTML5 도입
소셜 미디어&엔터프라이즈
차세대 반도체&디스플레이
콘텐츠 서비스
신정부의 IT 정책

<주요 기관별 2013년 IT기술 전망>

기관	2013년 IT기술 키워드	
가트너	모바일 기기 전쟁	전략적 빅데이터
	모바일 앱과 HTML5	실용 분석
	개인 클라우드	인 메모리 컴퓨팅
	사물 인터넷	통합 에코시스템
	하이브리드 IT와 클라우드	기업용 앱 스토어
딜로이트	소셜 비즈니스	빅데이터
	게임화(Gamification)	지리 공간 시각화
	엔터프라이즈 모빌리티 확산	디지털 ID
	사용자 권한 부여 (User Empowerment)	균형 잡힌 혁신
	하이퍼-하이브리드 클라우드	아웃사이드-인 아키텍처
EMC	IT, '비용절감'→ '가치창출'로 전환	빅데이터 분석기법은 IT 사고를 변화
	디지털 비즈니스 모델 도출 논의 확산	'애플리케이션 공장', 새로운 모델로 부각
	경영진의 디지털 사고 능력 배양	소프트웨어 정의, As-A-Service화
	기업 사용자, 100% 모바일화	IT 프로세스 기술자가 세상을 지배
	예측분석기법, 비즈니스 업계 확산	보안 패러다임 변화

*출처:한국 정보화 진흥원 동향분석시리즈 참조

Big Data 솔루션

빅 데이터의 수집과 저장 기술

NoSQL
MongoDB
Cassandra
Hbase

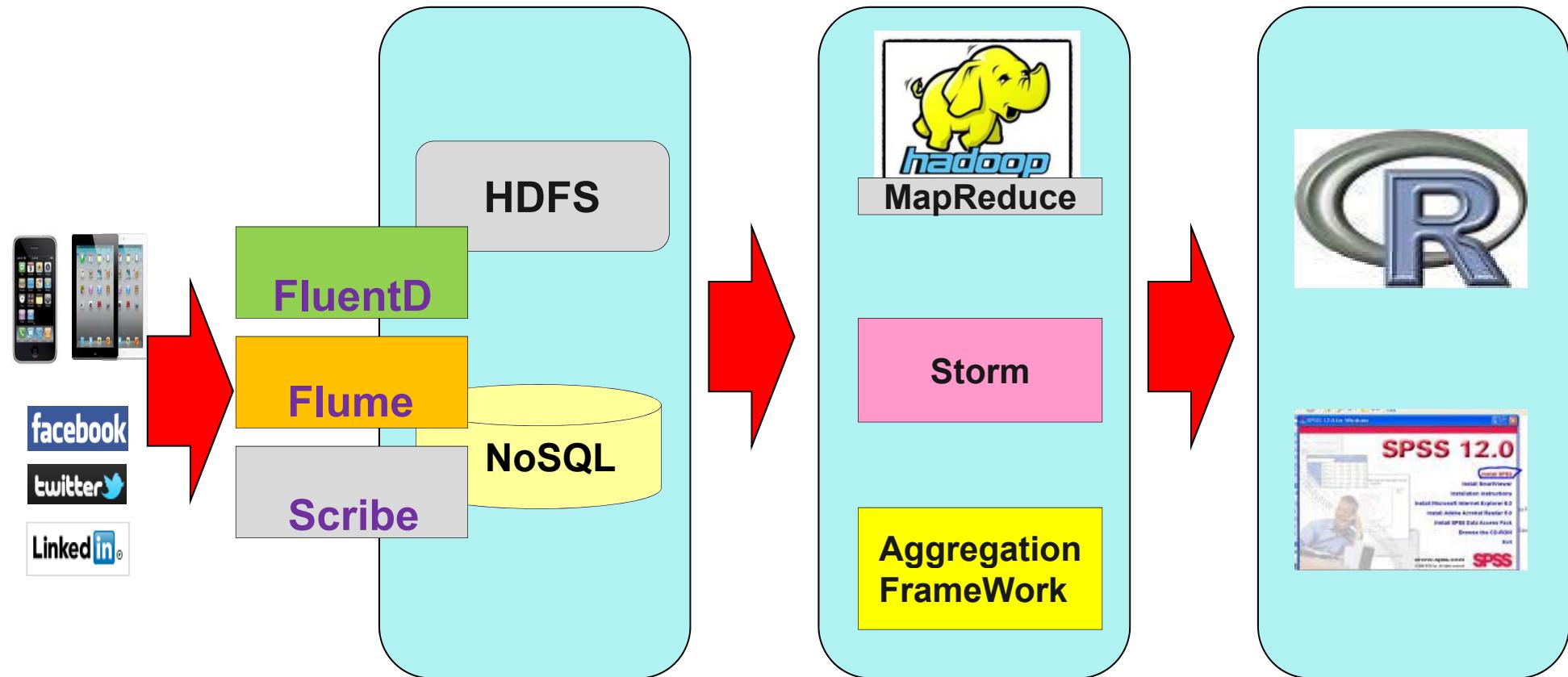
빅 데이터의 추출과 분산 기술

Hadoop
Storm
Spark
Kafka

빅 데이터의 분석 및 통계기술

R
SAS
SPSS

Big Data Processing Flow



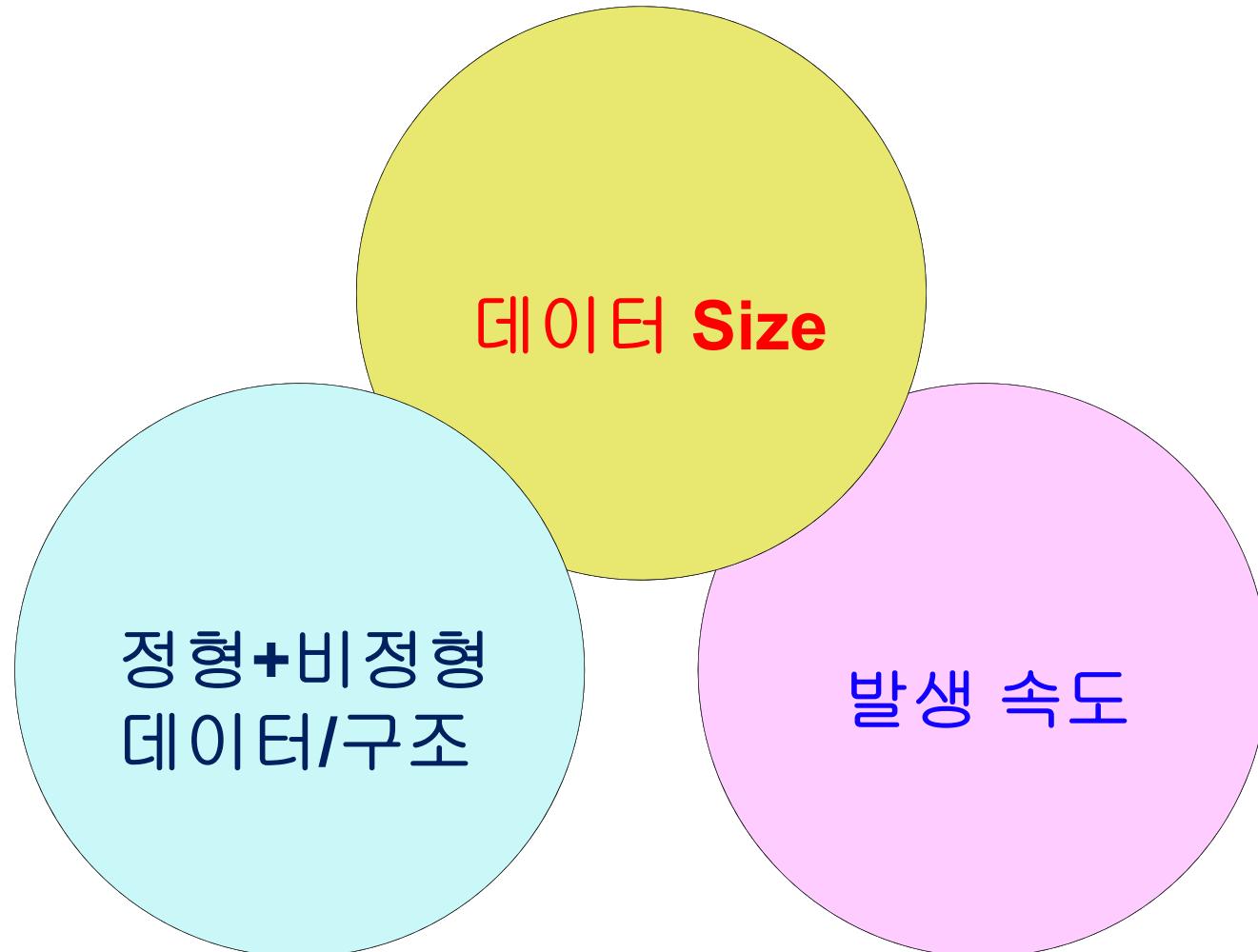
Big Data의
발생

Big Data의
수집과 저장

Big Data의
분석 처리

Big Data의
통계/시각화

Big Data 의미



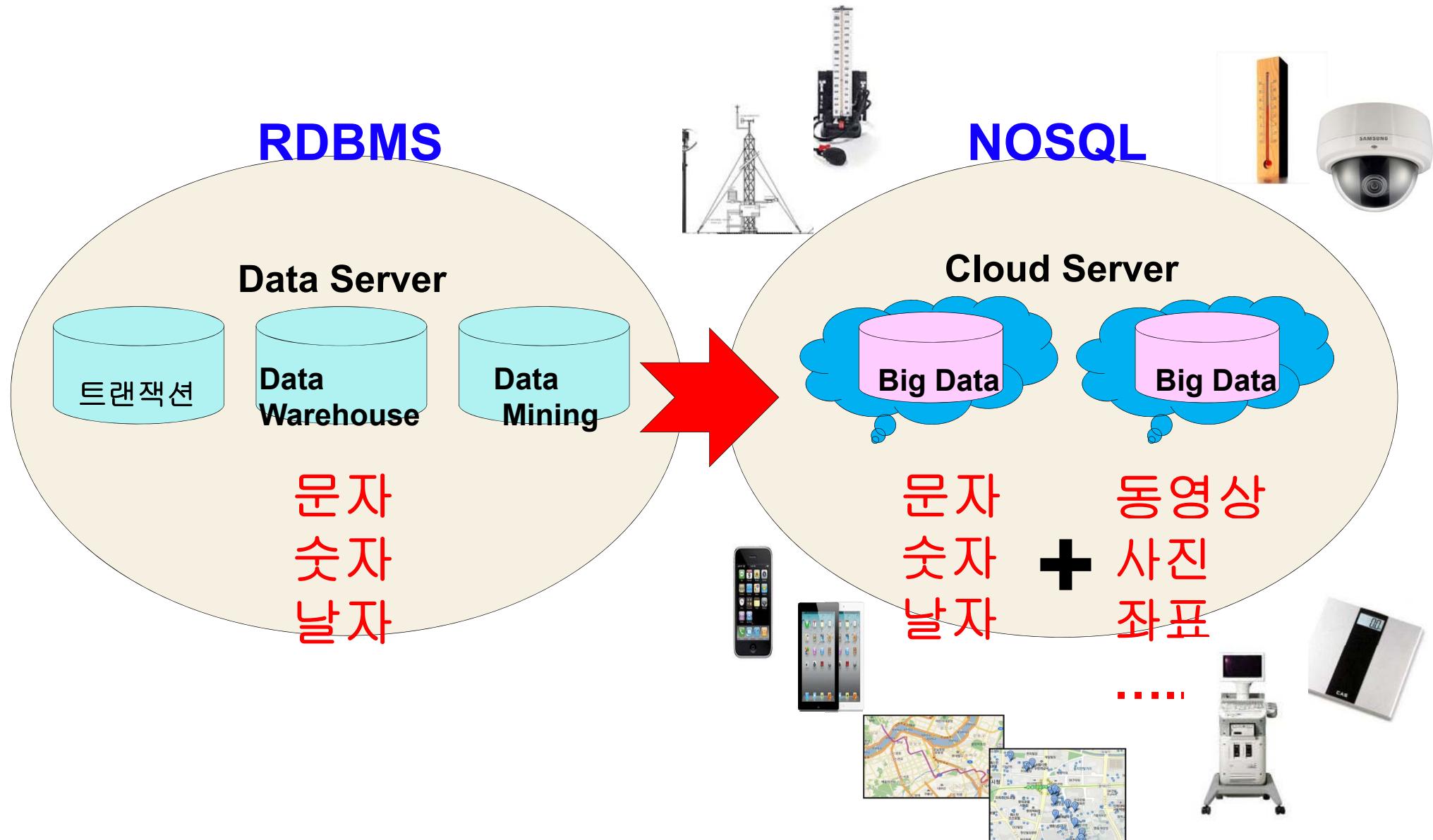
Big Data Size

Big Data

Unit	Symbol	Bytes
Kilobyte	KB	1024
Megabyte	MB	1048576
Gigabyte	GB	1073741824
Terabyte	TB	1099511627776
Petabyte	PB	1125899906842624
Exabyte	EB	1152921504606846976
Zettabyte	ZB	1180591620717411303424
Yottabyte	YB	1208925819614629174706176



비정형 데이터/구조



2 차시

NoSQL 개념

NoSQL ?

No SQL

Not Only SQL



Non-Relational
Operational Database
SQL

NoSQL의 시대적 요구



RS-232/RS-485
환경

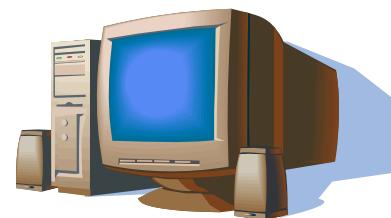
LAN 환경

인터넷
환경(1세대)

인터넷
환경(2세대)

무선 인터넷
환경(3세대)

Main (Host) 중심
Paradigm



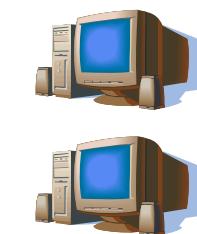
File System

Client/Server
Paradigm



RDBMS

Cloud Computing
Paradigm



NoSQL

NoSQL의 장점

1. 클라우드 컴퓨팅 환경에 적합하다.

- 1) **Open Source**이다.
- 2) 하드웨어 확장에 유연한 대처가 가능하다.
- 3) 무엇보다 **RDBMS**에 비해 저렴한 비용으로 분산 처리와 병렬 처리가 가능하다.

2. 유연한 데이터 모델이다

- 1) 비정형 데이터 구조 설계로 설계 비용 감소
- 2) 관계형 데이터베이스의 **Relationship**과 **Join** 구조를 **Linking**과 **Embedded**로 구현하여 성능이 빠르다.

3. Big Data 처리에 효과적이다

- 1) **Memory Mapping** 기능을 통해 **Read/Write**가 빠르다.
- 2) 전형적인 **OS**와 **Hardware**에 구축할 수 있다.
- 3) 기존 **RDB**와 동일하게 데이터 처리가 가능하다.

DBMS for NoSQL



<http://www.nosql-database.org/>

NoSQL 제품군

1. Key-Value Database

- 1) Amazon's Dynamo Paper
- 2) Data Model : Collection of K-V pairs
- 3) 제품유형 : Riak, Voldemort, Tokyo*

3. Document Database

- 1) Lotus Notes
- 2) Data Model : Collection of K-V collection
- 3) 제품유형 : Mongo DB, Cough DB

2. BigTable Database

- 1) Google's BigTable paper
- 2) Data Model : Column Families
- 3) 제품유형 : Hbase, Casandra, Hypertable

4. Graph Database

- 1) Euler & Graph Theory
- 2) Data Model : nodes, rels, K-V on both
- 3) 제품유형 : AllegroGraph, Sones

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

DBMS Ranking

280 systems in ranking, August 2015

Aug 2015	Rank			DBMS	Database Model	Score		
	Jul 2015	Aug 2014				Aug 2015	Jul 2015	Aug 2014
1.	1.	1.	Oracle	Relational DBMS	1453.02	-3.70	-17.83	
2.	2.	2.	MySQL	Relational DBMS	1292.03	+8.69	+10.81	
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1108.66	+5.60	-133.84	
4.	4.	↑ 5.	MongoDB 	Document store	294.65	+7.26	+57.30	
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	281.86	+9.04	+32.01	
6.	6.	6.	DB2	Relational DBMS	201.23	+3.12	-5.19	
7.	7.	7.	Microsoft Access	Relational DBMS	144.20	-0.10	+4.58	
8.	8.	↑ 10.	Cassandra 	Wide column store	113.99	+1.28	+32.09	
9.	9.	↓ 8.	SQLite	Relational DBMS	105.82	-0.05	+16.95	
10.	10.	↑ 11.	Redis 	Key-value store	98.81	+3.73	+28.01	
11.	11.	↓ 9.	SAP Adaptive Server	Relational DBMS	85.11	-2.10	-1.06	
12.	12.	12.	Solr	Search engine	81.90	+2.61	+12.87	
13.	13.	13.	Teradata	Relational DBMS	73.59	+1.28	+8.21	
14.	14.	↑ 16.	Elasticsearch	Search engine	69.64	-0.52	+30.84	
15.	15.	15.	HBase 	Wide column store	59.95	-0.97	+18.03	
16.	↑ 17.	↑ 19.	Hive 	Relational DBMS	53.87	+5.67	+23.06	
17.	↓ 16.	↓ 14.	FileMaker	Relational DBMS	51.87	+1.05	-0.20	
18.	18.	↑ 20.	Splunk 	Search engine	42.20	+0.55	+14.59	
19.	19.	↑ 23.	SAP HANA	Relational DBMS	38.25	+1.63	+15.92	
20.	20.	↓ 17.	Informix	Relational DBMS	36.80	+0.91	+3.64	

유형별 Ranking

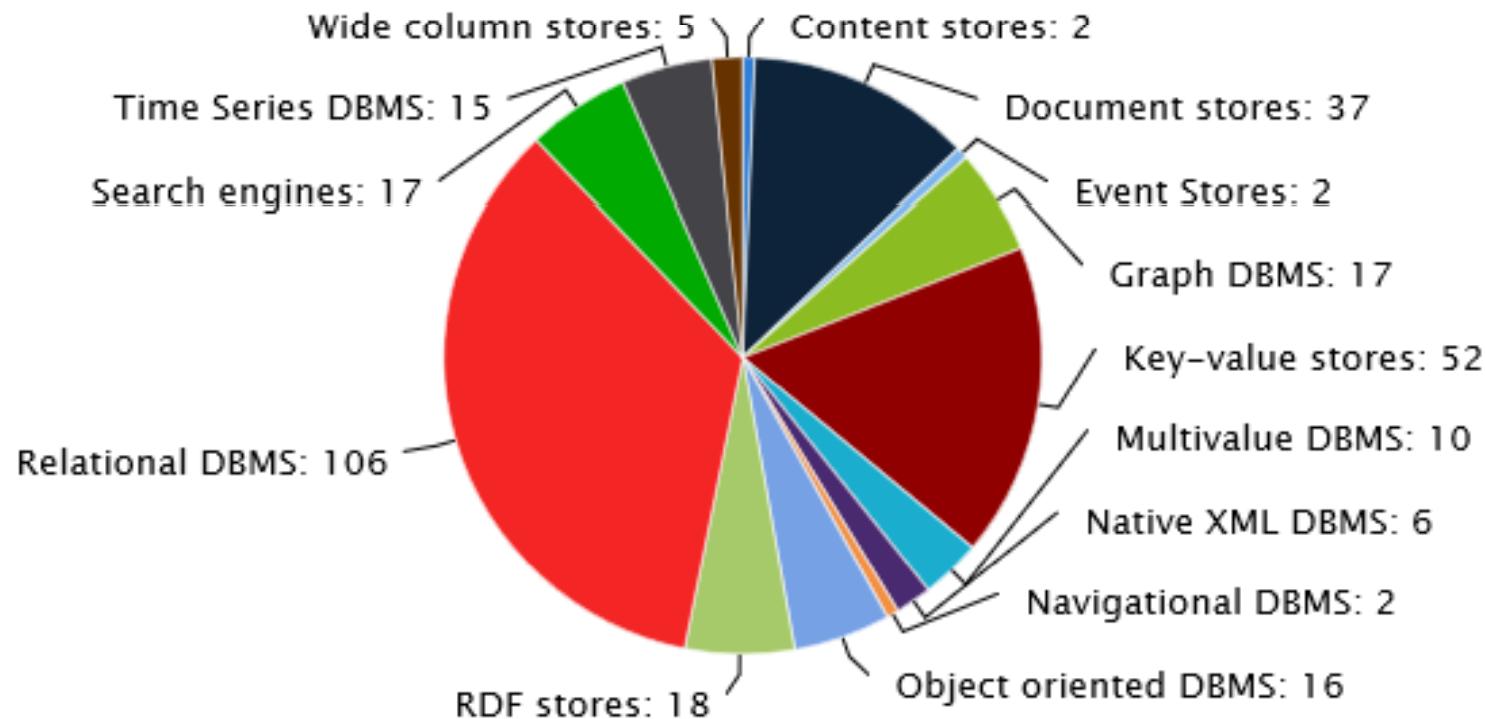
Rank			DBMS	Database Model	Score		
Aug 2015	Jul 2015	Aug 2014			Aug 2015	Jul 2015	Aug 2014
1.	1.	1.	MongoDB	Document store	294.65	+7.26	+57.30
2.	2.	2.	CouchDB	Document store	27.28	+0.35	+3.15
3.	3.	3.	Couchbase	Document store	26.16	-0.11	+8.62
4.	4.	4.	Amazon DynamoDB	Multi-model	18.45	+1.93	+8.34
5.	5.	5.	MarkLogic	Multi-model	11.71	+0.58	+3.63
6.	6.	6.	RavenDB	Document store	6.28	+0.12	+1.71
7.	↑9.	↑8.	Cloudant	Document store	4.90	+0.74	+3.05
8.	↓7.	↑9.	OrientDB	Multi-model	4.65	+0.19	+2.87
9.	↓8.	↓7.	GemFire	Document store	4.35	+0.04	+2.27
10.	10.	↑12.	RethinkDB	Document store	2.77	+0.17	+2.14

Rank			DBMS	Database Model	Score		
Aug 2015	Jul 2015	Aug 2014			Aug 2015	Jul 2015	Aug 2014
1.	1.	1.	Redis	Key-value store	98.81	+3.73	+28.01
2.	2.	2.	Memcached	Key-value store	33.38	+0.25	+2.39
3.	3.	↑4.	Amazon DynamoDB	Multi-model	18.45	+1.93	+8.34
4.	4.	↓3.	Riak	Key-value store	14.81	+1.16	+3.14
5.	5.	5.	Ehcache	Key-value store	8.07	+0.31	+1.25
6.	6.	6.	Hazelcast	Key-value store	6.27	+0.19	+1.60
7.	↑7.	↑10.	OrientDB	Multi-model	4.65	+0.19	+2.87
8.	8.	↓7.	Berkeley DB	Key-value store	4.06	+0.22	+0.96
9.	9.	9.	Oracle Coherence	Key-value store	3.71	+0.21	+1.18
10.	10.	↓8.	Amazon SimpleDB	Key-value store	3.13	+0.14	+0.13

Rank			DBMS	Database Model	Score		
Aug 2015	Jul 2015	Aug 2014			Aug 2015	Jul 2015	Aug 2014
1.	1.	1.	Cassandra	Wide column store	113.99	+1.28	+32.09
2.	2.	2.	HBase	Wide column store	59.95	-0.97	+18.03
3.	3.	3.	Accumulo	Wide column store	3.73	+0.25	+1.11
4.	4.	4.	Hypertable	Wide column store	0.71	+0.02	+0.05
5.	5.	5.	Sqrrl	Multi-model	0.41	+0.03	+0.26

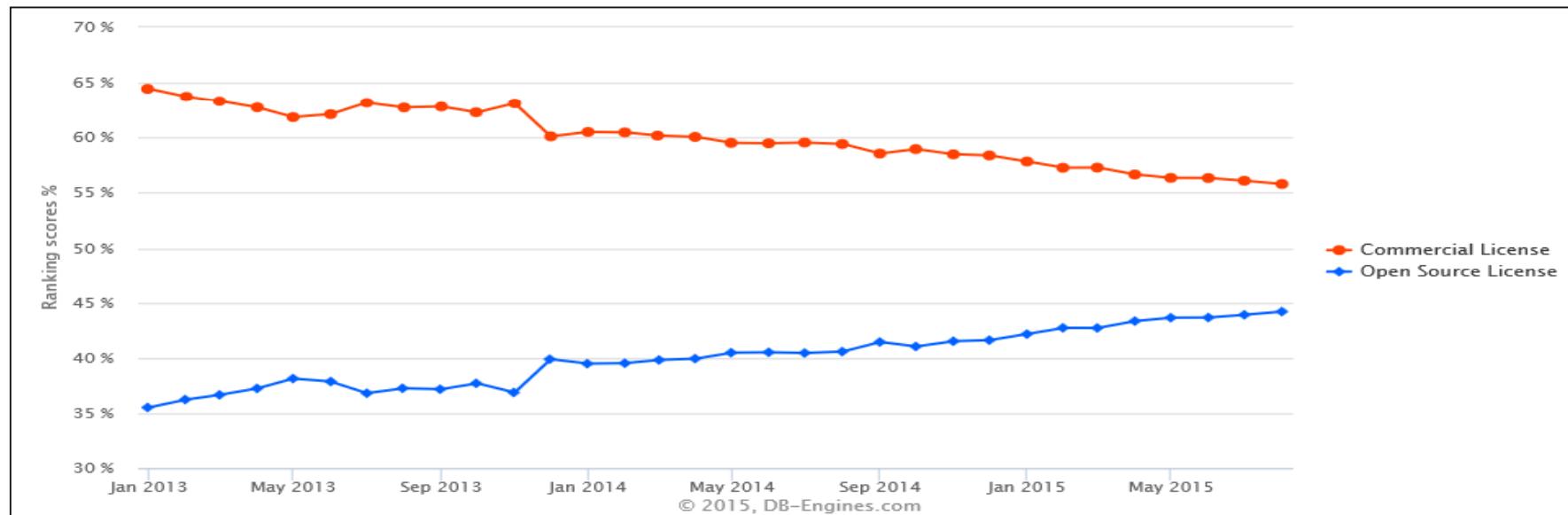
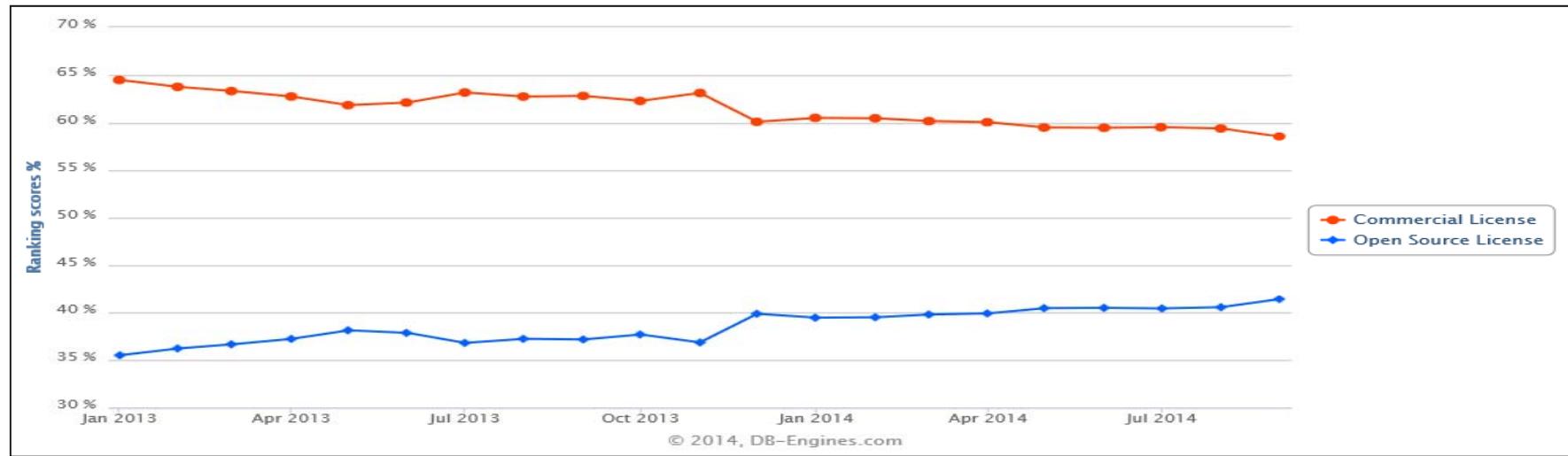
Rank			DBMS	Database Model	Score		
Aug 2015	Jul 2015	Aug 2014			Aug 2015	Jul 2015	Aug 2014
1.	1.	1.	Neo4j	Graph DBMS	33.16	+1.81	+10.25
2.	2.	↑3.	OrientDB	Multi-model	4.65	+0.19	+2.87
3.	3.	↓2.	Titan	Graph DBMS	4.23	+0.34	+2.17
4.	4.	↑6.	ArangoDB	Multi-model	1.39	+0.09	+1.13
5.	5.	5.	Giraph	Graph DBMS	0.98	-0.05	+0.58

유형별 증가 추이



© 2015, DB-Engines.com

상용 & OpenSource 별 증가 추이

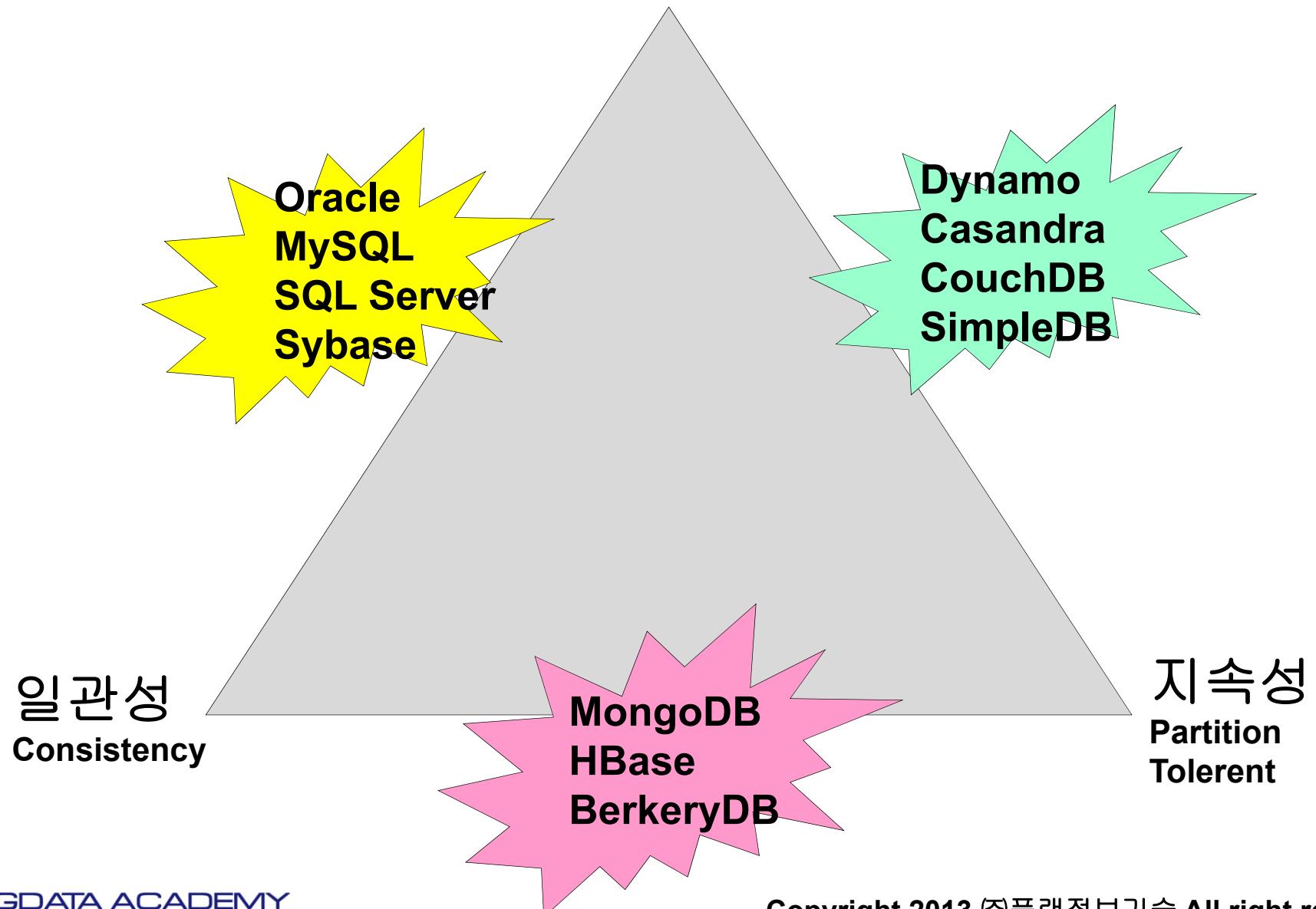


3 차시

NoSQL 유형

CAP 이론

가용성
Availability



NoSQL & RDBMS

	Dynamo	Hbase	MongoDB	RDBMS (Oracle, MySQL 등)
기능	Key/Value 데이터 저장 기술 -Amazon co.	ColumnFamily 데이터 저장기술 -Apache 재단	Document 데이터 저장 기술 -10gen co.	테이블 데이터 저장 -Oracle co, MS, IBM 등
성능	빠른 쓰기/읽기 성능 우수 -In Memory 기술 적용 -모니터링 툴 제공 -Map/Reduce 사용 가능 -Hash Key 및 Secondary 인덱스 기능 제공	빠른 쓰기/읽기 성능 우수 -In Memory 기술 적용 -Hadoop Sub-System 활용가능 -Hadoop Map/Reduce 사용 가능 -Row Key 인덱스만 제공	빠른 쓰기/읽기 성능 우수 -In Memory 기술 적용 -다양한 INDEX 기능 제공 -Map/Reduce 제공 -다양한 데이터 추출함수 제공 -모니터링 툴 기본 제공	트랜잭션 위주 데이터 처리 우수 -다양한 INDEX 제공 -다양한 함수 제공
확장성	.Scale Out 가능 -다수의 노드 확장 용이	.Scale Out 가능 -다수의 노드 확장 용이	.Scale Out 가능 -다수의 노드 확장 용이	-Scale Up 가능 -Scale Up도 가능하지만 추가 라이센스로 시간/비용증가
안정성	.복제 기능 제공 -다수의 노드로 복제 용이 -빠른 패치 제공	.복제 기능 제공 -다수의 노드로 복제 용이	.복제 기능 제공 -다수의 노드로 복제 용이 -빠른 패치 제공	.오랜 세월 시스템 안정성 확보 -다수의 노드로 복제 가능 -빠른 패치 제공
단점	.데이터 scan 처리량 1mb제한 .사용자 Interface 불편 .트랜잭션 처리가 매우 낮음 (Auto-Commit 만 지원)	.비영리 단체 Apache 재단 운영 .Hadoop System 연계 가능하지만 여러 프로젝트에서 만들어짐 .트랜잭션 처리가 매우 낮음 (Auto-Commit 만 지원) .기술지원 업체가 없음	.트랜잭션 처리가 낮음 (Auto-Commit/Rollback 가능)	-Scale Out 가능하지만 고가비용 -대용량 데이터 처리에 적절하지 않는 메모리 구조 -조인으로 인한 성능 저연유발 -분석/설계 시 시간 및 비용 증가 -유지보수 비용 증가 -고 사양의 시스템 요구 -지속성 보장이 안됨

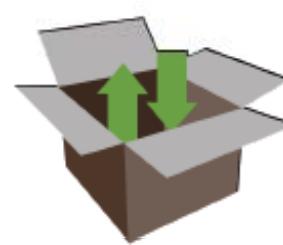
MongoDB EcoSystem Benefit



**General
Purpose**



**Document
Database**

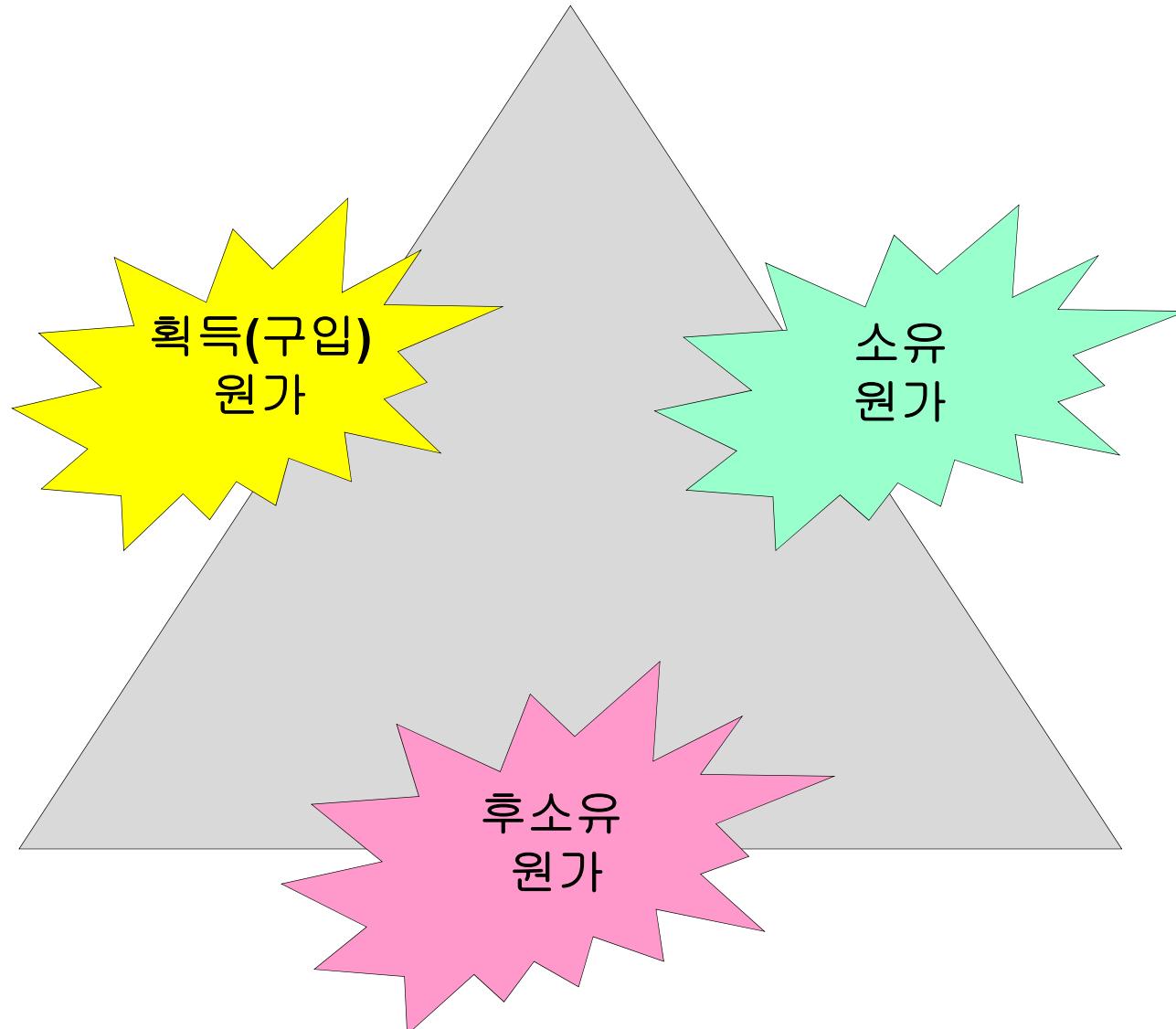


Open Source



Low Cost

TCO (Total Cost of OwnerShip)



TCO FrameWork

선행 비용(UpFront Cost)	진행 비용(Ongoing Cost)
초기 개발자 비용 (Application Coding 및 Data Store)	진행 개발자 비용 (사용자 요구에 따라 변경 등)
초기 관리자 비용 (설치, 환경설정, Shard/Replication 등)	진행 관리자 비용 (Data 유지보수, Health Check, Backup & Recovery 등)
Software Licenses	Software Maintenance & Support
Server Hardware	Server Maintenance & Support
Server Storage	Storage Maintenance & Support
	기타 개발 비용

UpFront Cost (Small Project)



Software	MongoDB Enterprise	Oracle Enterprise (Scale Out-Add on)
Server HW	3 Server (8 Core/32GB ram/per Server)	3 Server (8 Core/32GB ram/per Server)
Storage HW	3 one-TB SSDs	3 TB SAN
Initial Dev. Effort	\$120K <ul style="list-style-type: none"> • 12 man-months • \$120K annual salary 	\$240K <ul style="list-style-type: none"> • 24 man-months • \$120K annual salary
Initial Admin. Effort	\$10K <ul style="list-style-type: none"> • 1 man-month • \$120K annual salary 	\$20K <ul style="list-style-type: none"> • 2 man-months • \$120K annual salary
Software Licenses	\$0 <ul style="list-style-type: none"> • (Ongoing Cost for Subscription) 	\$423K <ul style="list-style-type: none"> • \$17.6K/core • 75% discount
Server HW	\$12K <ul style="list-style-type: none"> • 3 servers; 8/cores & 32GB RAM/server • \$4K/server 	\$12K <ul style="list-style-type: none"> • 3 servers; 8/cores & 32GB RAM/server • \$4K/server
Storage HW	\$24K <ul style="list-style-type: none"> • 2 One-TB SSDs/server (1 mirrored SSD) • \$4K/SSD 	\$125K <ul style="list-style-type: none"> • 3 TB SAN • \$125K
Total Upfront	\$166K	\$820K

OnGoing Cost(Small Project)



Software	MongoDB Enterprise	Oracle Enterprise (Scale Out-Add on)
Server HW	3 Server (8 Core/32GB ram/per Server)	3 Server (8 Core/32GB ram/per Server)
Storage HW	3 one-TB SSDs	3 TB SAN
Ongoing Dev. Effort	\$60K <ul style="list-style-type: none">• 0.5 developers• \$120K annual salary	\$120K <ul style="list-style-type: none">• 1 developer• \$120K annual salary
Ongoing Admin. Effort	\$30K <ul style="list-style-type: none">• 0.25 DBAs• \$120K annual salary	\$60K <ul style="list-style-type: none">• 0.5 DBAs• \$120K annual salary
SW Maint. & Support	\$23K <ul style="list-style-type: none">• 3 servers• \$7.5K/server/year	\$93K <ul style="list-style-type: none">• 22% of license fees
HW Maint. & Support	\$4K <ul style="list-style-type: none">• 10% of HW cost	\$14K <ul style="list-style-type: none">• 10% of HW cost
Total Ongoing per Year	\$116K per Year	
Total Ongoing over 3 Years	\$348K	
	\$860K	

Small Project : 3 Year TCO

69% savings vs.

ORACLE

\$1,680K

\$860K

\$514K

\$348K

\$166K

\$820K



TCO (Large Project)



Software	MongoDB Enterprise	Oracle Enterprise & RAC (Scale Out-Add on)
Server HW	30 Server (8 Core/32GB ram/per Server)	30 Server (8 Core/32GB ram/per Server)
Storage HW	30 one-TB SSDs	30 TB SAN

Total Upfront

\$ 750K

\$ 5,630K

Total Ongoing
over 3 Years

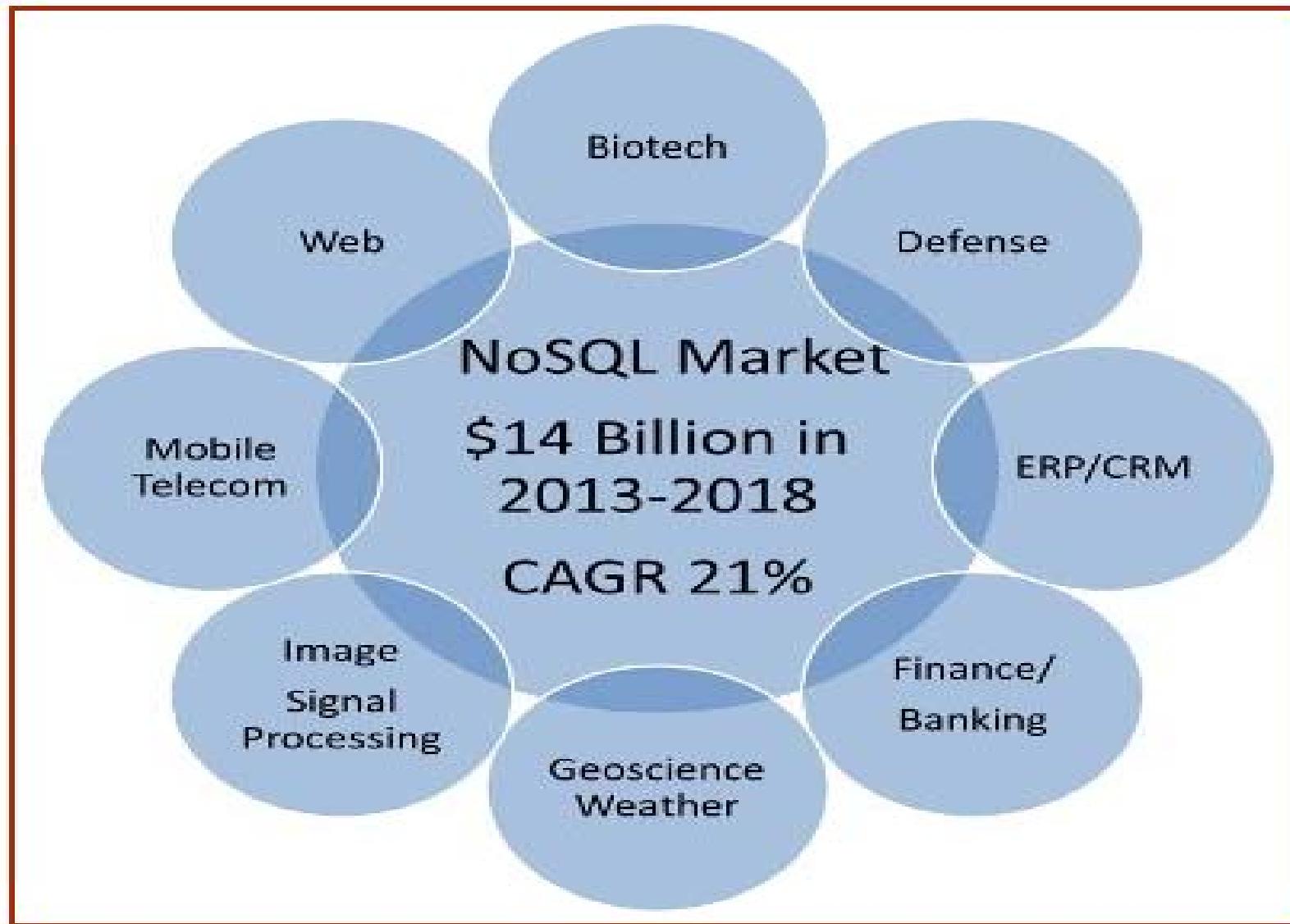
\$1,242K

\$4,597K

적용 사례(국내)



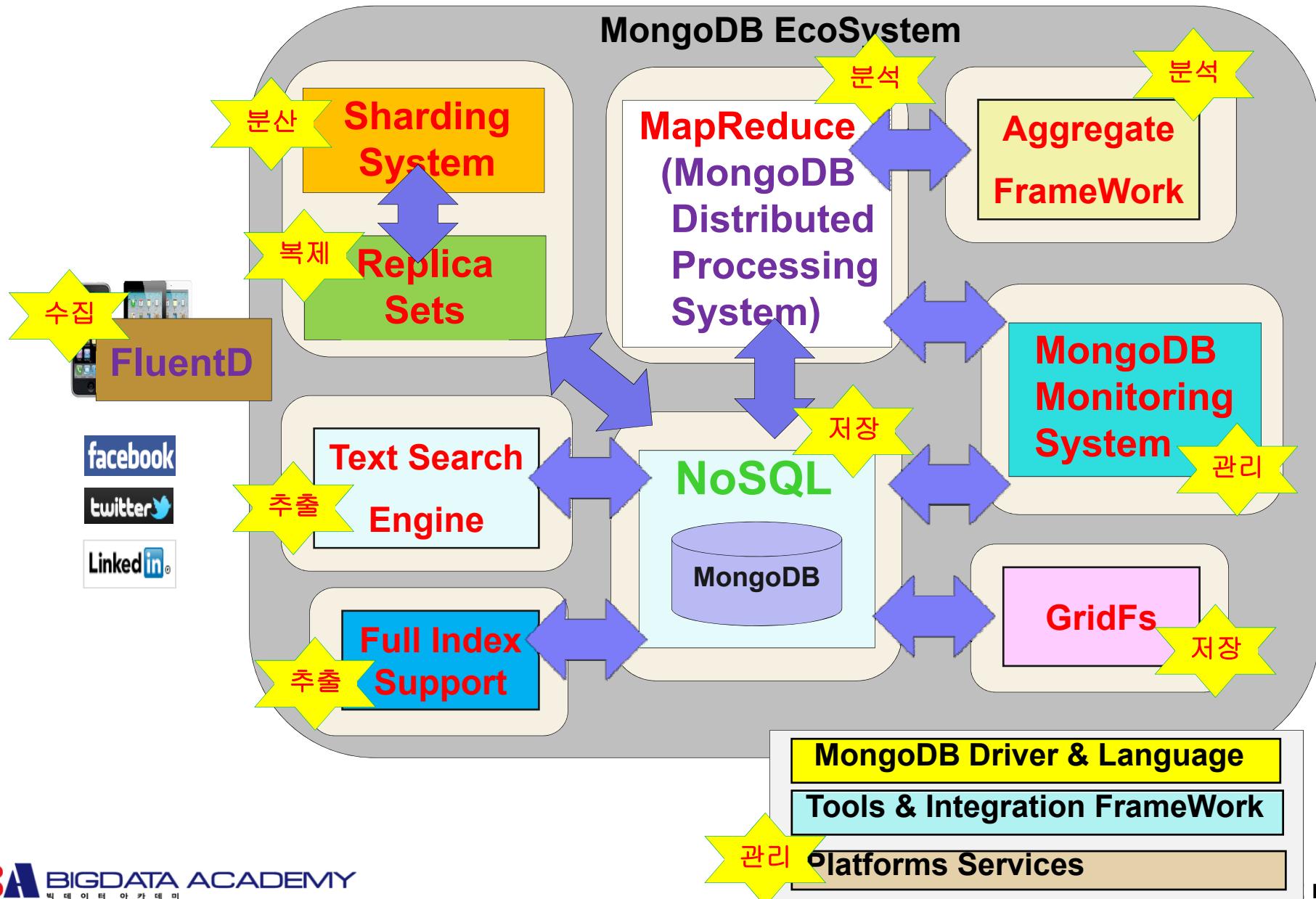
NoSQL 적용 영역



4 차시

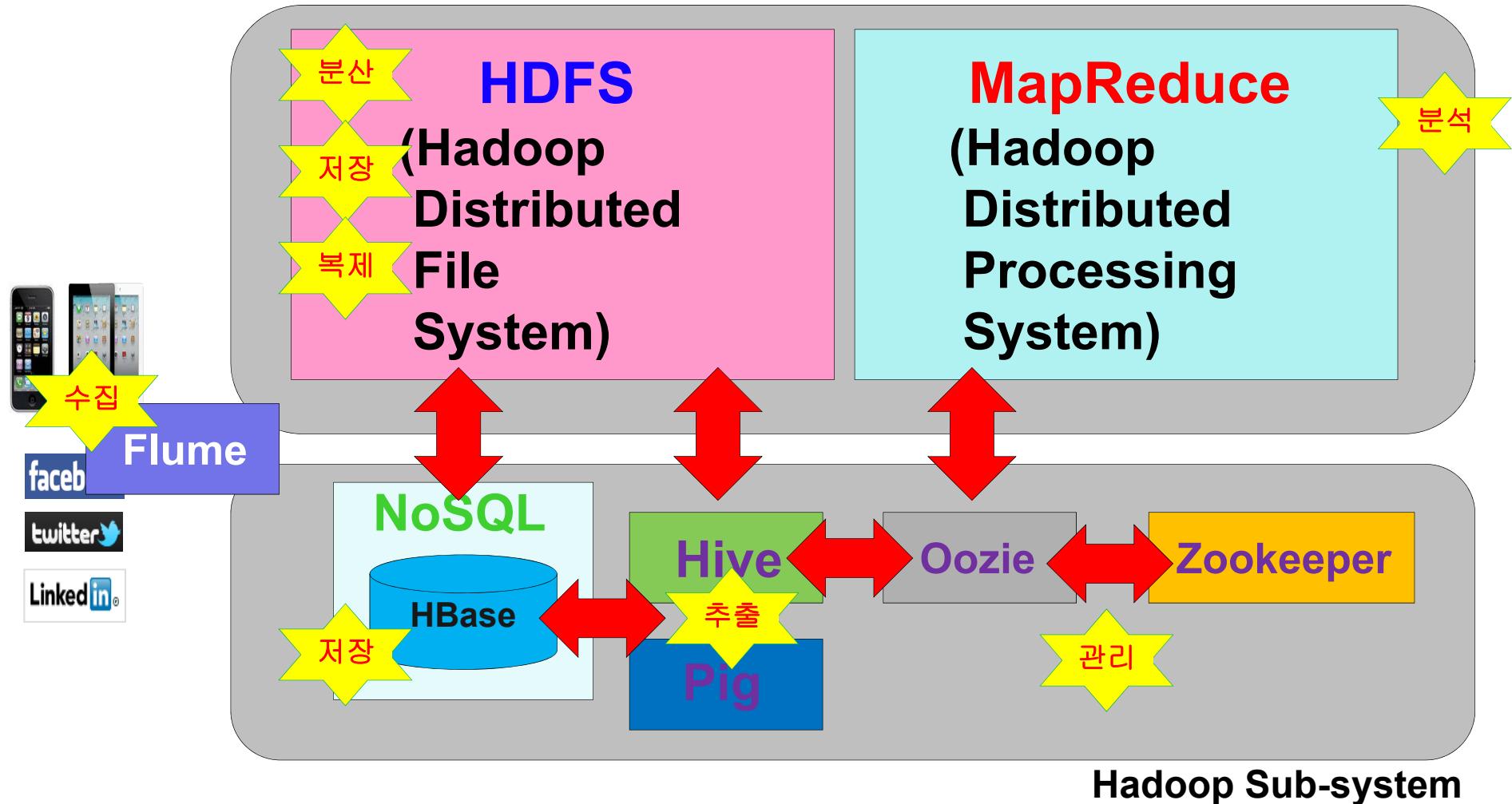
MongoDB EcoSystem & Hadoop EcoSystem

MongoDB EcoSystem



Hadoop EcoSystem

Hadoop System

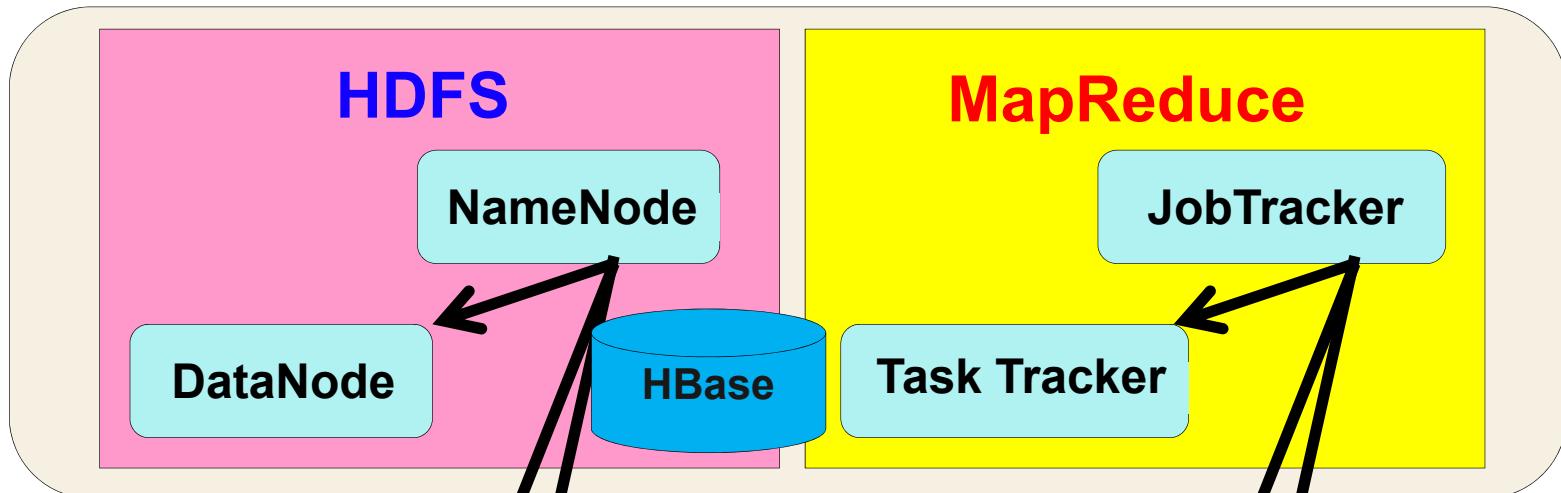


HDFS(Hadoop Distributed File System)

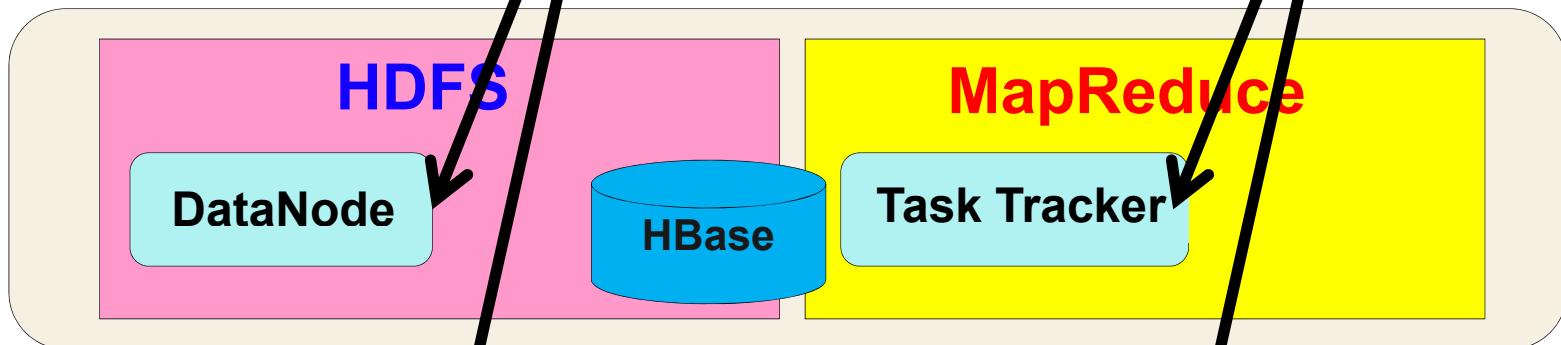
- 1) 저 비용 하드웨어를 이용한 빅데이터의 효율적인 처리를 위한 분산 파일 시스템을 의미한다.
- 2) 아파치 너치(**Apache Nutch**) 웹 검색 엔진 프로젝트를 위한 하부 구조로 만들어졌으며 아파치 루씬 (**Apache Lucene**) 프로젝트의 일 부분인 아파치 하둡(**Apache Hadoop**) 프로젝트에 의해 시작되었다. (Mr. Doug Cutting에 의해 개발됨)
<http://projects.apache.org/projects/hadoop.html>
- 3) 수평적 확장을 통한 시스템의 가용성을 극대화시킬 수 있으며 이 기종 간의 하드웨어와 소프트웨어 플랫폼의 이식성이 뛰어나다.

Hadoop Architecture

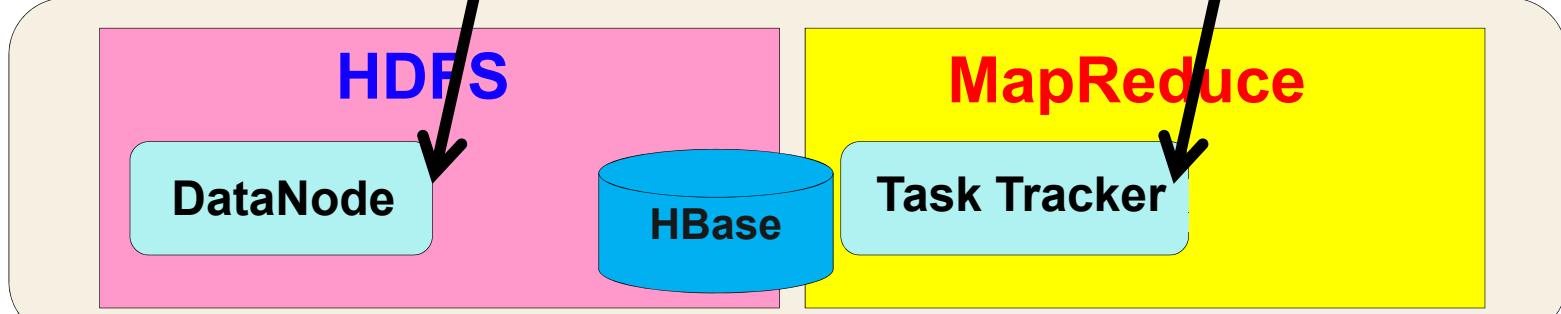
Master



Slave 1



Slave 2



HDFS 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/

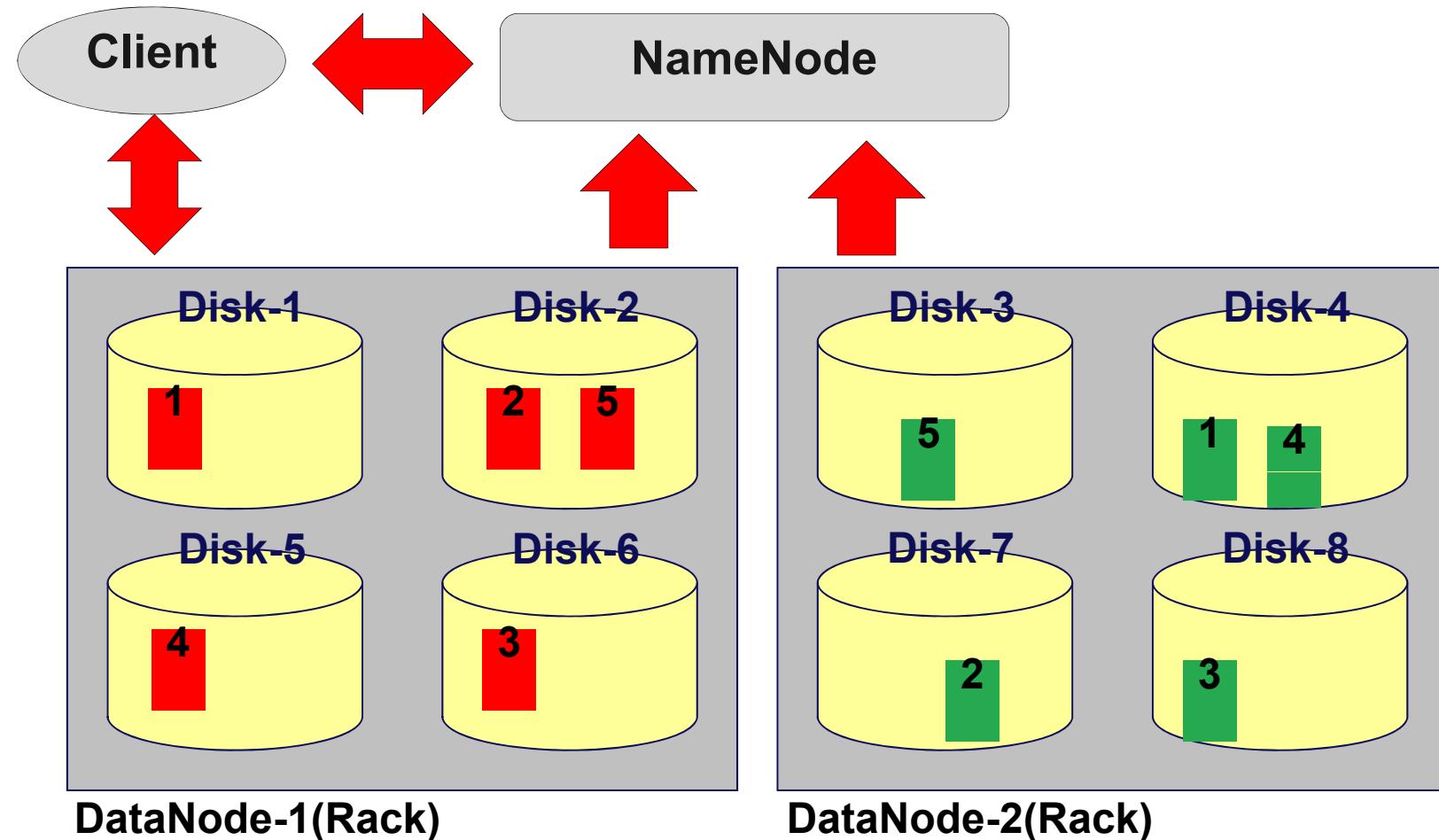
t reserved.

HDFS

* Block 단위 **KODB**
한국 데이터베이스 진흥원

```
$ hadoop fs -copyFromLocal /a.txt
```

- 64MB~128MB
- Mirror-Block



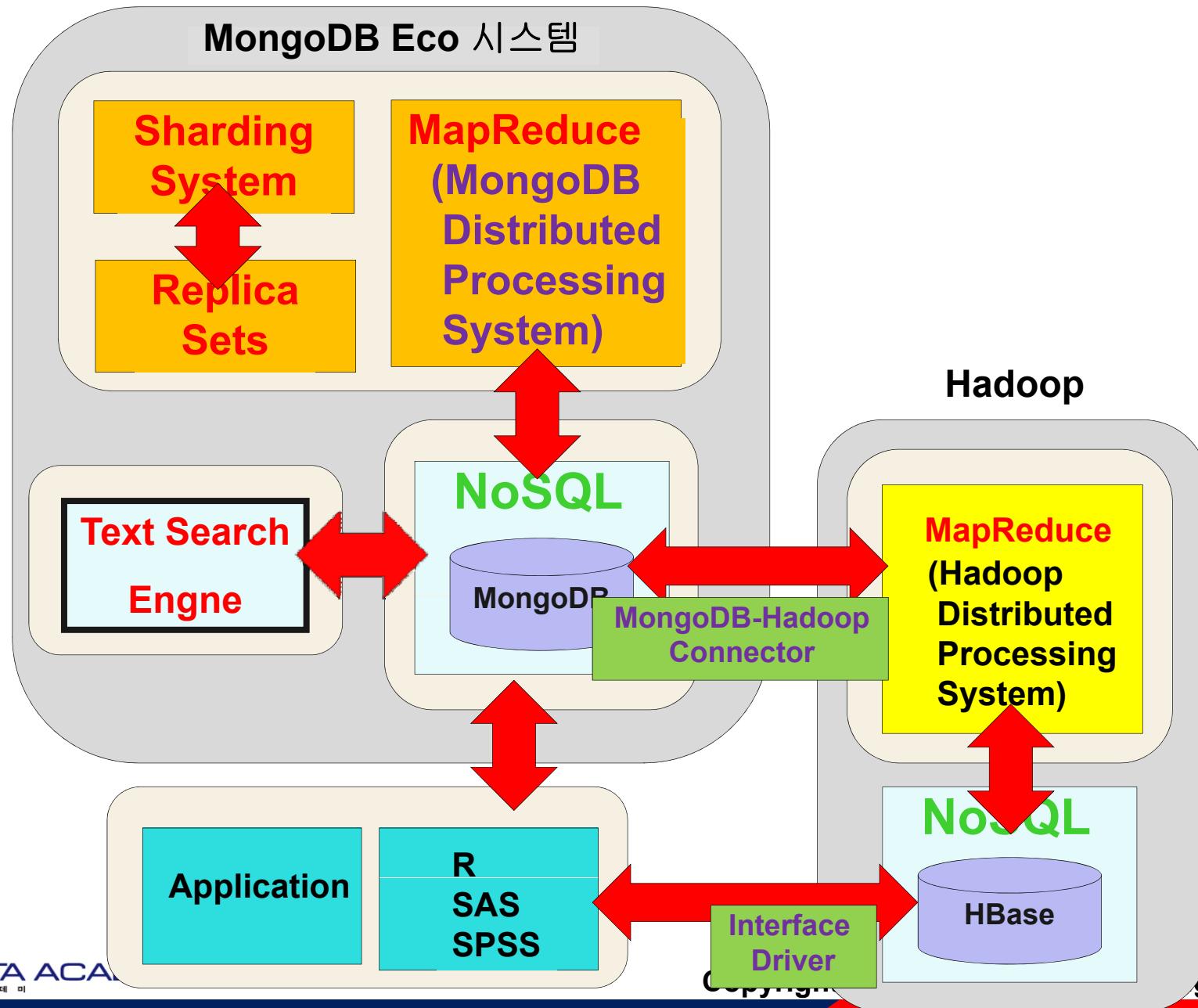
MapReduce

- 1) 구글에서 **분산 컴퓨팅을 지원하기 위한 목적으로 제작하여 2004년**
발표한 소프트웨어 프레임워크이다.
이 프레임워크는 페타 데이터 이상의 클러스터 환경에서 병렬처리
하기 위해 개발되었으며 MAP과 REDUCE 함수로 구성된다.
- 2) **MAP 함수를 통해 Input 데이터(key, value)를 여러 개의 작은**
Split-Peace로 분산 입력하고, REDUCE 함수를 통해 중복 데이터를
제거 한 후 사용자가 원하는 형태로 데이터를 집계한다,
- 3) 구글 MapReduce를 기반으로 Hadoop MapReduce가 설계되었
으며 Hadoop HDFS를 통해 수집 저장된 빅 데이터의 효과적인
분석 처리 위한 프로그래밍 모델을 의미한다.

MongoDB & Hadoop 비교

	MongoDB EcoSystem	Hadoop EcoSystem
개발사	MongoDB inc. (미국 샌프란시스코 소재)	Apache 재단 (Sub Project에 의해 개발됨)
기능	<ul style="list-style-type: none">-MapReduce (분산/병렬 Processing System)-Sharding Systema(분산 저장 System)-ReplicaSets (분산 복제 System)-Text Search Engine (검색 엔진)-Aggregate FrameWork (데이터 추출/분석 도구)-MongoDB (NoSQL)-GridFs (비정형 관리도구)-MongoDB Monitoring Service(모니터링 도구)	<ul style="list-style-type: none">-MapReduce (분산/병렬 Processing System)-HDFS (분산 파일 System)-Sharding Systema(분산 저장 System)-ReplicaSets (분산 복제 System)-HBASE(NoSQL)-Hive/Pig(데이터 추출/분석 도구)-Zookeeper (분산 Cordinator)
확장성	<ul style="list-style-type: none">-Scale Out 가능-다수의 노드 확장 용이 (추가 라이선스가 요구되지 않음)	<ul style="list-style-type: none">-Scale Out 가능-다수의 노드 확장 용이 (추가 라이선스가 요구되지 않음)
안정성	<ul style="list-style-type: none">-Fail 발생시 복제 기능 제공-다수의 노드로 복제 용이 (추가 라이선스가 요구되지 않음)	<ul style="list-style-type: none">-Fail 발생시 복제 기능 제공-다수의 노드로 복제 용이 (추가 라이선스가 요구되지 않음)
주요 특징	<ul style="list-style-type: none">-Document 데이터 저장 구조 제공(Data Set 타입에 유리)-사용자 Interface 매우 좋음(MongoDB)-트랜잭션 처리 가능-일반 기업인 MongoDB inc. 개발-모든 Sub-System은 MongoDB inc에 의해 개발되기 때문에 버전관리가 용이하며 연동 시 버그 발생율이 낮고 시스템 안정성이 높음-안정성 확보를 위해 MongoDB inc 를 통해 기술 지원이 가능	<ul style="list-style-type: none">-ColumnFamily 데이터 저장 구조 제공(Key Value 타입에 유리)-사용자 Interface 매우 나쁨(HBase)-트랜잭션 처리가 매우 낮음-비영리 단체 Apache 재단에서 개발됨-각 Sub-System은 여러개 프로젝트에서 개발되기 때문에 버전 관리가 용이하지 않으며 연동 시 버그 발생율이 높음-Apache 재단에서 직접 기술 지원이 안됨

MongoDB & Hadoop & R 연동 Architecture



5 차시

MongoDB & 데이터 처리

MongoDB 주요 특징

1) **Humongos**라는 회사의 회사명이 변경되었다.

2) **JSON Type**의 데이터

(Standard ECMA-262 3rd Edition-1999을 근거로 하는 **JavaScript** 형태의 데이터 표현 방식을 근거로 한다. [European Computer Manufacturers Association])

3) **Sharding(분산)/Replica(복제)** 기능을 제공한다.

4) **MapReduce(분산/병렬처리)** 기능을 제공한다.

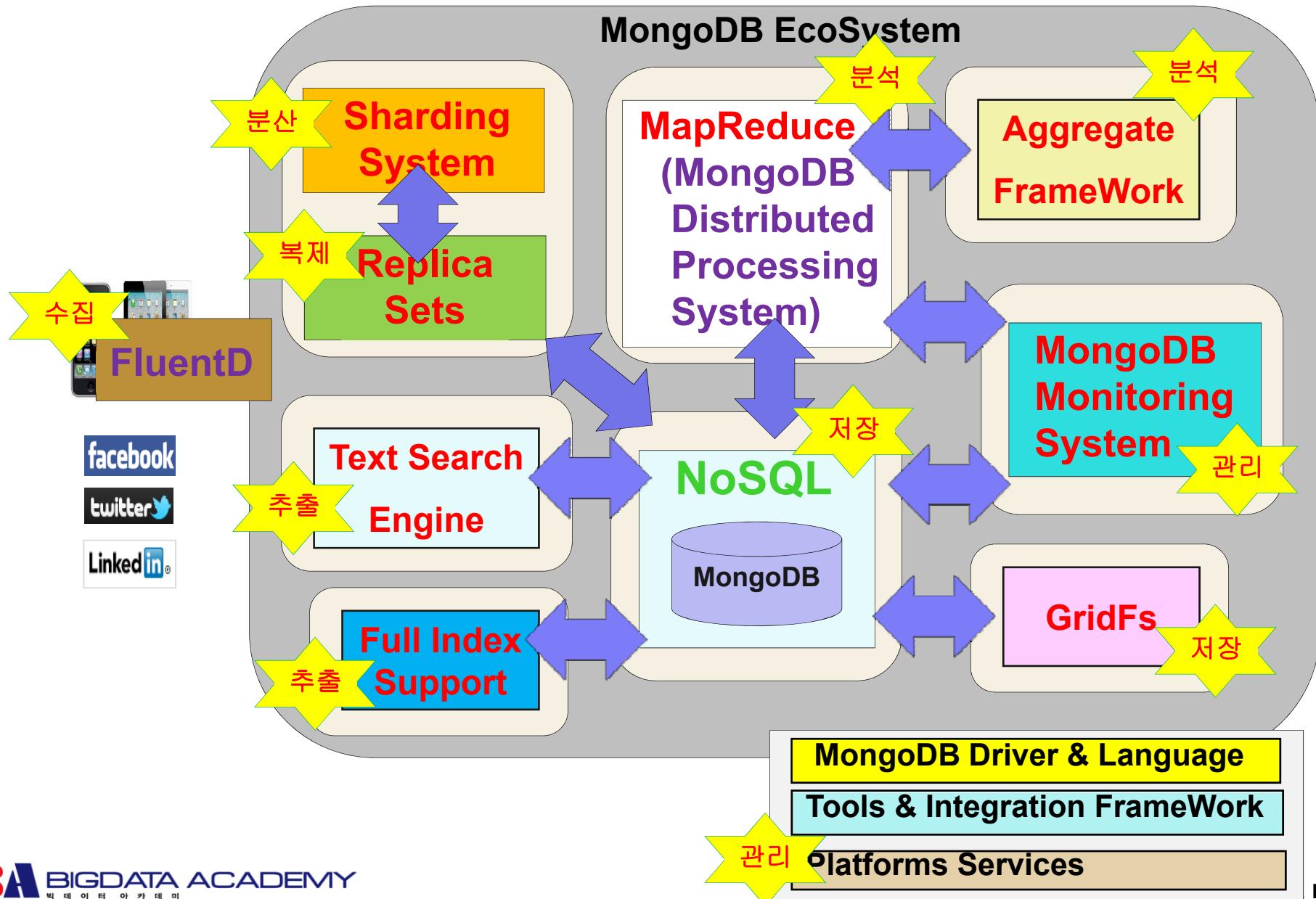
5) **CRUD(Create, Read, Update, Delete)** 위주의 다중 트랜잭션 처리도 가능하다.

6) **Memory Mapping** 기술을 기반으로 **Big Data** 처리에 탁월한 성능을 제공한다.

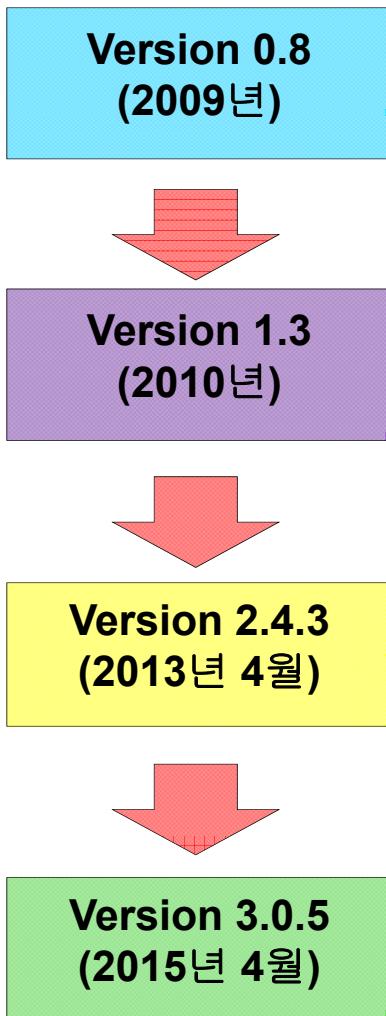


<http://www.youtube.com/watch?v=CvIrl-2IMLsk>

MongoDB EcoSystem



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

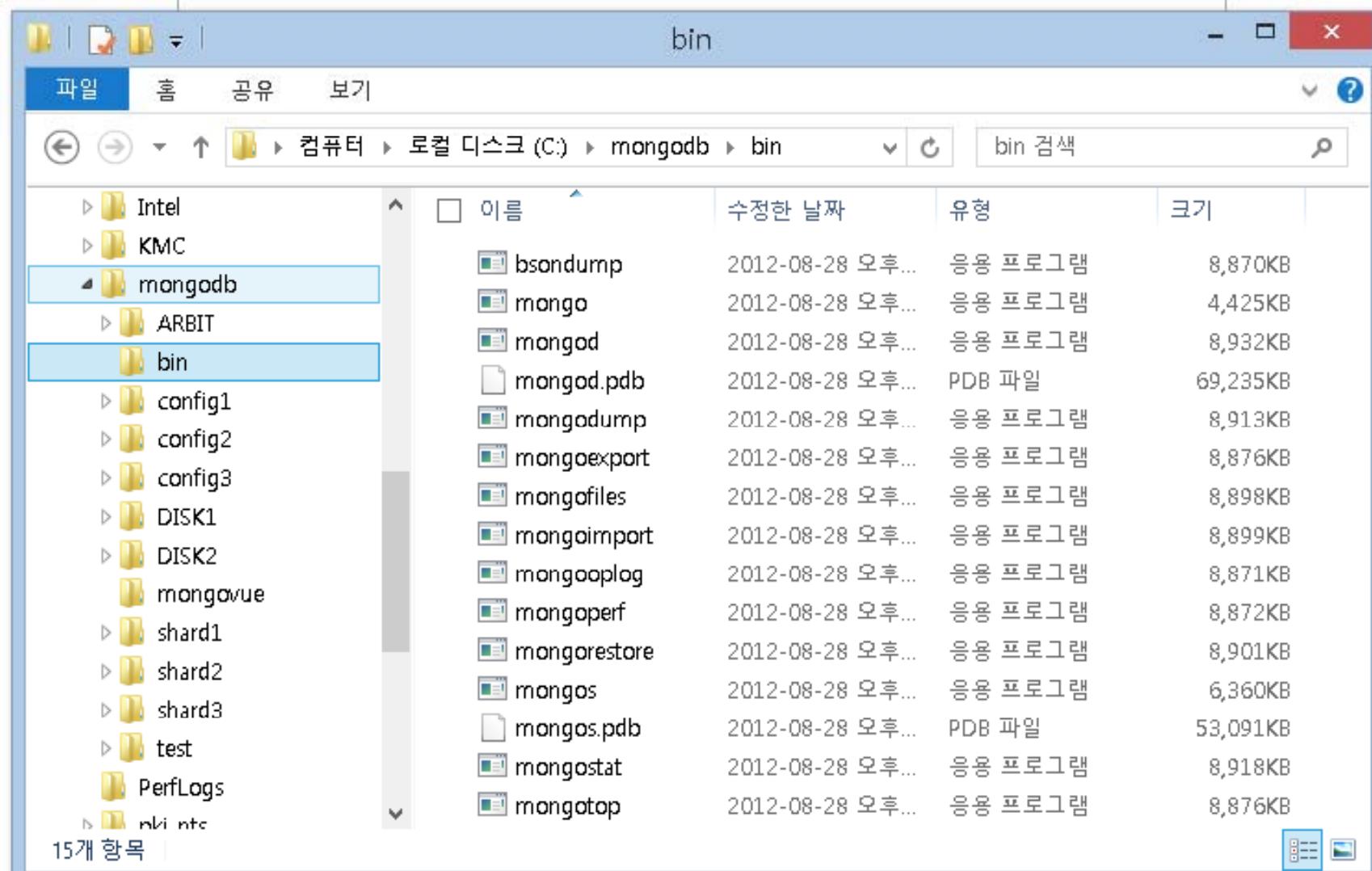
제품 타입

주요기능	MongoDB Personal	MongoDB Enterprise
JSON Data Model with Dynamic Schemas	•	•
Auto-Sharding for Horizontal Scalability	•	•
Built-In Replication and High Availability	•	•
Full, Flexible Index Support	•	•
Rich Document Queries	•	•
Fast In-Place Updates	•	•
Aggregation Framework and MapReduce	•	•
Large Media Storage with GridFS	•	•
Text Search	•	•
Cloud, On-Premise and Hybrid Deployments	•	•
Role-Based Privileges	•	•
Advanced Security with Kerberos		•
On-Prem Monitoring		•
SNMP Support		•
OS Certifications		•

The screenshot shows a Microsoft Internet Explorer window displaying the MongoDB Downloads page at <http://www.mongodb.org/downloads>. The page features a dark brown header with the MongoDB logo and navigation links for Forums, Blog, Twitter, Facebook, Events, and International. Below the header is a search bar. The main content area is titled "MongoDB Downloads" and includes links for "MONGO DOCS", "TRY IT OUT", "DRIVERS", and "Source". The central part of the page is a table listing MongoDB distributions by platform and version.

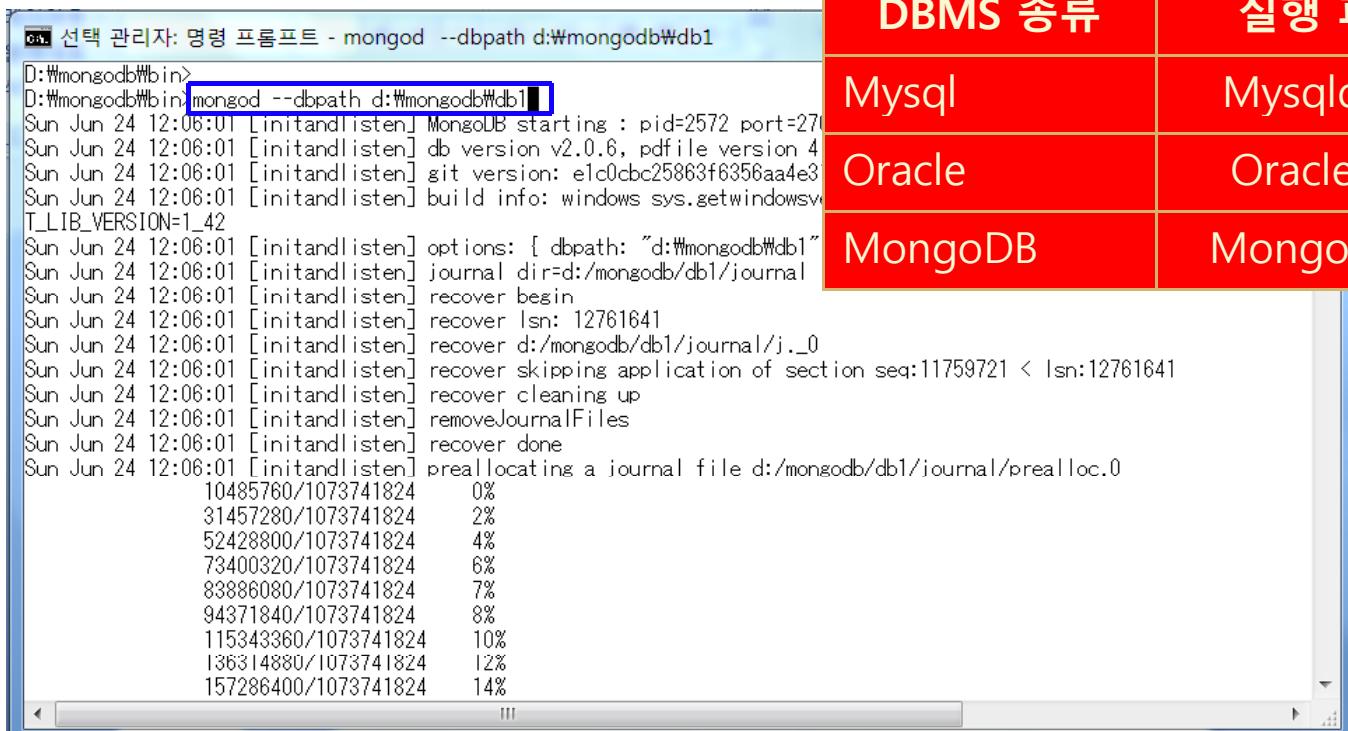
	OS X 64-bit	Linux 32-bit <small>note</small>	Linux 64-bit	Windows 32-bit <small>note</small>	Windows 64-bit	Solaris 64-bit	Source
Production Release (Recommended)							
2.2.1 10/30/2012 Changelog Release Notes	download	download *legacy-static	download *legacy-static	download	download *2008R2+	download	tgz zip
2.2.2-rc1 11/20/2012 Changelog	download	download *legacy-static	download *legacy-static	download	download *2008R2+	download	tgz zip

Demonstration



MongoDB의 시작

1) Mongod.exe



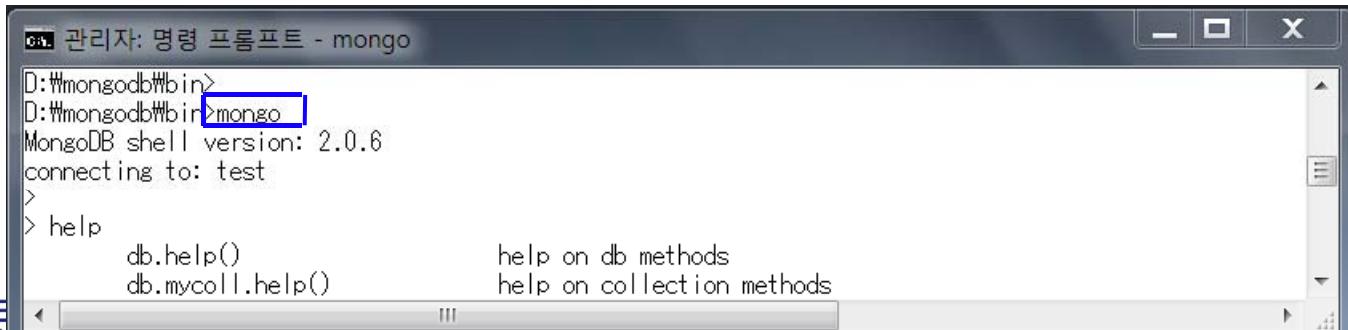
```

선택 관리자: 명령 프롬프트 - mongod --dbpath d:\mongodb\dbname
D:\mongodb\bin>mongod --dbpath d:\mongodb\dbname
Sun Jun 24 12:06:01 [initandlisten] MongoDB starting : pid=2572 port=27017
Sun Jun 24 12:06:01 [initandlisten] db version v2.0.6, pdfile version 4
Sun Jun 24 12:06:01 [initandlisten] git version: e1c0cbc25863f6356aa4e3
Sun Jun 24 12:06:01 [initandlisten] build info: windows sys.getwindowsversion()
T\_LIB\_VERSION=1.42
Sun Jun 24 12:06:01 [initandlisten] options: { dbpath: "d:\mongodb\dbname" }
Sun Jun 24 12:06:01 [initandlisten] journal dir=d:/mongodb/dbname/journal
Sun Jun 24 12:06:01 [initandlisten] recover begin
Sun Jun 24 12:06:01 [initandlisten] recover lsn: 12761641
Sun Jun 24 12:06:01 [initandlisten] recover d:/mongodb/dbname/journal/j_0
Sun Jun 24 12:06:01 [initandlisten] recover skipping application of section seq:11759721 < lsn:12761641
Sun Jun 24 12:06:01 [initandlisten] recover cleaning up
Sun Jun 24 12:06:01 [initandlisten] removeJournalFiles
Sun Jun 24 12:06:01 [initandlisten] recover done
Sun Jun 24 12:06:01 [initandlisten] preallocating a journal file d:/mongodb/dbname/journal/prealloc.0
    10485760/1073741824 0%
    31457280/1073741824 2%
    52428800/1073741824 4%
    73400320/1073741824 6%
    83886080/1073741824 7%
    94371840/1073741824 8%
    115343360/1073741824 10%
    136314880/1073741824 12%
    157286400/1073741824 14%

```

DBMS 종류	실행 파일	Client 툴
Mysql	Mysqld.exe	Mysql.exe
Oracle	Oracle.exe	Sqlplus.exe
MongoDB	Mongod.exe	Mongo.exe

2) Mongo.exe

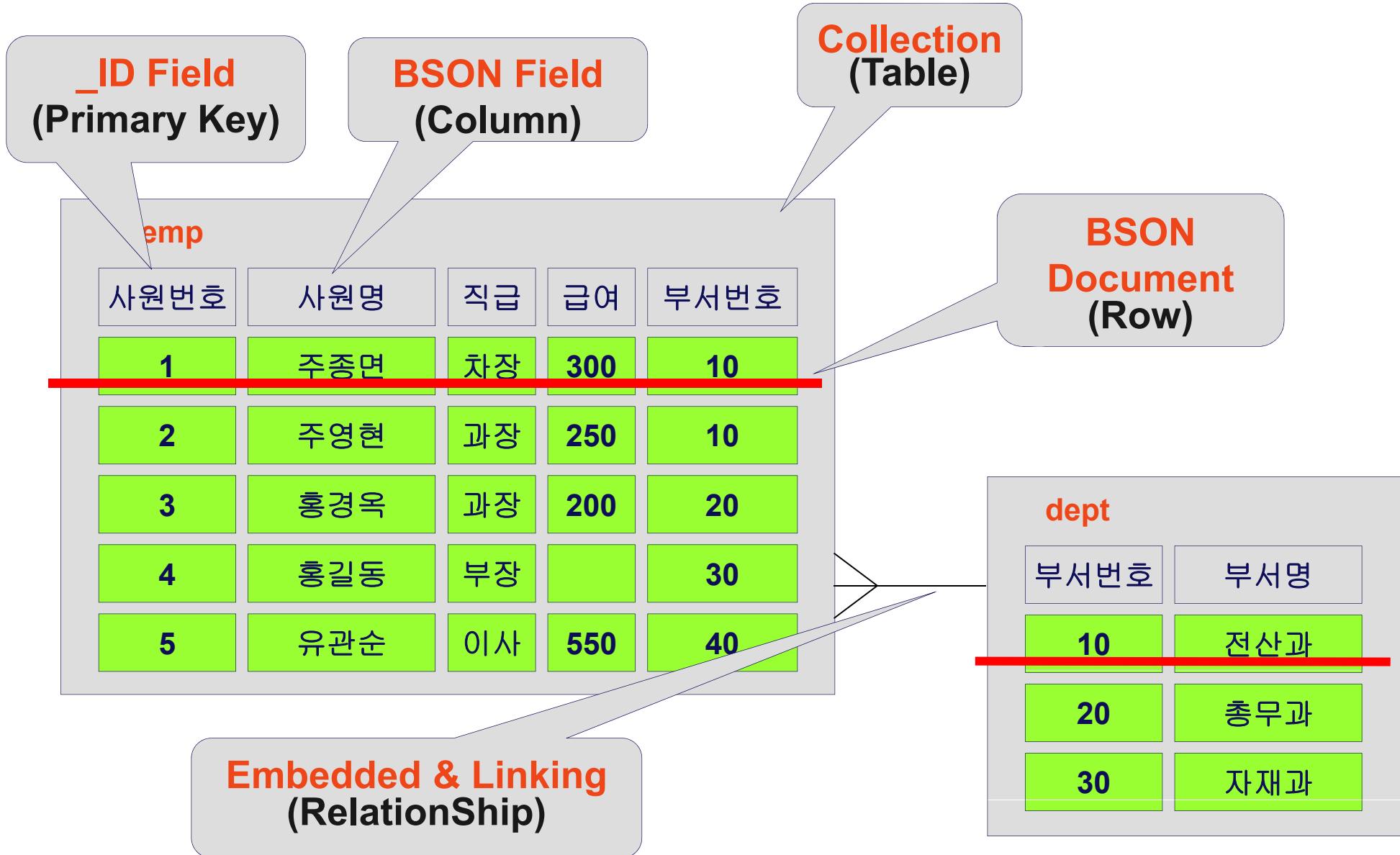


```

관리자: 명령 프롬프트 - mongo
D:\mongodb\bin>
D:\mongodb\bin>mongo
MongoDB shell version: 2.0.6
connecting to: test
>
> help
    db.help()                      help on db methods
    db.mycoll.help()                help on collection methods

```

Terminology



JSON (Java Script Object Notation)

사원 (EMP) Document

```
> p = { eno : 1101,  
        fname : "Adam",  
        lname : "Kroll",  
        job : "Manager",  
        salary : 100000,  
        dept_name : "SALES" }
```

괄호(Bracket)

```
> db.emp.save(p)
```

BSON (Binary Serial Object Notation)

- . 경량의 데이터 교환 형식인 **JSON**(JavaScript Object Notation) 타입을 근거로 하며 사람이 읽고 쓰기에 용이하며 기계가 분석하고 생성하기에 용이하다. (**Name**과 **Value**로 구성됨)
- . **JavaScript Programming Language**와 **Standard ECMA-262 3rd Edition-1999**을 근거로 한다.

{ **ename** : “주종면” }

Document Length

\x16\x00\x00\x00
\x06\x00\x00\x00

Value Type

\x02ename\x00
주종면\x00\x00

Value Length

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
```

```
true
```

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});

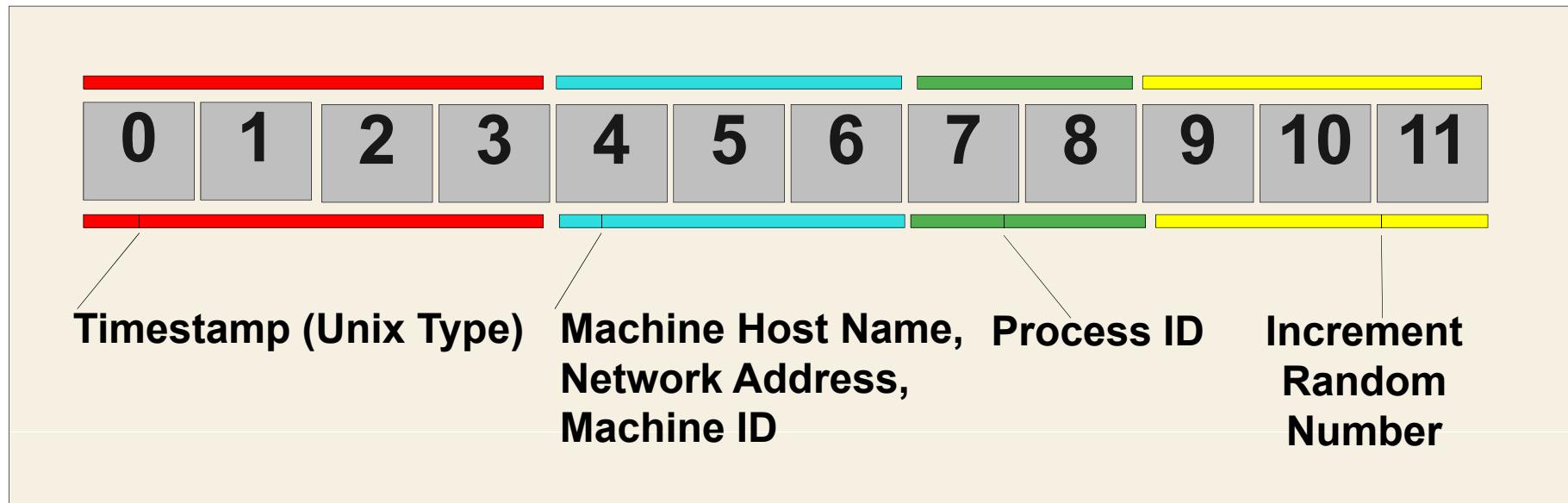
{ "_id" : ObjectId("4fe6852f5642c534a77fbdb1"),
  "eno" : 1103, "fname" : "SMITH", "job" : "CERK", "lname" : "STANFORD" }
{ "_id" : ObjectId("4fe685195642c534a77fbdb0"),
  "eno" : 1102, "fname" : "ADAM", "job" : "CHIEF", "lname" : "KROLL" }

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

Demonstration

BSON Object ID

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



SQL & Mongo Query 비교

SQL Statement

CREATE TABLE emp
(empno Number, ename Number)

INSERT INTO emp VALUES(3,5)

SELECT * FROM emp

SELECT empno, ename FROM emp

SELECT * FROM emp WHERE empno=3

SELECT empno, ename
FROM emp WHERE empno=3

SELECT * FROM emp
WHERE empno=3 ORDER BY ename

Mongo Query Statement

db.createCollection("emp")

db.emp.insert({empno:3, ename:5})

db.emp.find()

db.emp.find({}, {empno:1, ename:1})

db.emp.find({empno:3})

db.emp.find({empno:3}, {empno:1, ename:1})

db.emp.find({empno:3}).sort({ename:1})

SQL Statement

```
SELECT * FROM emp WHERE empno > 3
```

```
SELECT * FROM emp WHERE empno != 3
```

```
SELECT * FROM emp  
WHERE ename LIKE "%Joe%"
```

```
SELECT * FROM emp WHERE ename LIKE "Joe%"
```

```
SELECT * FROM emp  
WHERE empno>1 AND empno <=4
```

```
SELECT * FROM emp ORDER BY ename DESC
```

```
SELECT * FROM emp  
WHERE empno=1 and ename='Joe'
```

```
SELECT * FROM emp  
WHERE empno=1 or empno=3
```

```
SELECT * FROM emp WHERE rownum = 1
```

NoSQL Statement

```
db.emp.find({empno:{$gt:3}})
```

```
db.emp.find({empno:{$ne:3}})
```

```
db.emp.find({ename:/Joe/})
```

```
db.emp.find({ename:/^Joe/})
```

```
db.emp.find({empno:{$gt:1,$lte:4}})
```

```
db.emp.find().sort({ename:-1})
```

```
db.emp.find({empno:1,ename:'Joe'})
```

```
db.emp.find( { $or : [ { empno : 1 } , { empno : 3 } ] } )
```

```
db.emp.findOne()
```

SQL Statement

```
SELECT empno FROM emp o, dept d
WHERE d.deptno=o.deptno AND d.deptno=10
```

```
SELECT DISTINCT ename FROM emp
```

```
SELECT COUNT(*) FROM emp
```

```
SELECT COUNT(*) FROM emp where deptno > 10
```

```
SELECT COUNT(sal) from emp
```

```
CREATE INDEX i_emp_ename ON emp(ename)
```

```
CREATE INDEX i_emp_no
ON emp(deptno ASC, ename DESC)
```

```
UPDATE emp SET ename='test' WHERE empno=1
```

```
DELETE FROM emp WHERE deptno=10
```

NoSQL Statement

```
o = db.emp.findOne({empno:1});
name = db.dept.findOne({deptno:o.deptno})
```

```
db.emp.distinct('ename')
```

```
db.emp.count()
```

```
db.emp.find({deptno: {'$gt': 10}}).count()
```

```
db.emp.find({sal: {'$exists': true}}).count()
```

```
db.emp.ensureIndex({ename:1})
```

```
db.emp.ensureIndex({deptno:1,ename:-1})
```

```
db.emp.update({empno:1}, {$set:{ename:' test'}})
```

```
db.emp.remove({deptno:10});
```

6 차시

데이터 추출 및 분석

1) MongoDB의 **Aggregation Framework** 함수를
이용한 빅 데이터의 추출

2) MongoDB의 **Map/Reduce** 기능을 이용한
빅 데이터의 추출

3) MongoDB와 **Hadoop**의 **Map-Reduce**를 연동
한 빅 데이터의 추출

1) Aggregation Framework 함수를 이용한 데이터 추출

- Aggregation Framework는 데이터 추출에 최적화되어 만들어진 기능이다. (v 2.1 지원)
- Aggregation을 위해 MongoDB의 Map/Reduce를 사용하여 빠른 성능을 보장한다.
- 실시간 Aggregation은 SQL의 Group 함수와 유사하다.
- MongoDB Map/Reduce는 JavaScript로 생성되어 있다.
- JavaScript는 외부 데이터 처리에 제한적이며 MongoDB 내의 데이터만 처리할 수 있다.

분석 대상 데이터

```
> db.yield_historical.in.find().pretty()
{
  "_id" : ISODate("1990-01-02T00:00:00Z"),
  "dayOfWeek" : "TUESDAY",
  "bc3Year" : 7.9,
  "bc5Year" : 7.87,
  "bc10Year" : 7.94, ← 집계 대상 Field
  "bc20Year" : null,
  "bc1Month" : null,
}

> db.yield_historical.out.find()
{ "_id" : 1990, "value" : 8.552400000000002 }
{ "_id" : 1991, "value" : 7.8623600000000025 }
{ "_id" : 1992, "value" : 7.008844621513946 }
{ "_id" : 1993, "value" : 5.866279999999999 }
{ "_id" : 1994, "value" : 7.085180722891565 }
{ "_id" : 1995, "value" : 6.573920000000002 }
{ "_id" : 1996, "value" : 6.443531746031742 }
{ "_id" : 1997, "value" : 6.353959999999992 } ← 집계 대상 Field
{ "_id" : 1998, "value" : 5.262879999999994 }
{ "_id" : 1999, "value" : 5.646135458167332 }

.....
.....
```

```
{
  "_id" : ISODate("1997-01-03T00:00:00Z"),
  "dayOfWeek" : "WEDNESDAY",
  "bc3Year" : 7.96,
  "bc5Year" : 7.92,
  "bc10Year" : 7.99, ← 집계 대상 Field
  "bc20Year" : null,
  "bc1Month" : null,
  "bc2Year" : 7.94,
  "bc3Month" : 7.89,
  "bc30Year" : 8.04,
  "bc1Year" : 7.85,
  "bc7Year" : 8.04,
  "bc6Month" : 7.94
```

Sample

```
db.yield_histor
```

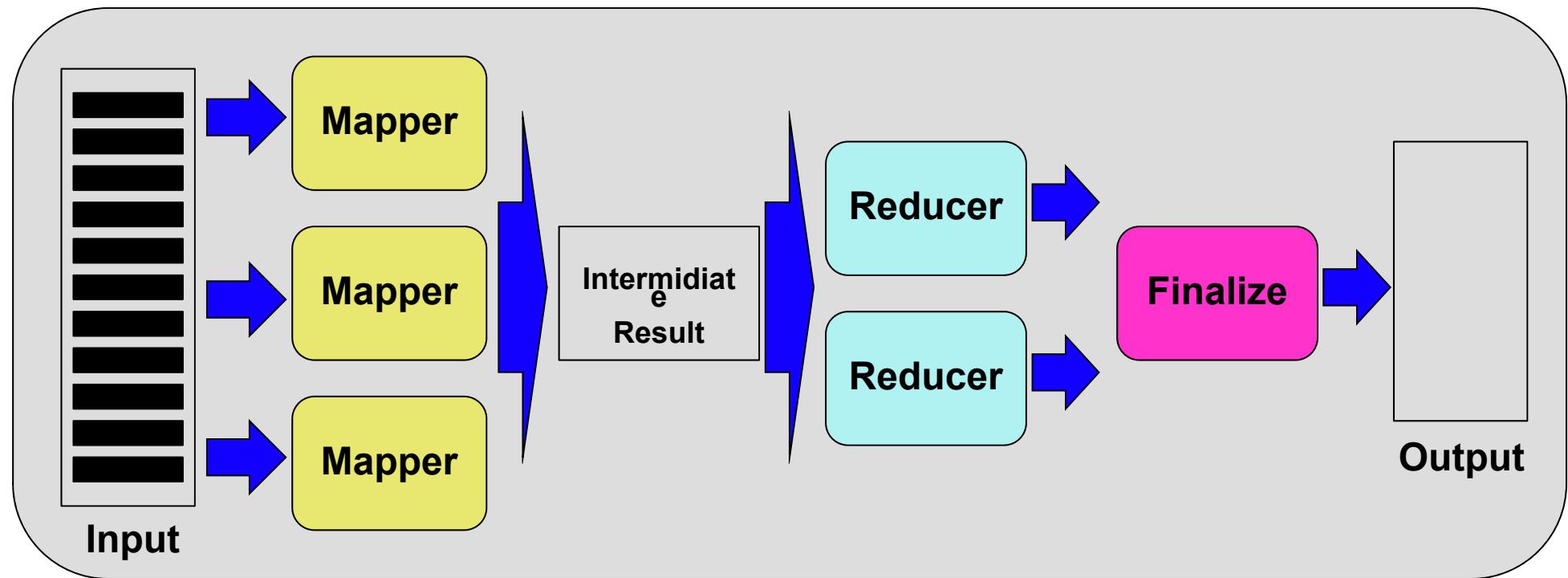
```
{ $project :  
  { _id : 1, bc10Year : 1 } ,  
 { $match :  
  {bc10Year : { $gte : 0.00, $lte : 9.99 } } ,  
 { $group :  
  { _id : { $year : "$_id" } ,  
    value : { $avg : "$bc10Year" } } } ,  
 { $sort :  
  { _id : 1 } } }
```

```
SELECT year, avg(bc10Year)  
FROM yield_historical.in  
WHERE bc10Year >= 0.00  
      and bc10Year <= 9.99  
GROUP BY year  
ORDER BY id asc;
```

Demonstration

MapReduce

- . 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 있다.
- . **MAP**과 **REDUCE** 함수 만으로 병렬 프로그래밍이 가능하다.



MongoDB Map/Reduce

```
> db.runCommand ({ mapreduce: "yield_historical.in",  
query: { bc10Year : { $gt : 0.00, $lt : 9.99 } },
```

검색 대상 컬렉션과
검색조건을 정의한다.

```
map: function() {  
    var d_year = this._id;  
    var value = this.bc10Year;  
    var key = d_year.getYear()+1900;
```

MAP 함수
하나의 Document씩 읽어온다

```
    emit( { d1:key }, { msum : this.bc10Year, recs : 1 } ); },
```

```
reduce: function(key, vals)
```

REDUCE 함수
Bc10year 필드를 집계하고 Count 한다.

```
{ msum : this.bc10Year; recs : 1 ;
```

```
var ret = { msum:0, recs:0 };
```

```
for (var i=0 ; i < vals.length; i++)  
    { ret.msum += vals[i].msum;  
        ret.rec
```

```
        s += vals[i].recs;
```

```
    } return ret; },
```

FINALIZE 함수
년도별 항목별 평균

```
finalize: function(key, val)
```

```
{ val.mavg=val.msum/val.rec
```

```
s; return val; },
```

```
out: "result1",
```

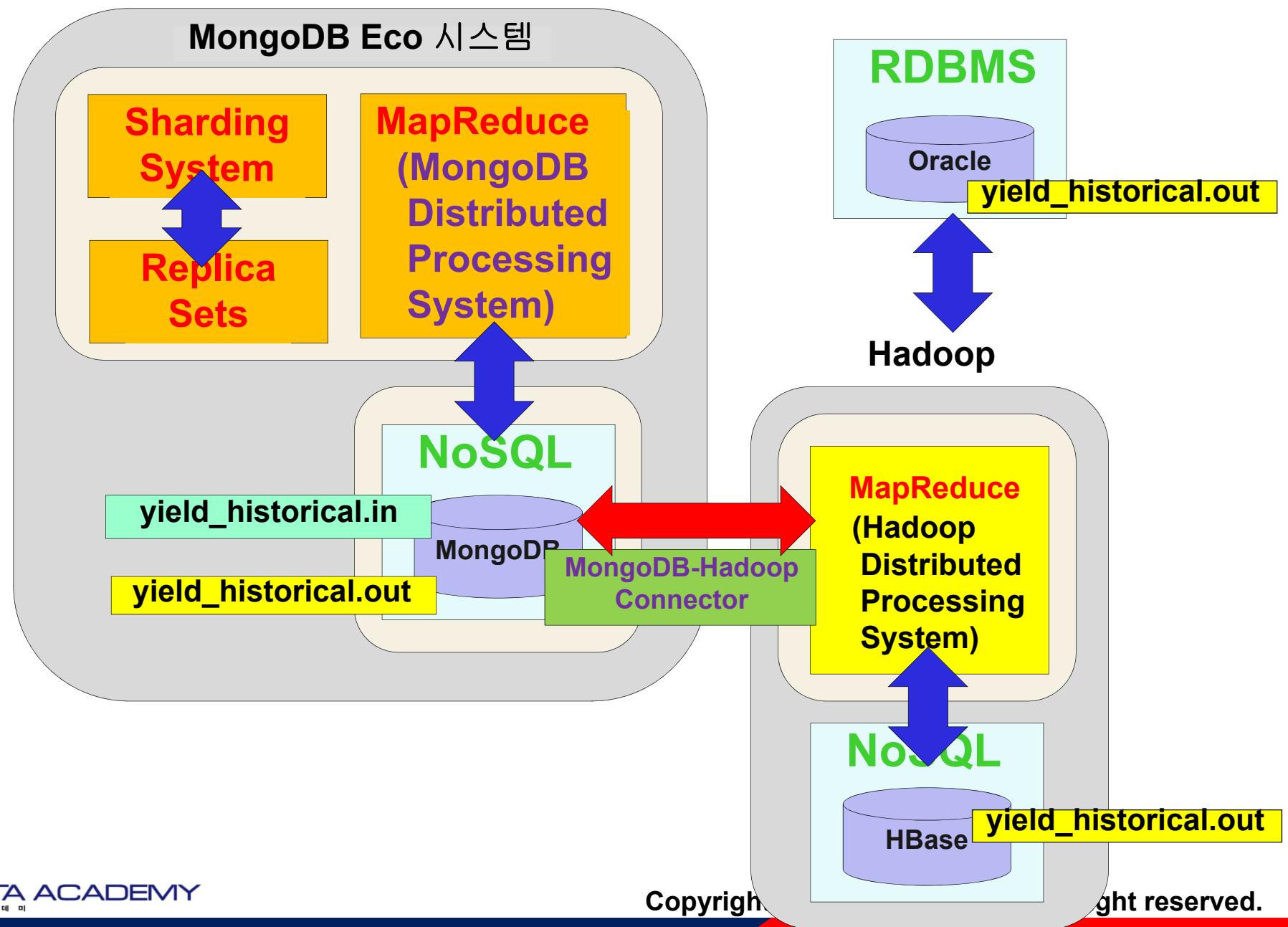
```
verbose: true } );
```

처리된 결과를
Result1에 저장한다.

```
db.yield_historical.in.aggregate(  
{ $project :  
    { _id : 1, bc10Year : 1 } },  
{ $match :  
    {bc10Year : { $gte : 0.00, $lte : 9.99 }},  
{ $group :  
    { _id : { $year : "$_id" },  
        value : { $avg : "$bc10Year" } } } },  
{ $sort :  
    { _id : 1 } } )
```

Demonstration

MongoDB & Hadoop 연동



(1) MongoDB와 Hadoop Map/Reduce 연동은 다양한 NoSQL 기술을 이용한 데이터 저장 환경에서 **타 DBMS** 간에 원활한 데이터 이관을 위해 요구된다.

(2) MongoDB Eco System과 Hadoop Eco System은 상호 보완적인 기술이며 MongoDB Eco System 만으로도 빅 데이터 처리가 가능하다.

(3) 빅 데이터 처리를 기반으로 하는 오픈 소스 환경은 절대적 솔루션이 있는 것이 아니라 사용자 환경에 맞는 최적의 오픈 소스를 필요에 따라 전략적으로 선택할 수 있다.

Map 함수 in Java

```
public class TreasuryYieldMapper  
    extends Mapper<Date, BSONObject, IntWritable, DoubleWritable> {
```

Hadoop의
Map 함수

@Override

```
public void map( final Date pKey,  
                 final BSONObject pValue,  
                 final Context pContext )  
    throws IOException, InterruptedException{
```

```
final int year = pKey.getYear() + 1900;
```

```
double bid10Year = ( (Number) pValue.get( "bc10Year" ) ).doubleValue();
```

```
pContext.write( new IntWritable( year ), new DoubleWritable( bid10Year ) );  
}
```

Reduce 함수 in Java

```
public class TreasuryYieldReducer  
    extends Reducer<IntWritable, DoubleWritable, IntWritable, DoubleWritable> {  
  
    @Override  
    public void reduce( final IntWritable pKey,  
                        final Iterable<DoubleWritable> pValues,  
                        final Context pContext )  
        throws IOException, InterruptedException{  
        int count = 0;  
        double sum = 0;  
        for ( final DoubleWritable value : pValues ) {  
            LOG.debug( "Key: " + pKey + " Value: " + value );  
            sum += value.get();  
            count++;  
        }  
        final double avg = sum / count;  
        LOG.debug( "Average 10 Year Treasury Yield: " + avg );  
        pContext.write( pKey, new DoubleWritable( avg ) );  
    }  
    private static final Log LOG = LoggerFactory.getLogger( TreasuryYieldReducer.class );  
}
```

```
> db.yield_historical.out.find()
{ "_id" : 1990, "value" : 8.552400000000002 }
{ "_id" : 1991, "value" : 7.8623600000000025 }
{ "_id" : 1992, "value" : 7.008844621513946 }
{ "_id" : 1993, "value" : 5.866279999999999 }
{ "_id" : 1994, "value" : 7.085180722891565 }
{ "_id" : 1995, "value" : 6.573920000000002 }
{ "_id" : 1996, "value" : 6.443531746031742 }
{ "_id" : 1997, "value" : 6.353959999999992 }
{ "_id" : 1998, "value" : 5.262879999999994 }
{ "_id" : 1999, "value" : 5.646135458167332 }
```

7 차시

트랜잭션 처리 & 인덱스

Isolation 유형

iSolation 타입	Dirty Read	Unrepeatable Read	Phantom Read	비고
Read Uncommitted	Y	Y	Y	Dirty Read 연산을 허용, 잠금/해제가 발생하지 않음
Read Committed	N	Y	Y	Unrepeatable와 Phantom 연산을 허용함
Repeatable Read	N	N	Y	Phantom Read 연산만 허용함
Serializable	N	N	N	데이터 일관성을 완벽하게 보장하지만 동시성은 떨어짐

* Dirty Read : commit 되지 않은 데이터를 Read 할 수 있다.

* Unrepeatable Read : 하나의 트랜잭션에서 하나의 컬렉션을 2번 검색할 때 Field의 변경된 다른 값을 Read할 수 있다.

* Phantom Read : 하나의 트랜잭션에서 하나의 컬렉션을 2번 검색할 때 추가된 Document 값을 Read할 수 있다.

- 1) 읽기 일관성과 데이터 공유를 위해 V 1.8까지 **Global Lock**을 제공하였으며 V 3.0부터 **Document Lock**을 제공한다.
- 2) Lock 문제로 인한 성능 저연 문제 해소를 위해 **PageFaultException** 기능을 추가로 제공한다. (V 2.2)



Two Task Rollback

```
db.accounts.save({name: "Jimm", balance: 2500, pendingTransactions: []})
db.accounts.save({name: "King", balance: 1300, pendingTransactions: []})
```

```
db.transactions.save({ source: "Jimm", destination: "King", value: 100, state: "initial"})
```

```
t = db.transactions.findOne({state: "initial"})
db.transactions.update({_id: t._id}, {$set: {state: "pending"}})
```

```
db.accounts.update({ name: t.source, pendingTransactions: {$ne: t._id}},
                    {$inc: {balance: -t.value}, $push: {pendingTransactions: t._id}})
db.accounts.update({ name: t.destination, pendingTransactions: {$ne: t._id}},
                    {$inc: {balance: t.value}, $push: {pendingTransactions: t._id}})
```

```
db.transactions.update({_id: t._id}, {$set: {state: "canceling"}})
```

```
db.accounts.update({name: t.source, pendingTransactions: t._id},
                    {$inc: {balance: t.value}, $pull: {pendingTransactions: t._id}})
db.accounts.update({name: t.destination, pendingTransactions: t._id},
                    {$inc: {balance: -t.value}, $pull: {pendingTransactions: t._id}})
```

```
db.transactions.update({_id: t._id}, {$set: {state: "cancelled"}})
```

Session-1

```
t = db.transactions.findOne({state: "initial"})
ObjectId("501b33f0681dd7da6321da42")

db.transactions.update(
 {_id: t._id}, {$set: {state: "pending"}})

db.accounts.update({ name: t.source, pendingTransactions:{$ne:t._id}},
 {$inc: {balance: -t.value}, $push: {pendingTransactions: t._id}})
db.accounts.update({ name: t.destination, pendingTransactions: {$ne: t._id}},
 {$inc: {balance: t.value}, $push: {pendingTransactions: t._id}})
```

Session-2

```
db.accounts.find()
{"balance":2400,"name":"Jimm",
 "pendingTransactions":ObjectId("501b33f0681dd7da6321da42")}

{"balance":1400,"name":"King",
 "pendingTransactions":ObjectId("501b33f0681dd7da6321da42")}
```

INDEX 종류

Non-Unique/Unique Index

GeoSpatial(2d) Index

Background Index

GeoSpatial(2dsphere) Index

Covered Index

GeoHayStack Index

DropDups Index

Text Index

Sparse Index

Hashed Index

TTL Index

INDEX 종류

INDEX 타입	설명
Non Unique Index (Single Key Index Compound Key Index)	하나 또는 하나 이상의 중복 값을 가진 Field로 구성되는 Index 타입으로 가장 대표적인 Balance Tree Index 구조로 생성된다. (예) db.things.ensureIndex({"city": 1}) db.things.ensureIndex({ deptno:1, loc:-1})
Unique Index	Index가 생성되는 Field가 유일한 속성 값을 가진 Index 타입이다. (예) db.things.ensureIndex({fname: 1, lname: 1}, {unique: true})
Sparse Index	하나 이상의 필드에 Null 값을 가진 데이터가 대부분이고 드물게 어떤 데이터를 값을 가지고 있는 경우에 생성하는 효율적이다. (예) db.people.ensureIndex({title : 1}, {sparse : true}) db.people.save({name:"Jim"}) db.people.save({name:"Sarah", title:"Princess"}) db.people.find().sort({title:1}) {name:"Sarah", title:"Princess"}

INDEX 타입

설명

Background Index

일반적으로 Index의 생성은 데이터베이스 전체의 성능 지연 현상을 유발시킬 수 있다. V1.3.2부터 Background에서 Index를 생성할 수 있다.

(예) db.people.ensureIndex({**idate : 1**}, {background : true})

Covered Index

여러 개의 Field로 생성된 Index를 검색할 때 Index 만의 검색 만으로도 조건을 만족하는 Document를 추출할 수 있는 타입이다.

(예)

```
db.users.ensureIndex( { username : 1, password : 1, roles : 1} );
db.users.save ({username: "joe", password: "pass", roles: 2})
db.users.save ({username: "liz", password: "pass2", roles: 4})
db.users.find ({username: "joe"}, { _id: 0, roles: 1})
{ "roles" : 2 }
db.users.find ({username: "joe"}, { _id: 0, roles: 1}).explain()
{ "cursor" : "BtreeCursor username_1_password_1_roles_1", ...
  "indexOnly" : true }
```

DropDups Index

동일한 값이 여러 개 저장되어 있는 필드에 DropDups Index를 생성하면 최초 입력된 Document 만 남고 나머지 Document는 제거된다.

(예) db.people.ensureIndex({**idate : 1**}, {dropdups: true})

GeoSpatial Index

좌표로 구성되는 2차원 구조로 하나의 Collection에 하나의 2D Index를 생성할 수 있다.(다음 페이지에서 자세히 소개됨)

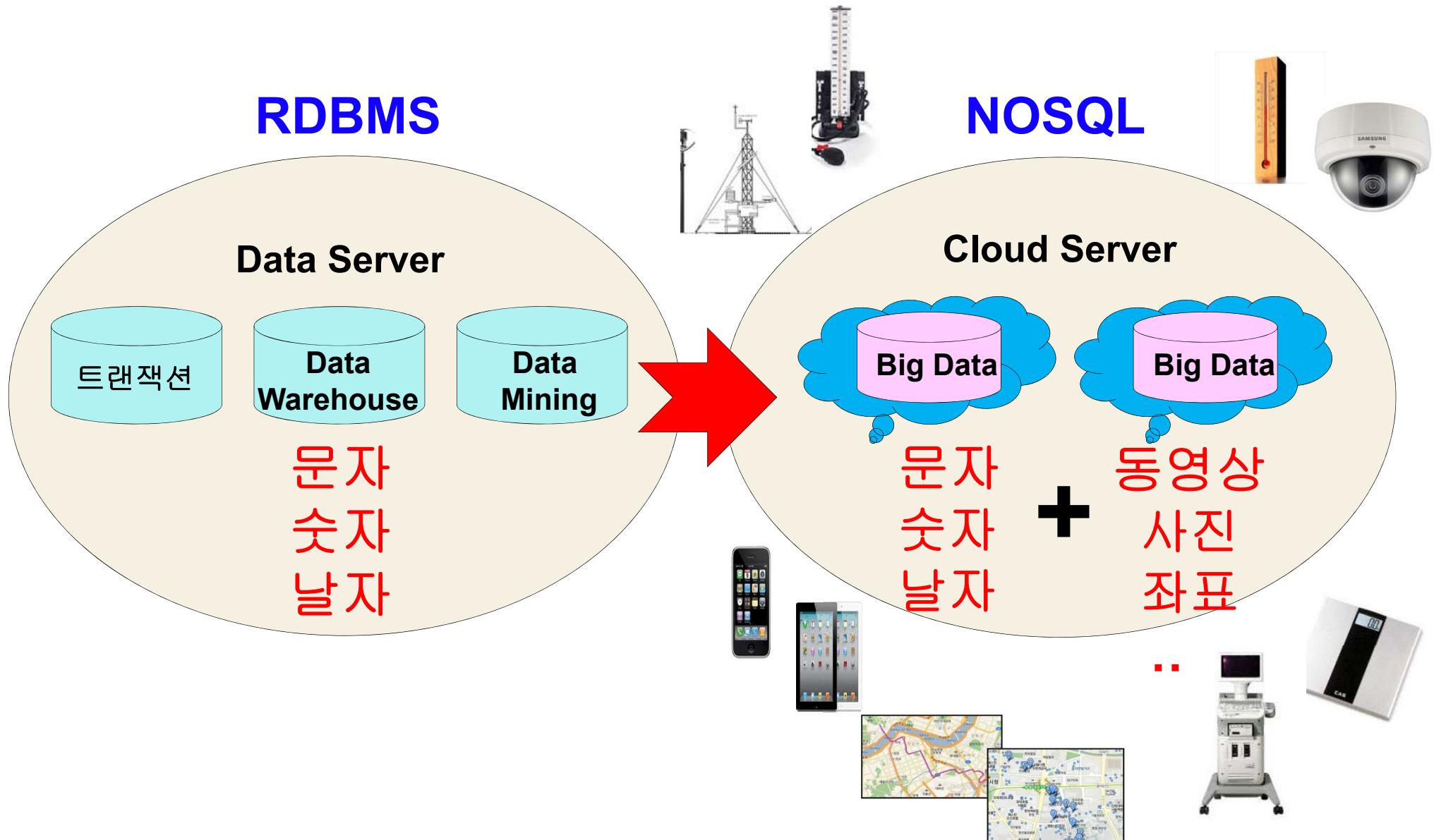
INDEX 생성과 관리

```
> db.emp.ensureIndex ({ eno: 1 }, { unique: true });
    ← 1 (Asc), -1 (Desc)
> db.emp.ensureIndex ({ job : -1 });
```

```
> db.emp.getIndexes()
{
    "v" : 1,
    "key" : {
        "eno" : 1
    },
    "unique" : true,
    "ns" : "test.emp",
    "name" : "eno_1"
}

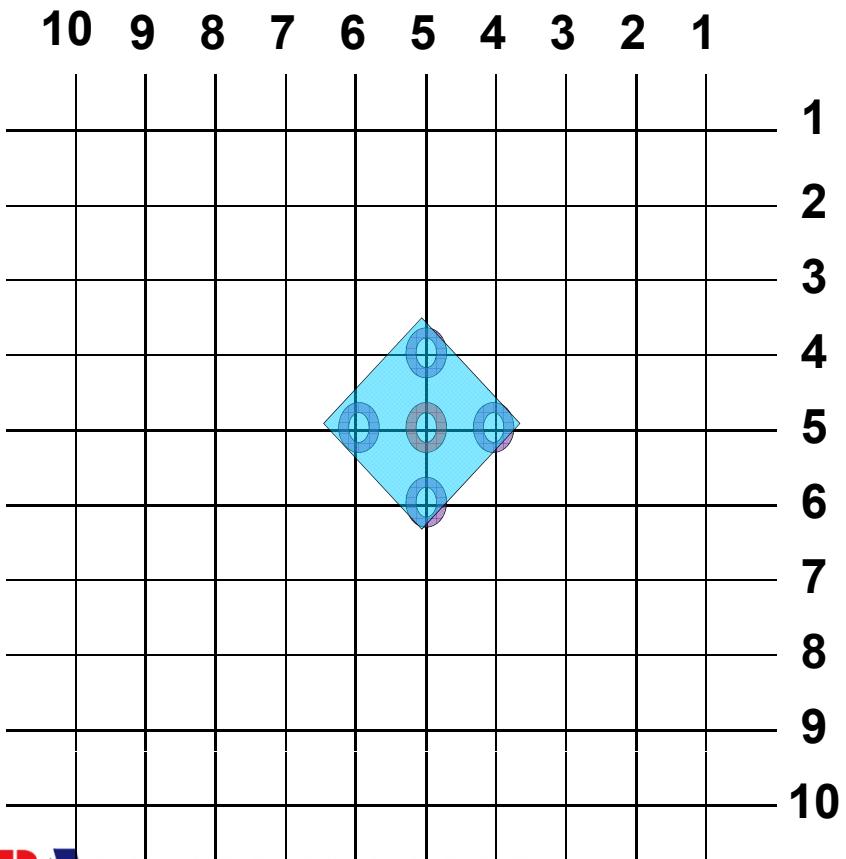
> db.system.indexes.find()
{ "v" : 1, "key" : { "eno" : 1 },
  "unique" : true, "ns" : "test.emp", "name" : "eno_1" }
```

RDBMS & NOSQSL

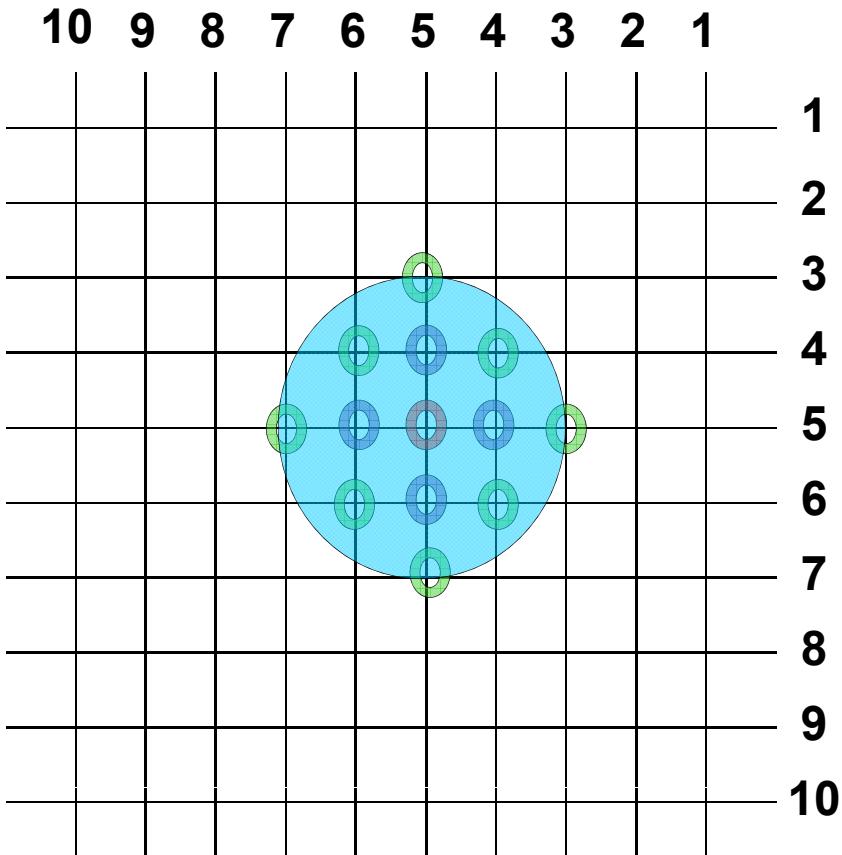


GeoSpatial INDEX

. 좌표에 의해 구성되는 2차원 구조로 하나의 **Collection**에
하나의 **2D Index**를 생성할 수 있다.



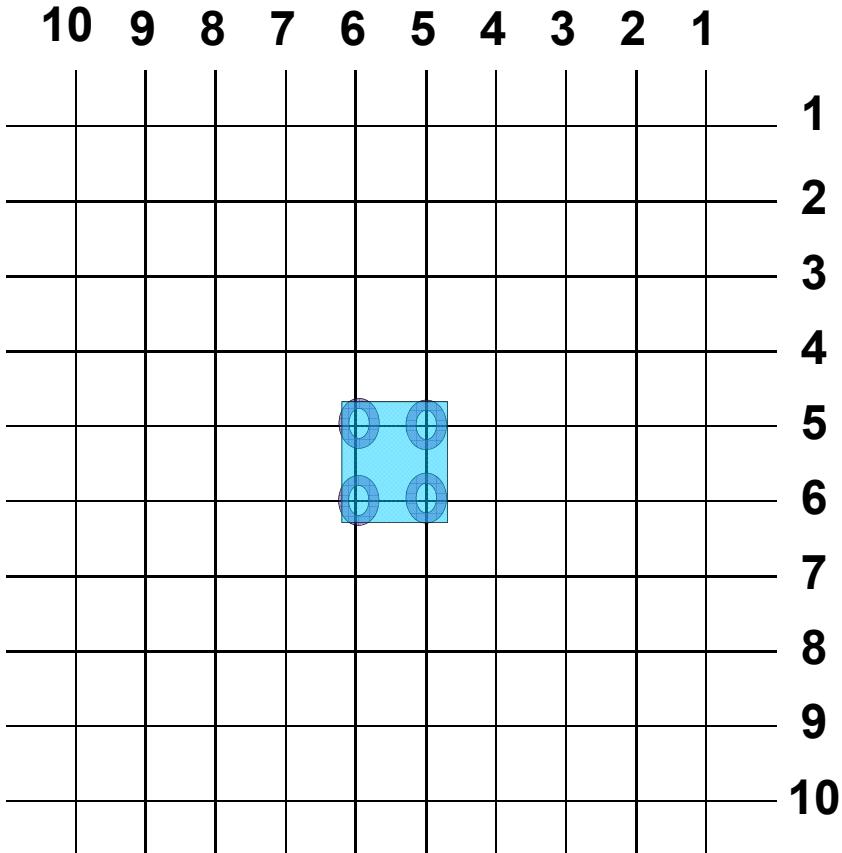
```
> for ( var i = 0; i < 10; i++ )  
    db.square.insert ({  
        pos : [ i % 10, Math.floor( i / 10 ) ] } )  
  
> db.square.ensureIndex ({ pos : "2d" })  
  
> db.square.find({ pos : { $near : [5, 5] } }).limit(5)  
{ "pos" : [ 5, 5 ] }  
{ "pos" : [ 5, 4 ] }  
{ "pos" : [ 4, 5 ] }  
{ "pos" : [ 5, 6 ] }  
{ "pos" : [ 6, 5 ] }
```



> db.square.find(

```
{ pos : { $within : { $center : [ [5, 5], 2 ] } },  
  { _id : 0 })  
  
{ "pos" : [ 5, 5 ] }  
{ "pos" : [ 5, 4 ] }  
{ "pos" : [ 5, 3 ] }  
{ "pos" : [ 4, 5 ] }  
{ "pos" : [ 3, 5 ] }  
{ "pos" : [ 4, 4 ] }  
{ "pos" : [ 4, 6 ] }  
{ "pos" : [ 5, 6 ] }  
{ "pos" : [ 5, 7 ] }  
{ "pos" : [ 6, 4 ] }  
{ "pos" : [ 6, 5 ] }  
{ "pos" : [ 7, 5 ] }  
{ "pos" : [ 6, 6 ] }
```

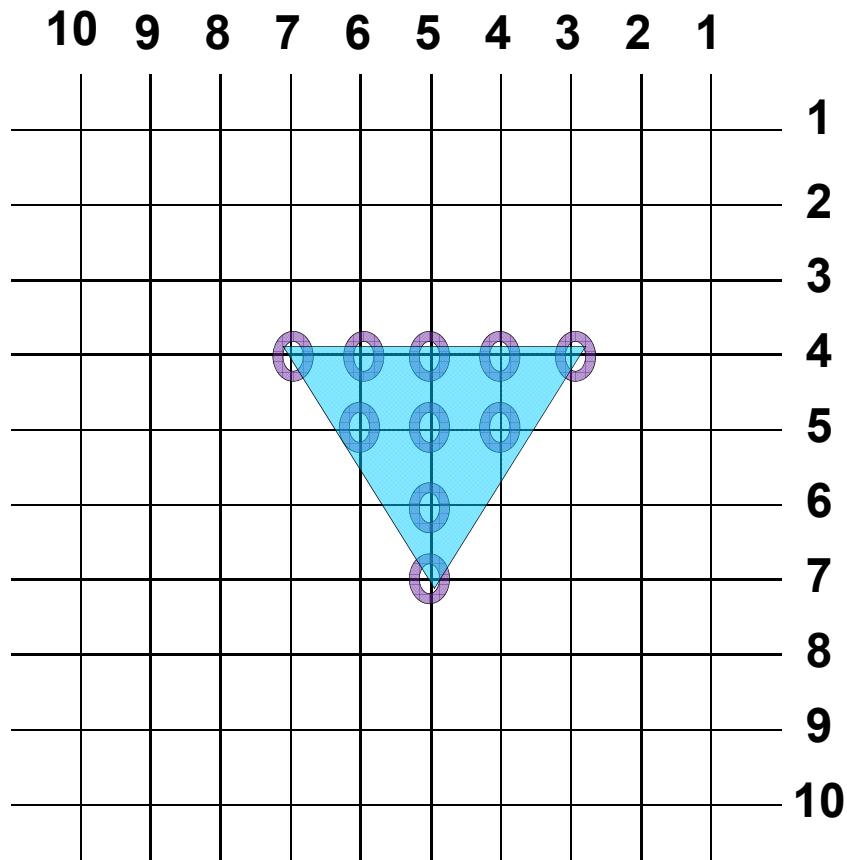
\$BOX



```
> db.square.find()
```

```
{ pos : { $within : { $box : [ [5, 5], [6, 6] ] } },  
  { _id : 0 })  
  
{ "pos" : [ 5, 5 ] }  
{ "pos" : [ 5, 6 ] }  
{ "pos" : [ 6, 5 ] }  
{ "pos" : [ 6, 6 ] }
```

\$POLYGON



```
> db.square.find({ pos : { $within :  
  { $polygon : [[3, 4], [5, 7], [7, 4]] } }}, { _id : 0 })  
1  { "pos" : [ 5, 5 ] }  
2  { "pos" : [ 6, 5 ] }  
3  { "pos" : [ 7, 4 ] }  
4  { "pos" : [ 6, 4 ] }  
5  { "pos" : [ 5, 7 ] }  
6  { "pos" : [ 5, 6 ] }  
7  { "pos" : [ 3, 4 ] }  
8  { "pos" : [ 4, 4 ] }  
9  { "pos" : [ 4, 5 ] }  
10 { "pos" : [ 5, 4 ] }
```

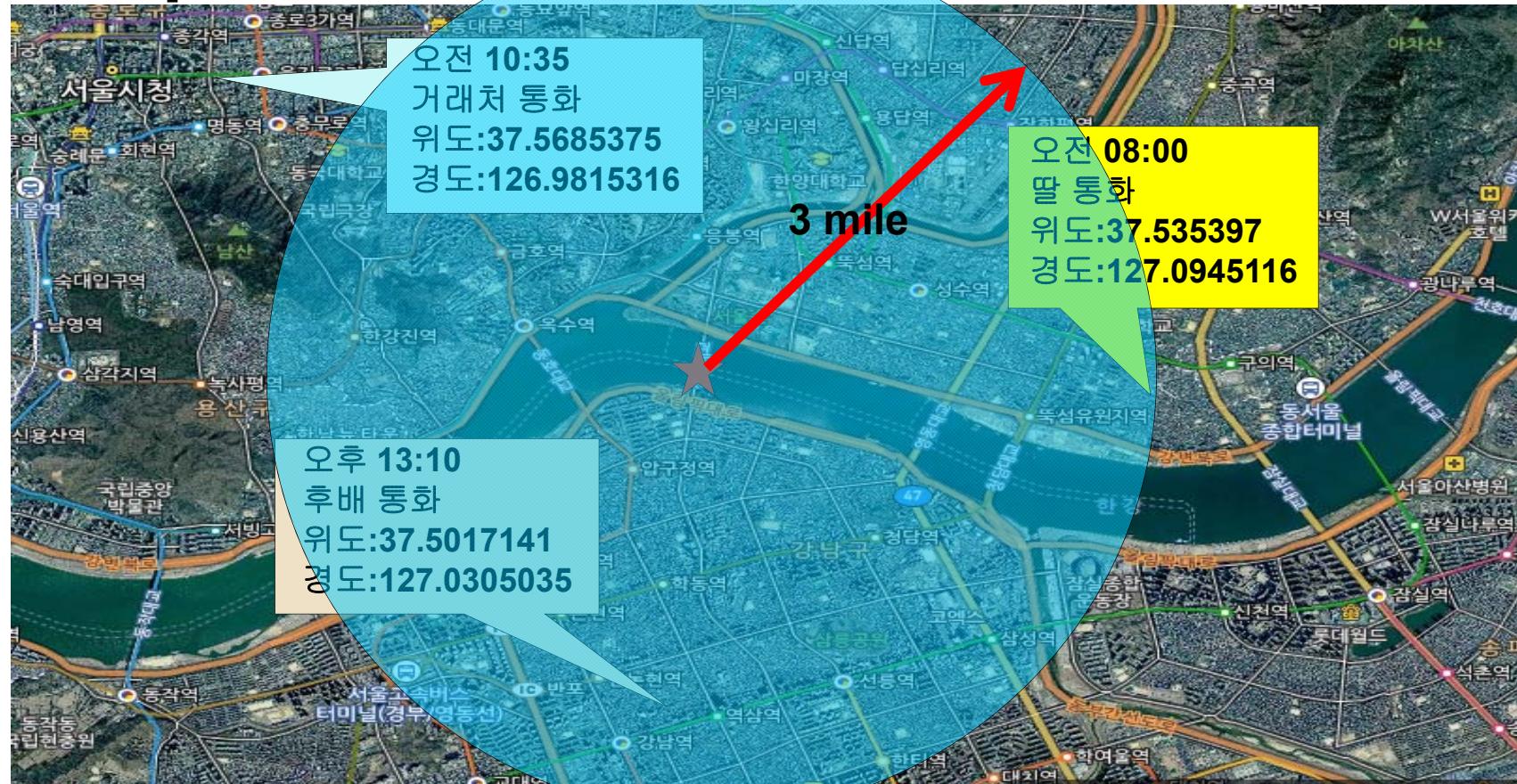
Demonstration

Multi-Location Documents



```
> db.tel_pos.save ({ mobile_no : 01038641858,  
                    last_pos   : [[ 127.0945116, 37.535397],  
                                [ 126.9815316, 37.5685375],  
                                [ 127.0305035, 37.5017141]]})
```

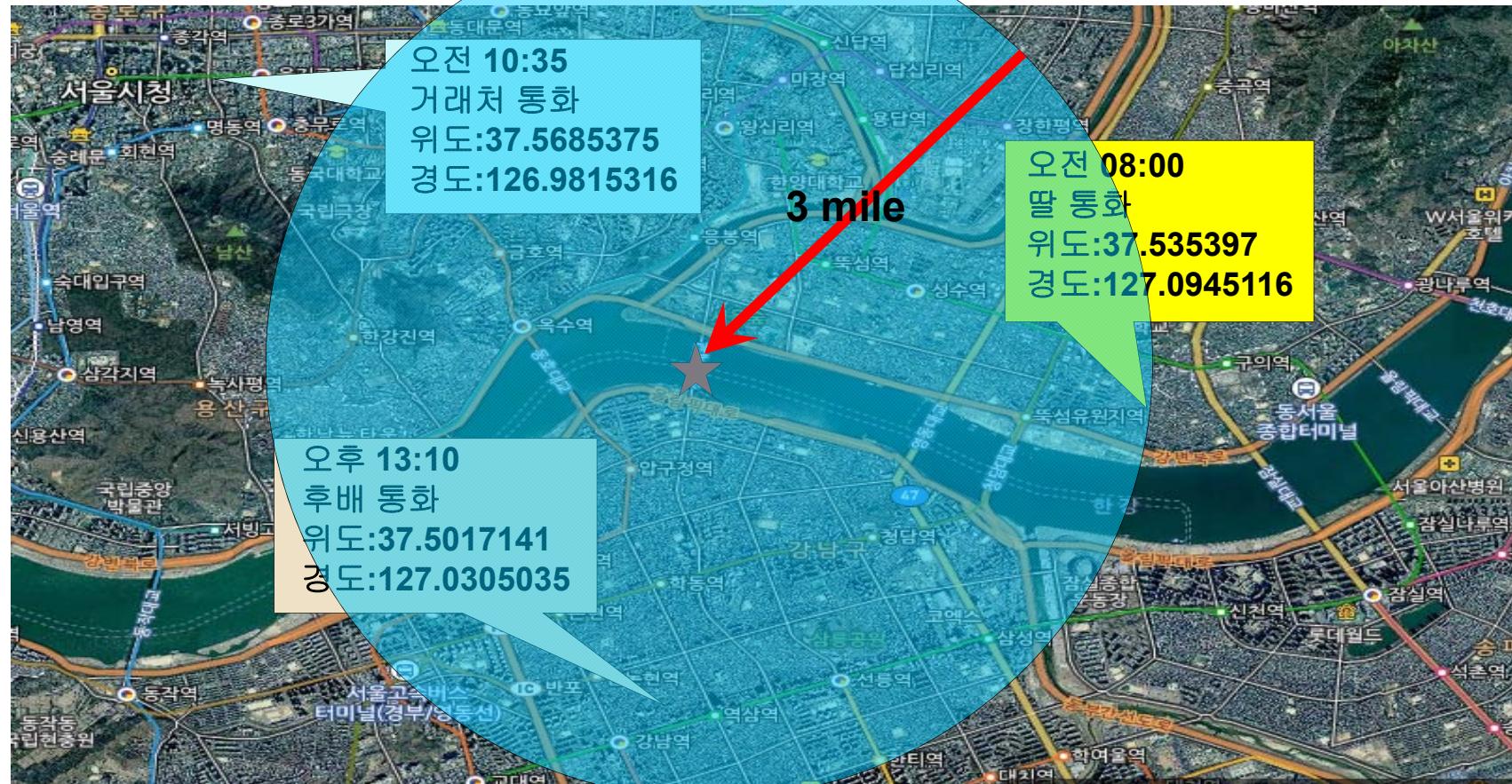
\$nearSphere



```
> db.tel_pos.find( { last_pos : { $nearSphere :  
[[ 127.0352915, 37.5360206]] }}, { _id:0, last_pos : 0 } )
```

← 성수대교를 기준으로 반경 3 Mile 이내

\$centerSphere



```
> db.tel_pos.find( { last_pos : { $within :  
          { $centerSphere : [[ 127.0352915, 37.5360206], 30/3963] }},  
          { _id:0, last_pos : 0 } })  
{ "mobile_no" : "01038641858"}
```

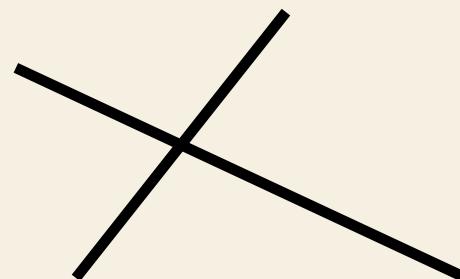
GeoMetry 인덱스

. geoJSON은 직선 또는 곡선의 교차에 의하여 이루어지는 추상적인 구조나 다각형(Polygon)과 같은 기하학(geoMetry) 구조를 일컫는 말이다.

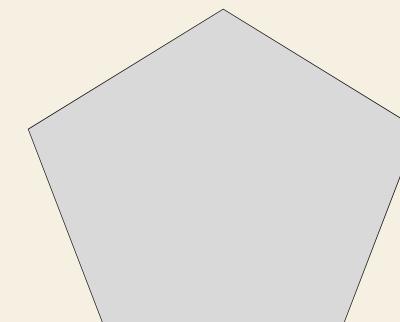
point



LineString



Multi LineString



Polygon

1) Point 탑입

```
db.position.ensureIndex({ loc : "2dsphere"})
db.position.insert( { "_id" : "m91",
"loc" : { "type" : "Point",
"coordinates" : [127.0980748, 37.5301218] },
"name" : [ "name=동서울 터미널" ] })
db.position.insert( { "_id" : "m90",
"loc" : { "type" : "Point",
"coordinates" : [127.0952154, 37.5398467] },
"name" : [ "name=강변역" ] })
db.position.insert( { "_id" : "m89",
"loc" : { "type" : "Point",
"coordinates" : [127.0742172, 37.5419541] },
"name" : [ "name=건대입구역" ] })
```



```
db.position.find( { loc : { $near : { $geometry : { type : "Point" ,
coordinates: [127.1058431, 37.5164113] }}, $maxDistance : 2000 } })
```

8 차시

Document DB 데이터 모델링

MongoDB Data Modeling 핵심



- 1) HOST 환경을 기반으로 했던 파일 시스템은 프로세스 중심의 데이터 구조 설계였다면 클라이언트/서버 환경의 관계형 DB는 트랜잭션의 효율적인 처리를 위한 Data 중심의 설계 기법을 지향하였다.
반면에 클라우드 컴퓨팅 환경의 NoSQL은 Data와 프로세스, 모두를 설계의 중심으로 둔다. (Big Data의 수집 및 저장이 중심)

2) Rich Document Structure

관계형 DB는 정규화를 통해 데이터 중복을 제거하며 무결성을 보장하는 설계 기법을 지향하지만 NoSQL은 데이터의 중복을 허용하며 역정규화된 설계를 지향한다. (관계형 DB가 요구되었던 당시와 달리 저장장치의 비약적 발전과 저렴한 가격 요인도 설계에 중요한 요소임)

- 3) 관계형 DB는 Entity 간의 Relationship을 중심으로 데이터의 무결성을 보장하지만 불필요한 JOIN을 유발시킴으로써 코딩양을 증가시키고 검색 성능을 저하시키는 원인을 제공한다. **NoSQL**은 중첩 데이터 구조를 설계 할 수 있기 때문에 불필요한 JOIN을 최소화 시킬 수 있다.
- 4) 관계형 DB는 Entity 간의 N:M 관계 구조를 설계할 수 없지만 **NoSQL**은 N:M 관계 구조를 설계할 수 있고 구축할 수 있다.
- 5) Document DB는 기본적으로 **Schema**가 없기 때문에 유연한 데이터 구조를 설계할 수 있다.
- 6) 기존의 업무 영역에서 처리할 수 없었던 비정형 데이터에 대한 저장과 관리가 가능하며 관계형 DB에 비해 빠른 쓰기와 읽기가 가능한 데이터 설계가 가능하다.

주문전표

주문번호	2012-09-012345	담당사원	Magee		
고객명	Womansport (주)				
주문날짜	2012-09-20	선적날짜	2012-09-20	선적여부	Y
주문 총금액	601,100	지불방법	현금 30일 이내		

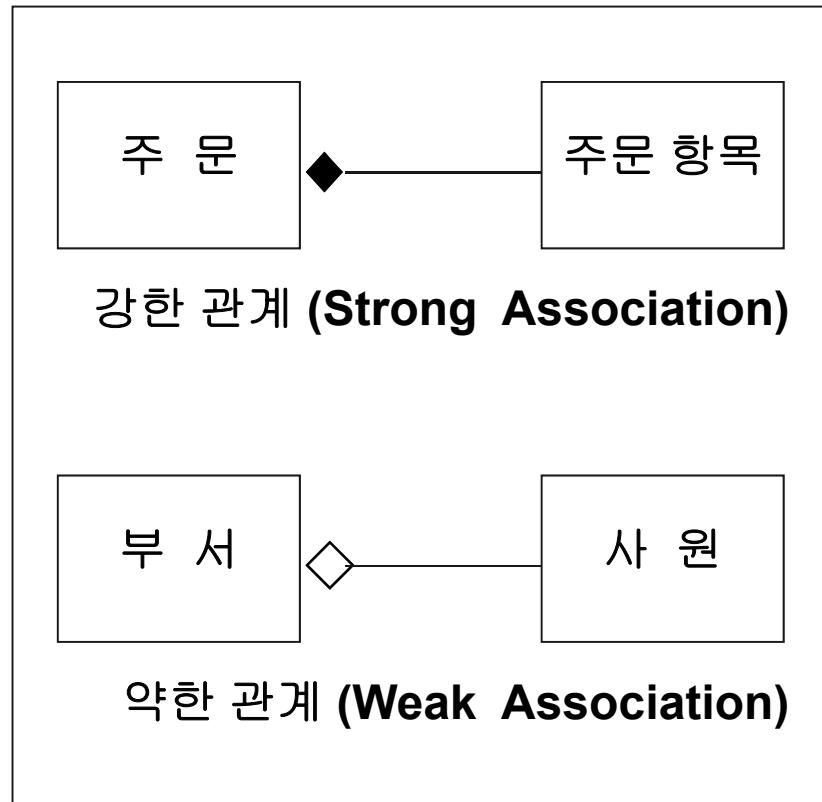
항목번호	제품명	단가	주문수량	금액
1	Bunny Boot	135	500	67,500
2	Pro Ski Boot	380	400	152,000
3	Bunny Ski Pole	14	500	7,000
4	Pro Ski Pole	36	400	14,400
5	Himalaya Bicycle	582	600	349,200
6	New Air Pump	20	450	9,000
7	Prostar 10Pd.Weight	8	250	2,000

SUMMIT2 (주)

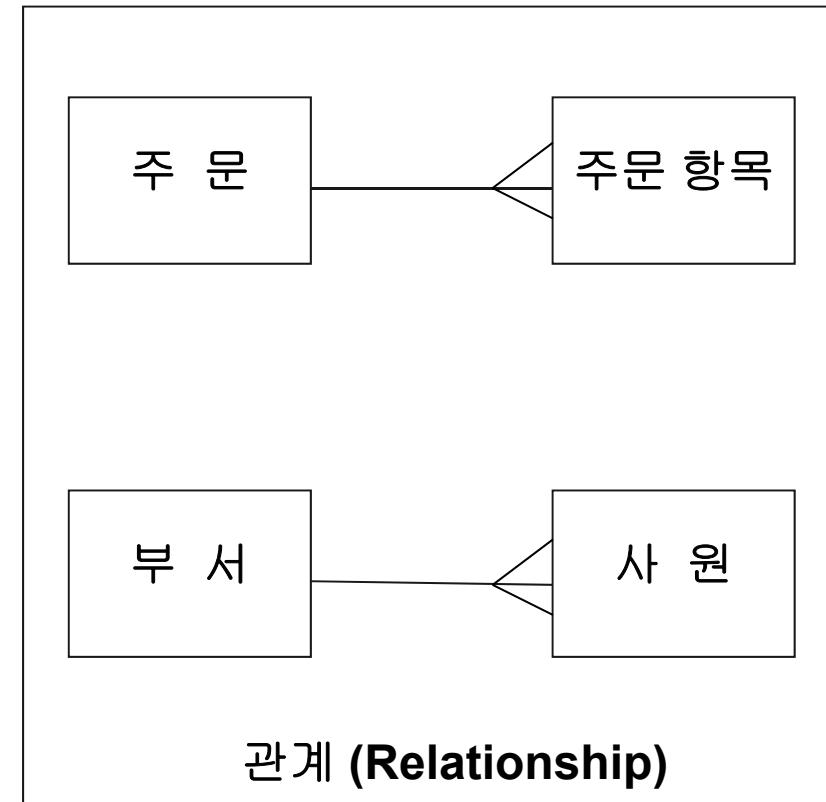
Schema 설계의 주요 특징

1) Embedded(Nested) & Linking 구조

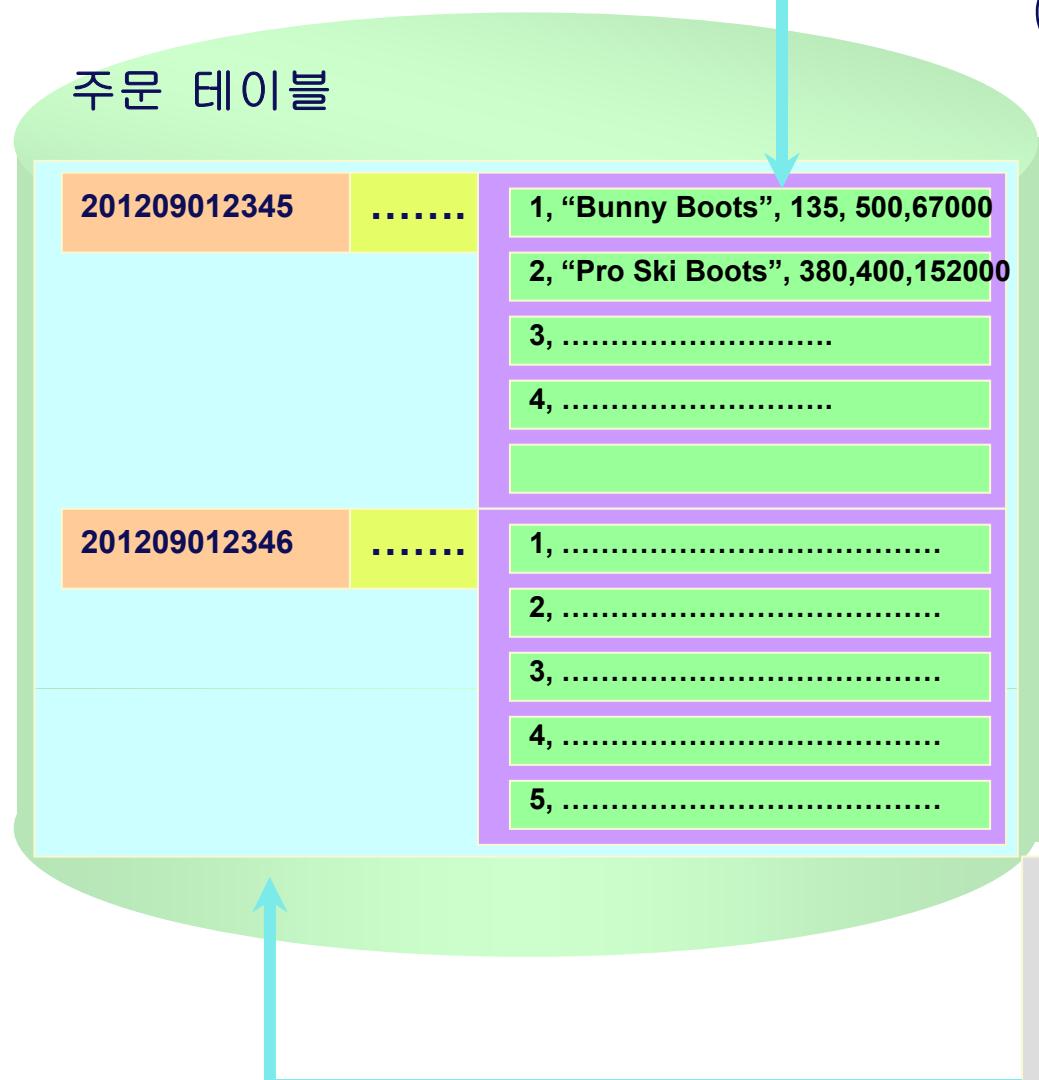
- . 객체 지향 Data 관계유형과 관계형 Data 유형 모두를 설계할 수 있다.



Object Oriented Database



Relationship Database



1

```
Create type product_detail as object
(item_no      number(2),
 p_name       varchar(50),
 s_price      number(8),
 qty          number(5),
 amount       number(10));
```

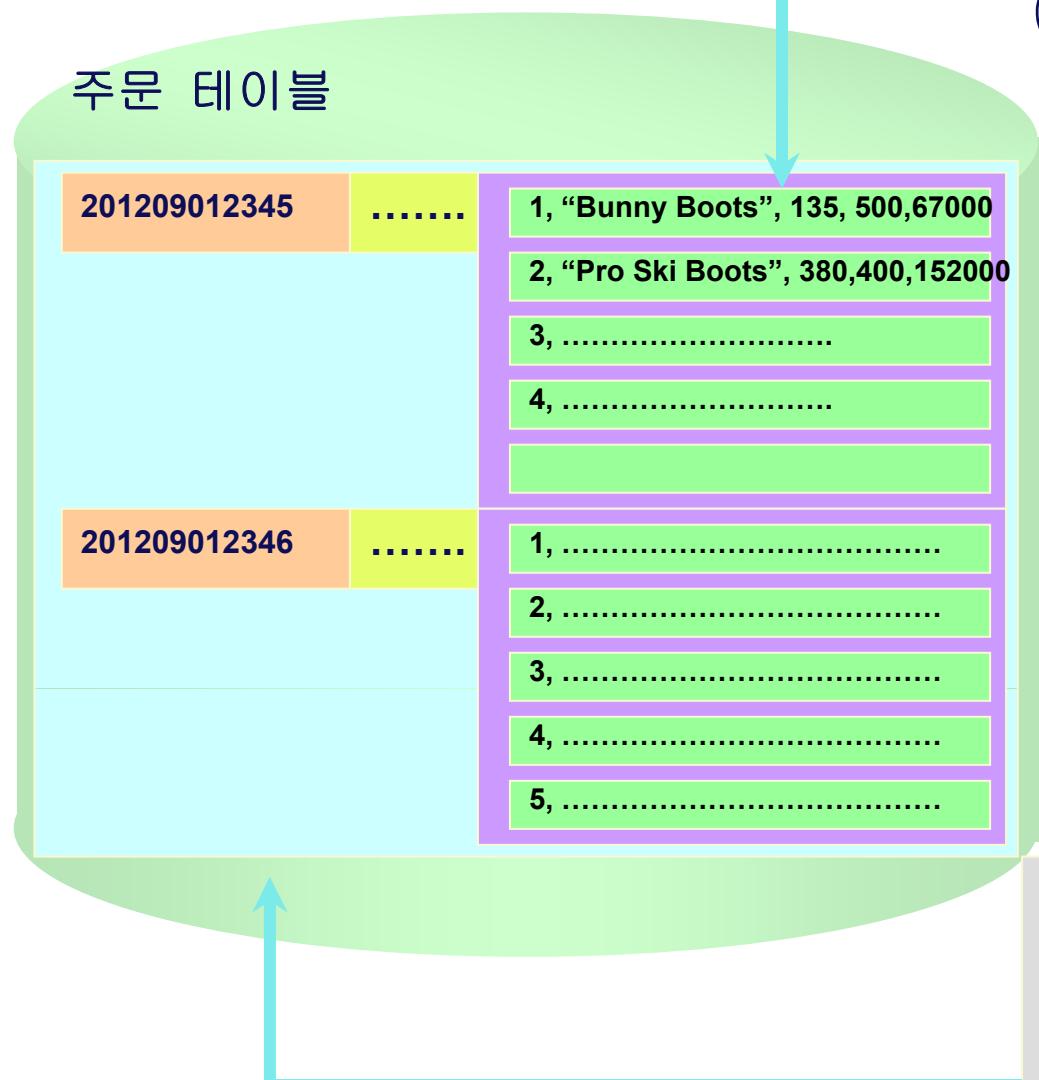
2

```
Create type order_detail
As Table of product_detail;
```

3

```
Create table order
(order_no      char(12),
ename         varchar2(10),
.....,
order_content order_detail)
Nested Table order_content;
```

ORDBMS 데이터 저장 (Varay)



1

```
Create type product_detail as object  
(item_no      number(2),  
 p_name       varchar(50),  
 s_price      number(8),  
 qty          number(5),  
 amount       number(10));
```

2

```
Create type order_detail  
As varray(90) of product_detail;
```

3

```
Create table order  
(order_no      char(12),  
 ename        varchar2(10),  
 .....  
 order_content order_detail)  
VARRAY order_content;
```

MongoDB 데이터 저장 (Embedded)

```
db.ord.insert(
```

```
{ ord_id : "2012-09-012345",
  customer_name : "Wonman & Sports",
  emp_name : "Magee",
  total : "601100",
  payment_type : "Credit",
  order_filled : "Y",
```

주문 공통 정보

```
  item_id : [ { item_id : "1",
    product_name : "Bunny Boots",
    item_price : "135",
    qty : "500",
    price : "67000" },
    { item_id : "2",
    product_name : "Pro Ski Boots",
    item_price : "380",
    qty : "400",
    price : "152000" } ]
```

주문 상세 정보

```
) )
```

MongoDB 데이터 저장 (Manual Link)

```
> db.ord.insert( { ord_id : "2012-09-012345",
   customer_name : "Wonman & Sports",
   emp_name : "Magee",
   total : "601100",
   payment_type : "Credit",
   order_filled : "Y" } )
```

주문 공통 정보

```
> o = db.ord.findone( { "ord_id" : "2012-09-012345" } )
{ "_id" : ObjectId("4fc21223e6cd4d2aadb38622"), ..... }
```

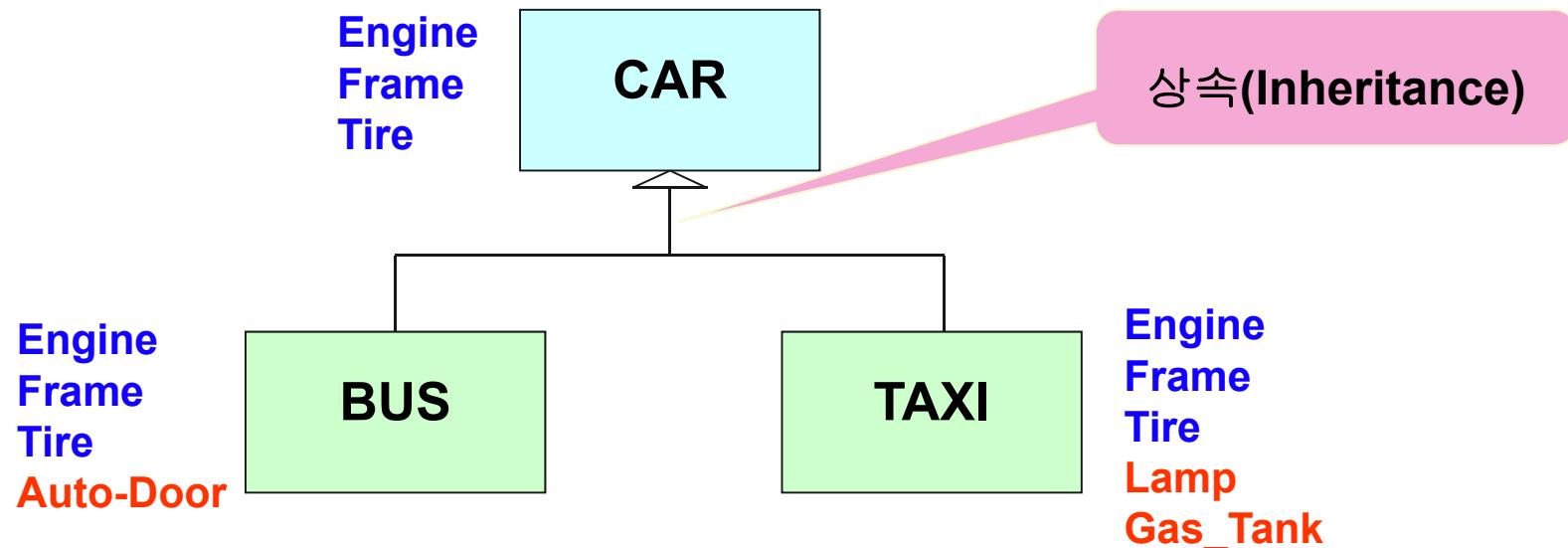
```
> db.ord_detail.insert( { ord_id : "2012-09-012345",
   item_id : [ { item_id : "1",
     product_name : "Bunny Boots",
     item_price : "135",
     qty : "500",
     price : "67000" },
    { item_id : "2",
     product_name : "Pro Ski Boots",
     item_price : "380",
     qty : "400",
     price : "152000" }
   ],
   ordid_id : ObjectId("4fc21223e6cd4d2aadb38622" } )
```

주문 상세 정보

```
> db.ord_detail.findOne({ordid_id : o._id})
```

Demonstration

3) Inheritance (OODBMS)



```
CREATE TYPE car AS OBJECT  
(engine      NUMBER(9)      Primary Key,  
 frame       VARCHAR(30),  
 tire        VARCHAR(30)) NOT FINAL;
```

```
CREATE TYPE bus UNDER car_typ  
(auto_door   VARCHAR(30)  FINAL;
```

```
CREATE TYPE taxi UNDER car_typ  
(lamp        VARCHAR(30),  
 gas_tank    VARCHAR(30)  FINAL;
```

Single Table Inheritance (RDBMS)

CAR
Engine
Frame
Tire
Car_Type
BUS
Auto_door
TAXI
Lamp
Gas_tank

```
CREATE TABLE car
(engine INTEGER(9) NOT NULL,
frame CHAR VARYING(30) NOT NULL,
tire CHAR VARYING(30) NOT NULL,
car_type CHAR VARYING(4) CHECK IN(' BUS' , 'TAXI' ),
auto_door INTEGER(2),
lamp CHAR VARYING(30),
gas_tank CHAR VARYING(30)
Constraint bus_pk PRIMARY KEY (engine, frame, tire);
```

Engine	Frame	Tire	Car_Type	Auto_Door	Lamp	Gas_Tank
A	AX_1	R16	TAXI		1	1
B	AK_3	R18	BUS	2		
A	AX_2	R18	TAXI		2	2

Single Table Inheritance (MongoDB)

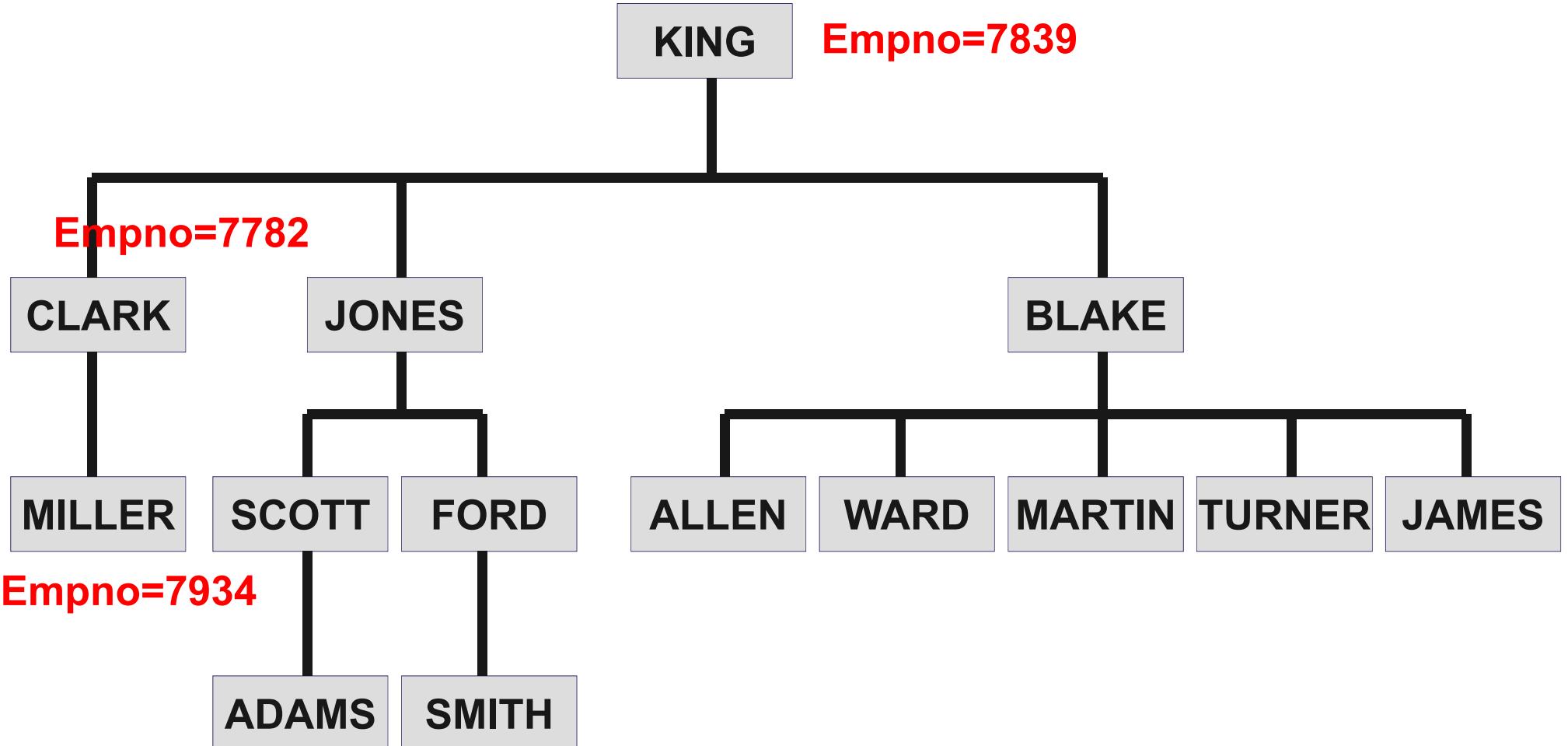
```
> db.createCollection ("car");
> db.car.insert({ engine : "A", frame : "AX_1", tire : "R16",
                  car_type : "TAXI", lamp : 1, gas_tank : 1 });
> db.car.insert({ engine : "B", frame : "AK_3", tire : "R18",
                  car_type : "BUS", auto_door: 2 });
> db.car.insert({ engine : "A", frame : "AX_2", tire : "R18",
                  car_type : "TAXI", lamp : 2, gas_tank : 2 });

> db.employees.find();
{"_id" : ObjectId("4f00574f81a153d6857897d2"),
 "engine" : "A", "ename" : "AX_1", "tire" : "R16",
 "car_type" : "TAXI", "lamp" : 1, "gas_tank" : 1 }; }
```

Engine: A	Frame: AX_1	Tire: R16	Car_type: TAXI	Lamp: 1	Gas_tank: 1
Engine: B	Frame: AK_3	Tire: R18	Car_type: BUS	Auto_door: 1	
Engine: A	Frame: AX_1	Tire: R18	Car_type: TAXI	Lamp: 2	Gas_tank: 2

Demonstration

4) 계층형 데이터 구조



Self Reference Join (RDBMS)

Empno	ename	mgr	b.Ename
7839	KING		
7698	BLAKE	7839	KING
7782	CLARK	7839	KING
7566	JONES	7839	KING
7654	MARTIN	7698	BLAKE
7902	FORD	7566	JONES
7876	ADAMS	7788	JIMMY
7934	MILLER	7782	CLARK

```
SELECT a.empno, a.ename, a.mgr, b.ename  
FROM emp a, emp b  
WHERE a.mgr = b.empno
```

Ancestor Reference (MongoDB)

7839 : KING

7782 : CLARK

7934 : MILLER

```
> db.emp.insert({ "_id" : "7839", "name" : "KING",    "job" : "PRESIDENT" })
> db.emp.insert({ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",
                  "PARENT"      : "7839" } )
> db.emp.insert({ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",
                  "ANCESTORS"  : "7839",
                  "PARENT"      : "7782" } )

> db.emp.find({"ANCESTORS" : "7839"})
{ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",
  "ANCESTORS" : [ "7839", "7782" ], "PARENT" : "7782" }

> db.emp.find({"PARENT" : "7839"})
{ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",
  "PARENT" : "7839" }
```

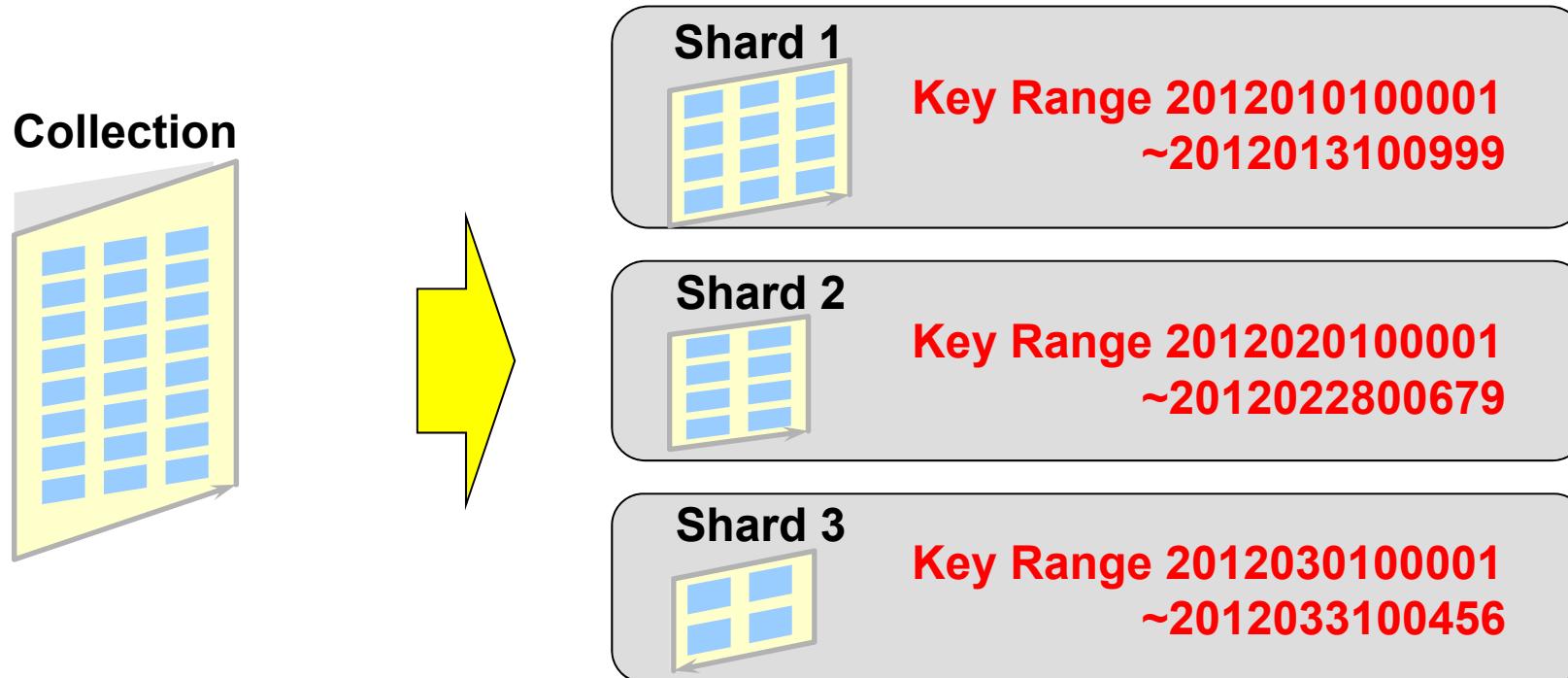
- 1) 기업에서 발생하는 데이터 구조 중에는 계층형 데이터 구조가 발생할 수 밖에 없는데 이런 경우 적용하면 가장 이상적인 데이터 모델이다.
- 2) ANCESTORS와 PARENT Field로 표현할 수 있으며 하나의 Document는 하나 이상의 ANCESTORS 와 PARENT를 가질 필요는 없다.

Demonstration

9 차시

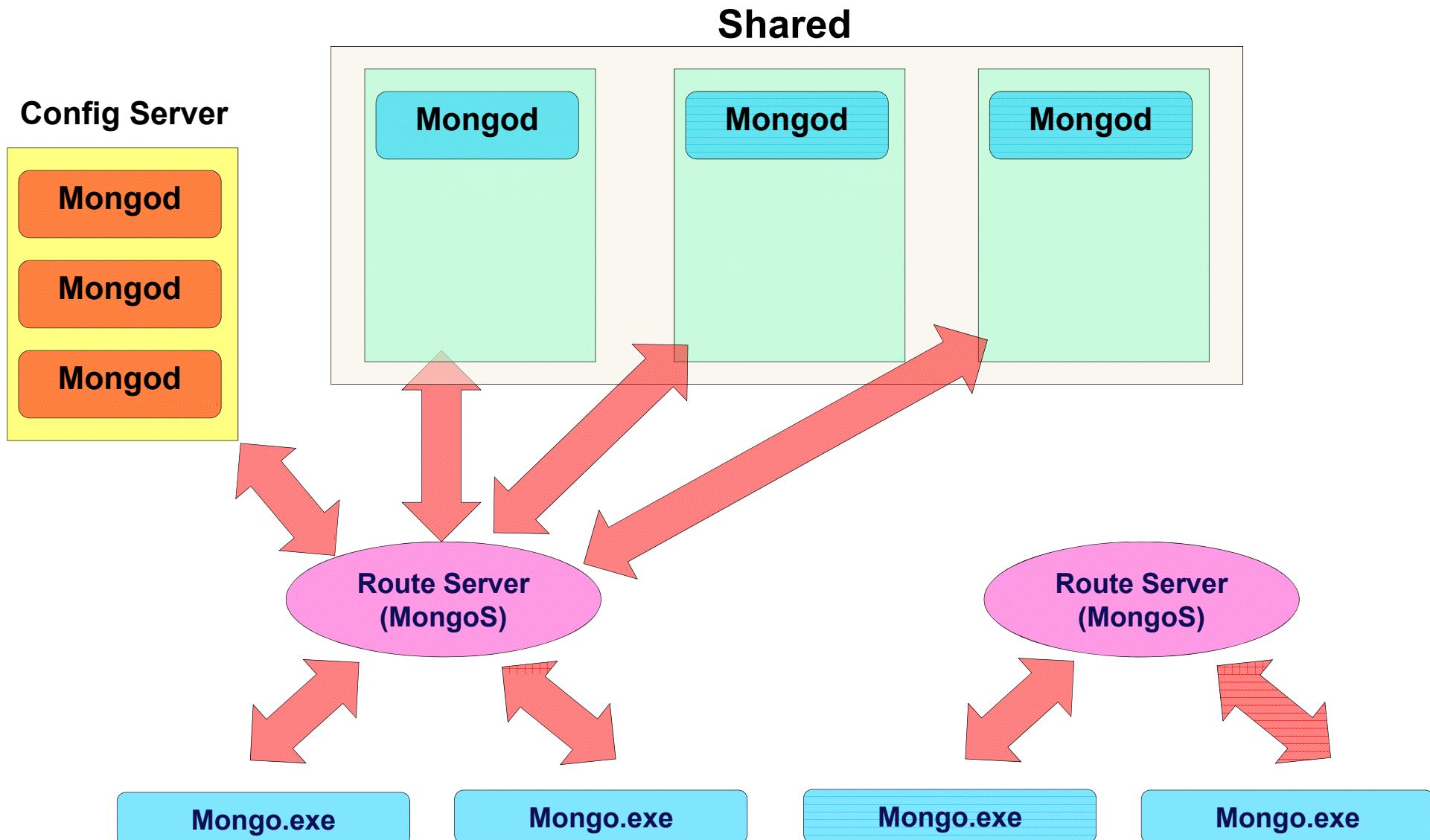
데이터 분산 처리

Sharding (Partition)

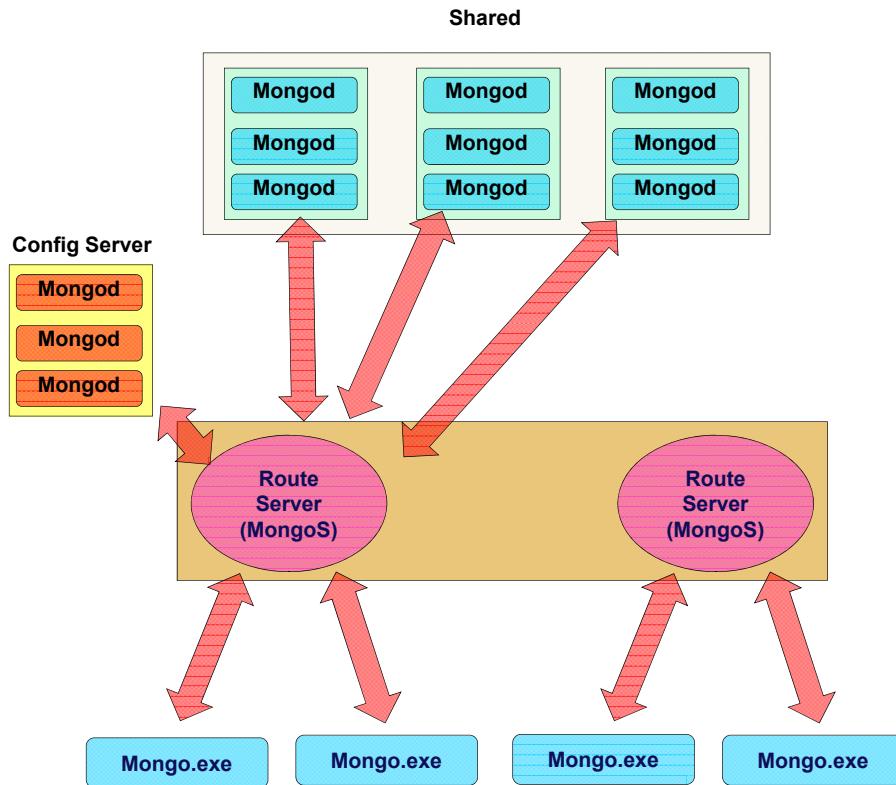


- 1) Sharding의 가장 큰 목적은 파티셔닝을 통한 데이터 분산 처리와 성능 향상을 위한 Load Balancing이다.
- 2) 또한, 빅 데이터의 효율적 관리와 백업 및 복구 전략 수립을 위한 솔루션이기도 하다.

MongoDB Architecture (Multi Node)

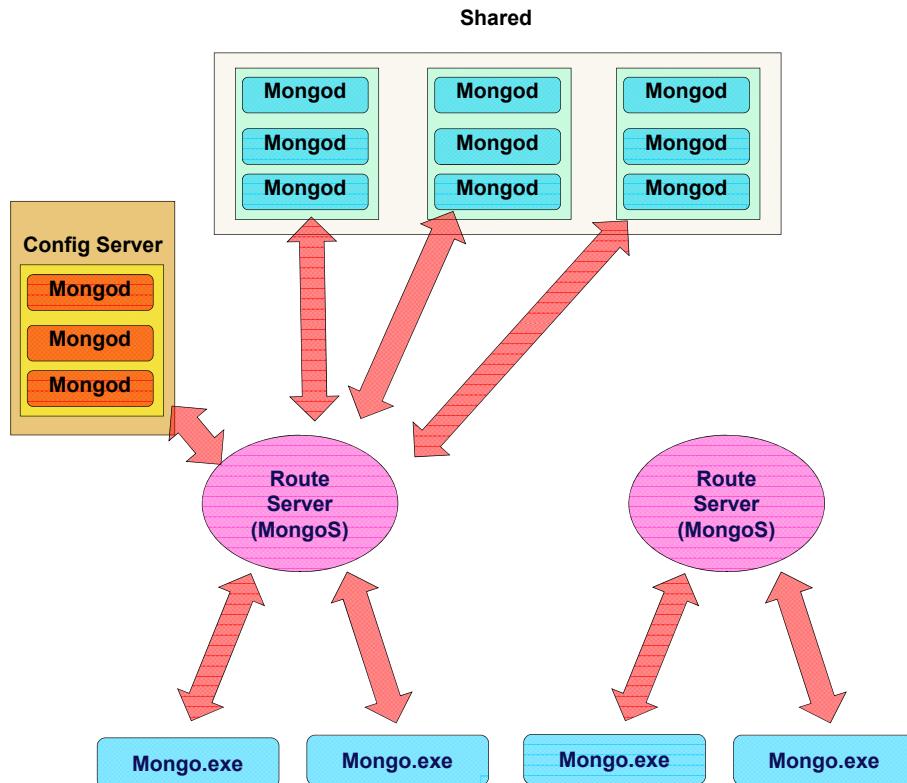


Mongos 프로세스



- 1) Shard로 분배해 주는 프로세스
- 2) 하나 이상의 프로세스가 활성화됨
- 3) Application Server에서 실행 가능 하다.
- 4) Config Server로부터 Meta-Data를 Cache한다,

Config Server



- 1) 3개의 **mongod**로 구성된다,
- 2) 2 Phase Commit을 수행한다.
- 3) 만약, 하나의 **Config Server**가 다운되면 나머지는 **Read Only**된다.
- 4) **Shard**에 대한 **Meta Data** 정보를 가지고 있다.

Shard 환경 설정

1) Node 1을 위한 Shard Server를 활성화 합니다.

```
C:\> mongod --shardsvr --dbpath c:\mongodb\shard1 --port 40001
```

2) Node 2를 위한 Shard Server를 활성화 합니다.

```
C:\> mongod --shardsvr --dbpath c:\mongodb\shard2 --port 40002
```

3) Node 3를 위한 Shard Server를 활성화 합니다.

```
C:\> mongod --shardsvr --dbpath c:\mongodb\shard3 --port 40003
```

4) Node 1과 Node2, Node3를 위한 Config Server를 활성화 합니다.

```
C:\> mongod --configsvr --dbpath c:\mongodb\config1 --port 50001
```

Shard 환경 설정

1) MongoS 프로세스를 활성화 한다.

```
C:\> mongos --configdb 192.168.0.1:10000,192.168.0.2:10001,192.168.0.3:10002
```

2) Config Server가 설치된 Node에서 각 Node가 상호 연결될 수 있도록 등록한다.

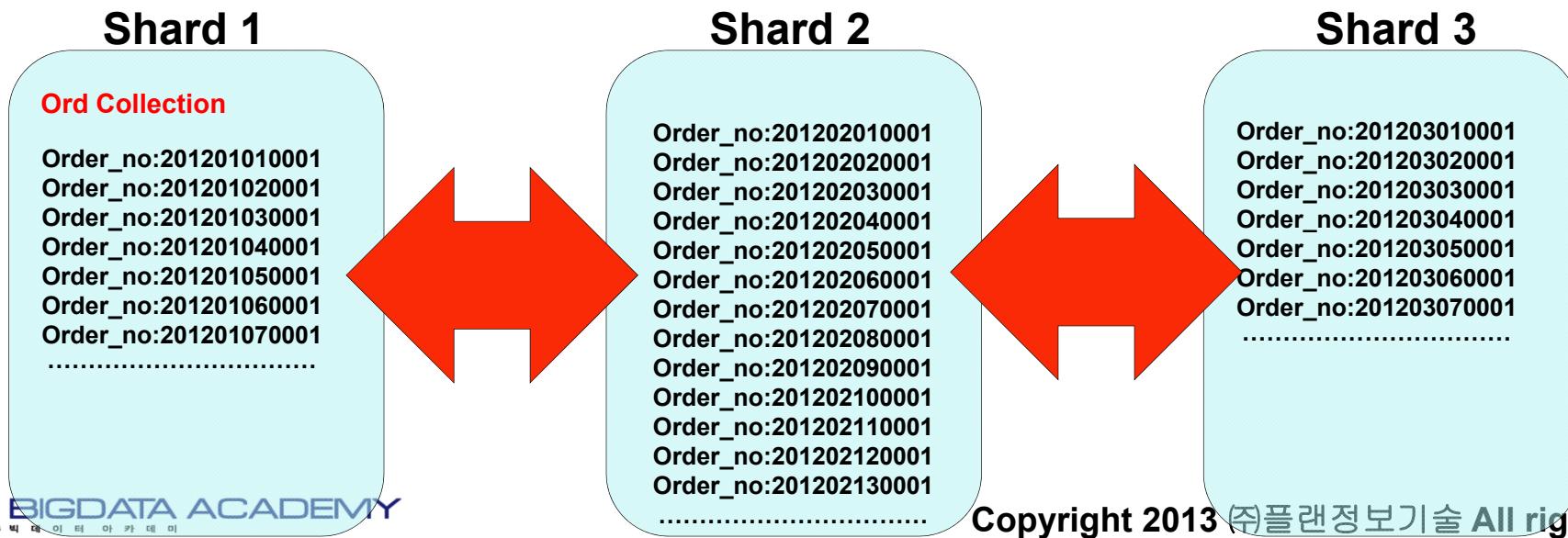
```
C:\> mongo localhost:27017/admin ← mongos에 접속하여 Node 정보 등록
mongos>
mongos> db.runCommand( { addshard : "192.168.0.10 : 10000" } ); ← Node1 등록
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos>
mongos> db.runCommand( { addshard : "192.168.0.10 : 10001" } ); ← Node2 등록
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos>
mongos> db.runCommand( { addshard : "192.168.0.10 : 10002" } ); ← Node3 등록
{ "shardAdded" : "shard0002", "ok" : 1 }
mongos>
mongos> db.runCommand( { enablesharding : "test" } ); ← 해당 DB Shard 기능 설정
{ "ok" : 1 }

> db.runCommand( { shardcollection : "test.things", key : { _id : 1 } } ); ← Shard Key
{ "collectionsharded" : "test.things", "ok" : 1 }
```

Demonstration

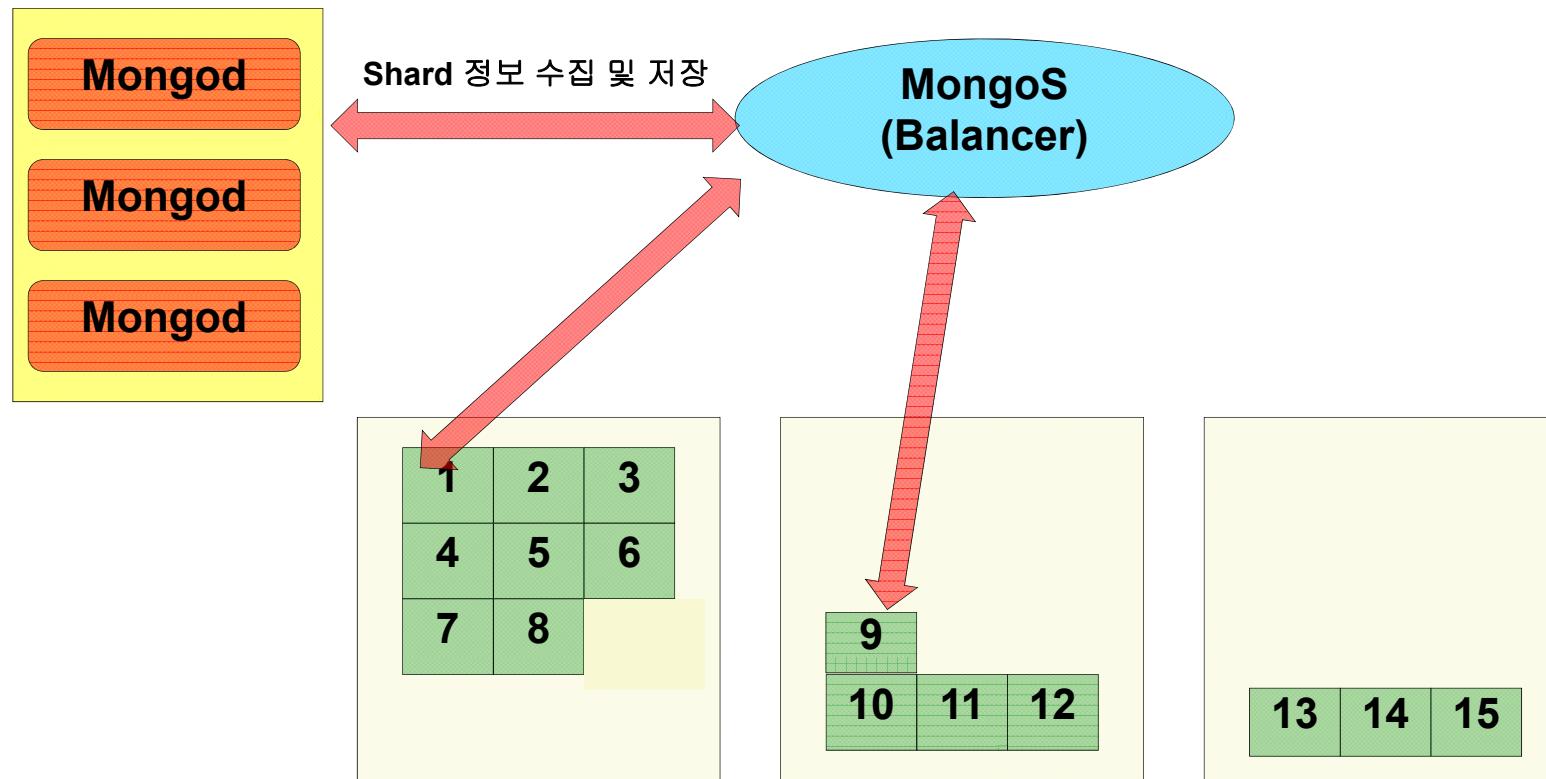
Shard System의 문제점

- 1) 하나의 **Shard** 서버에 데이터가 집중되고 **Load Balancing**이 되지 않는 경우
- 2) 특정 **Shard** 서버에 IO 트래픽이 증가하는 경우
- 3) 샤드 서버 클러스터의 밸런스가 균등하지 않는 경우
- 4) 과도한 **Chunk Migration**이 클러스터 작동을 멈추는 경우



Chunk Migration

Config Server



- 1) MongoS는 각 Shard의 Balance에 불균형(8개 이상의 Chunk 개수)이 발생하면 Chunk의 Migration 작업을 수행한다.
- 2) Migration이 발생할 때 Mongos에 Lock이 발생한다.
- 3) 실제로는 Move가 아니라 데이터를 복사하고 순서로 Chunk 데이터를 삽입된다.
Copyright 2013 BIGDATA Korea All rights reserved.

Shard Key

```
> db.runCommand( { shardcollection : "sales.order",  
                   key : { idate : 1 } })
```

- 1) **Shard Key**는 여러 개의 **Shard Server**로 분할될 기준 **Field** 이므로 충분한 고려를 해야 한다.
- 2) **Shard Key**는 **Partition**과 **Load Balancing**의 기준이 되므로 MongoDB의 데이터 저장과 성능에 절대적 영향을 미친다.
- 3) **Shard Key**를 선정하는 것은 **Cardinality**, 데이터의 분산 저장, **Query Isolation**, 신뢰도, **Index Locality** 등을 고려하여 결정해야 한다.
- 4) 하나 이상의 **Field**로 **Shard Key**를 구성할 수 있다.
(최대 512 Byte를 초과할 수 없다.)

Hashed Index

```
> db.employees.ensureIndex( { deptno: "hashed" } )
```

- 1) Hashed Index는 MongoDB V2.4에서 제공되었다.
- 2) 사용자에 의해 정의된 **Shard Key**가 적절하게 분산되지 않는 문제 해결을 위해 MongoDB에서 제공하는 해시 알고리즘을 적용할 수 있다.
- 3) 인덱스 대상 필드의 값이 해시 값으로 변환되어 저장되기 때문에 불필요한 변환이 발생함으로 사용자가 해당 기술에 대한 정확한 이해를 바탕으로 사용하되 적절한 **Shard Key**가 없거나 기술에 대한 이해가 부족한 경우 적용하는 것이 좋다.
- 4) Compound Index, Unique Index, Multi-Key Index에는 적용 할 수 없다

Shard Server의 추가와 삭제

1) Shard Server의 추가

```
> sh.addShard( "<hostname>:<port>" )
```

또는

```
> db.runCommand( { addshard : "hostname:<port>" } );
```

2) Shard Server의 삭제

```
> db.runCommand( { removeshard: "<Shard Server명" } )
```

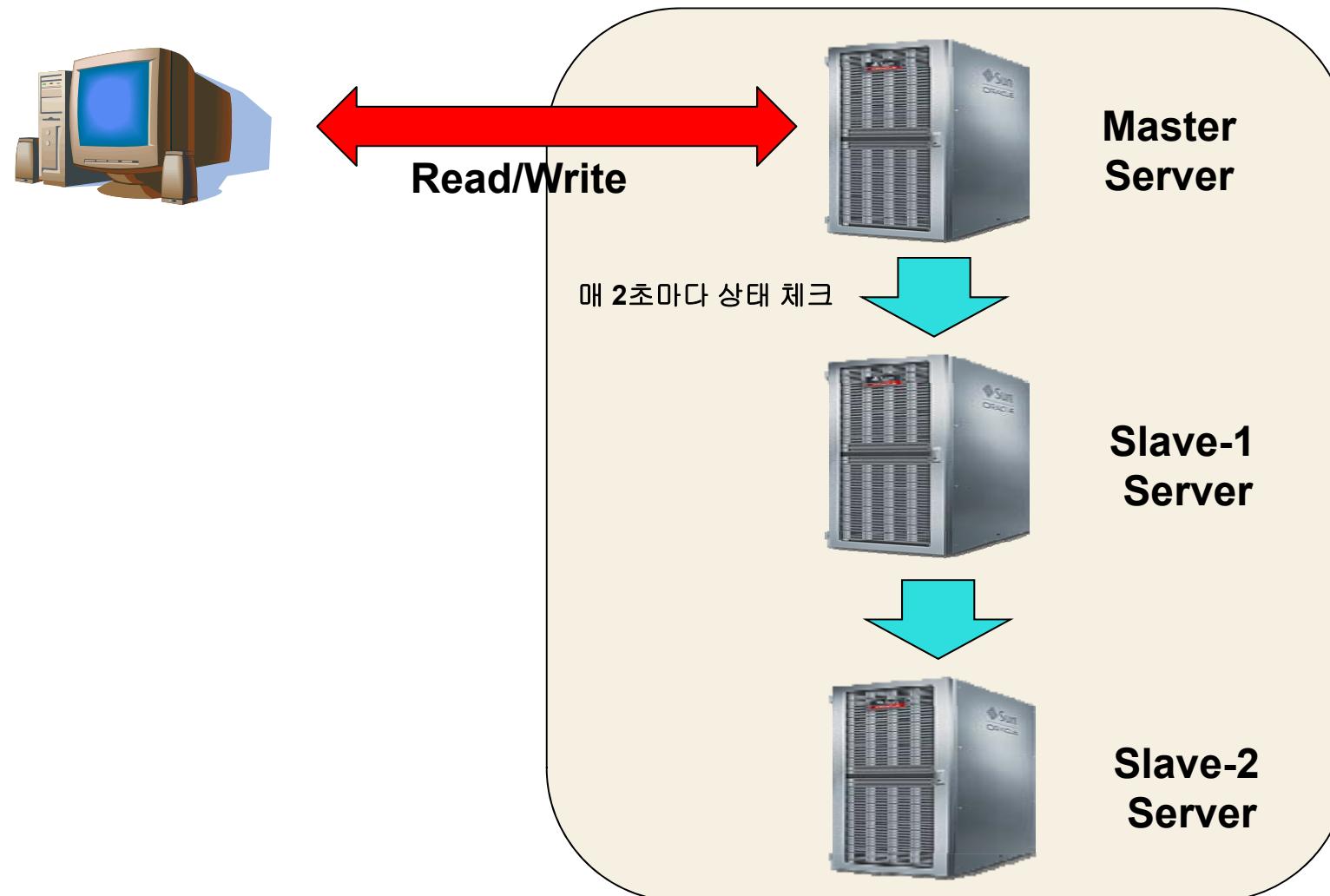
3) Shard Server의 이동

```
> db.runCommand(  
  { movePrimary: "<대상 Shard>, to: "<이동될 Shard>" })
```

10 차시

데이터 복제

Master & Slave



환경 설정

1) Node 1을 Master DB로 설정한다.

```
C:\> mongod --dbpath C:\MONGODB\MASTER\ --port 10000 --master
```

2) Node 2를 Slave-1 DB로 설정한다.

```
C:\> mongod --dbpath C:\MONGODB\SLAVE1\  
--port 10001 --slave --source localhost:10000
```

3) Node 3를 Slave-2 DB로 설정한다.

```
C:\> mongod --dbpath C:\MONGODB\SLAVE2\  
--port 10002 --slave --source localhost:10000
```

- 1) Replica 기능은 기본적으로 최소 2개 이상의 Node로 구성되어야 한다.
- 2) Master DB와 Slave DB는 별도의 Node에 구축해야 한다.
- 3) 하나의 Node에 Shard Server와 Config Server를 구축하는 경우에는 반드시 별도의 PORT 번호를 지정해야 한다.

환경 설정

1) Primary DB에서 입력된 데이터가 Secondary DB에 복제될 수 있도록 데이터를 입력한다.

```
C:\> mongo localhost:10000
> use test
> db.things.insert( { empno : 1101,ename : "james",dept : "account" } );
> db.things.find();

{"_id":ObjectId("4f03b6c6e5c6022a325f7181"),
 "empno":1101, "ename":"james", "dept":"account"}
```

2) Secondary DB에서 복제된 데이터를 확인한다.

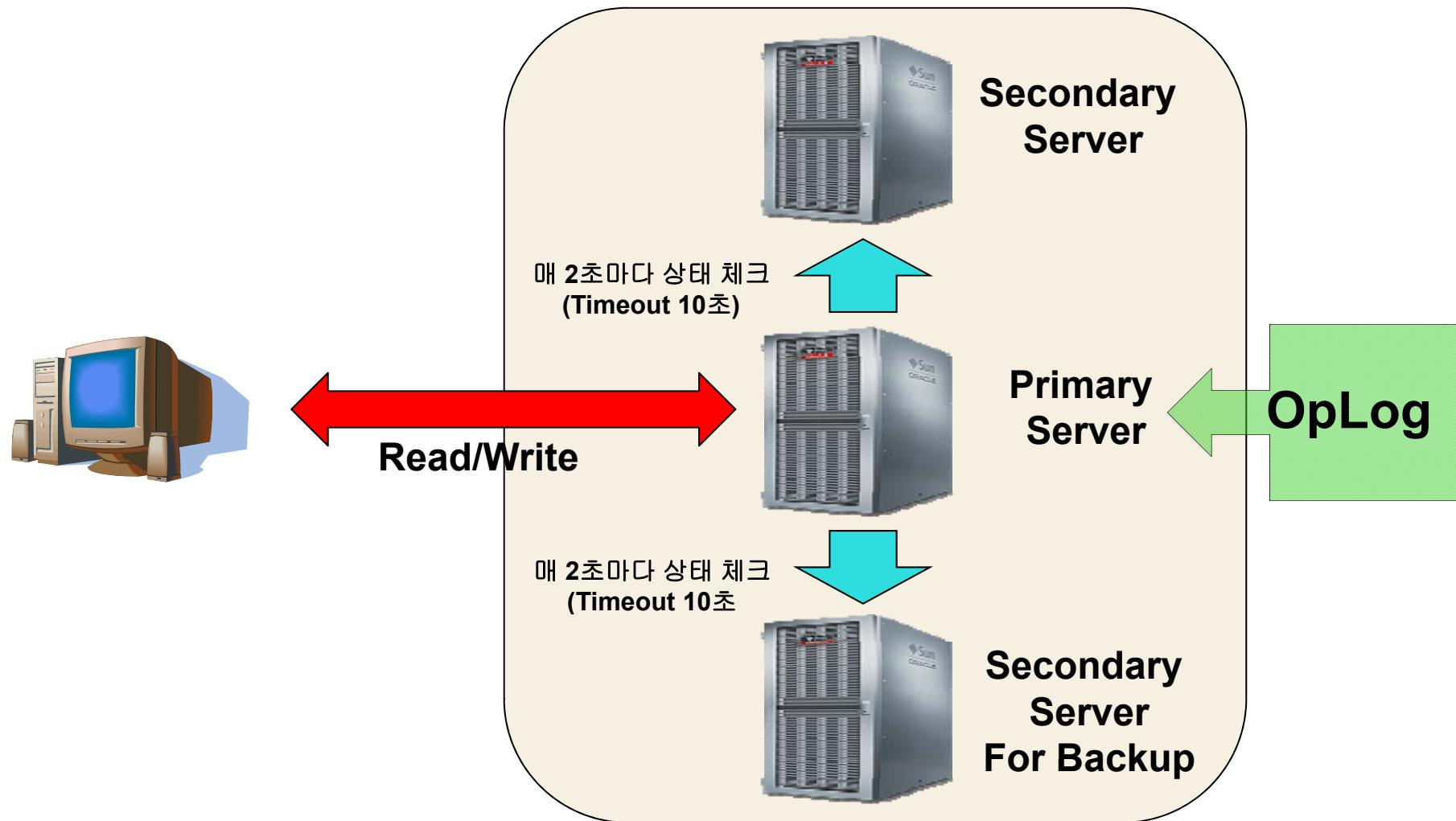
```
C:\> mongo localhost:10001
> use test
> db.things.find();      ← Primary DB에서 생성된 Collection과 Document

{"_id":ObjectId("4f03b6c6e5c6022a325f7181"),
 "empno":1101, "ename":"james", "dept":"account"}
```

Demonstration

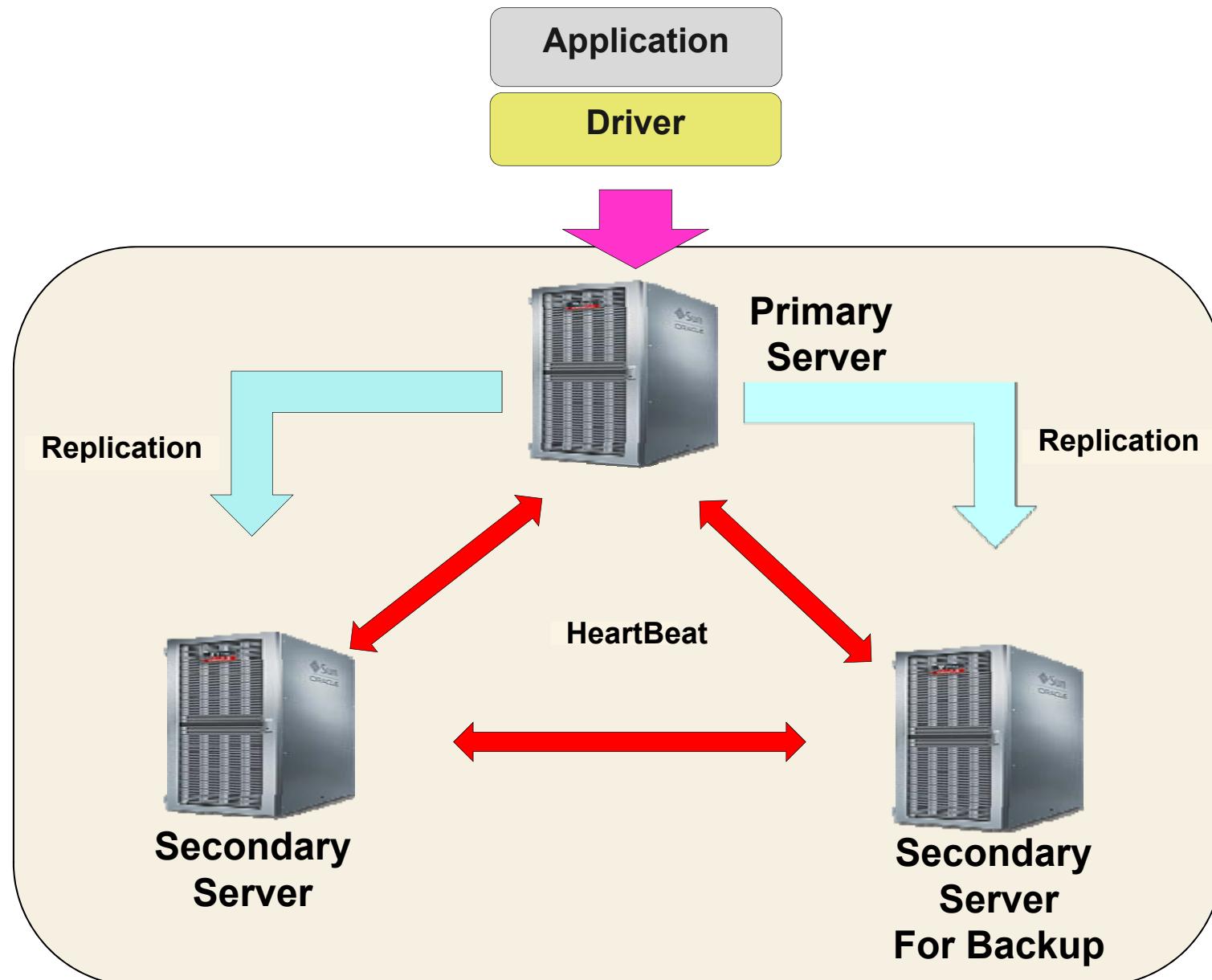
ReplicaSets 기능

Replica Sets



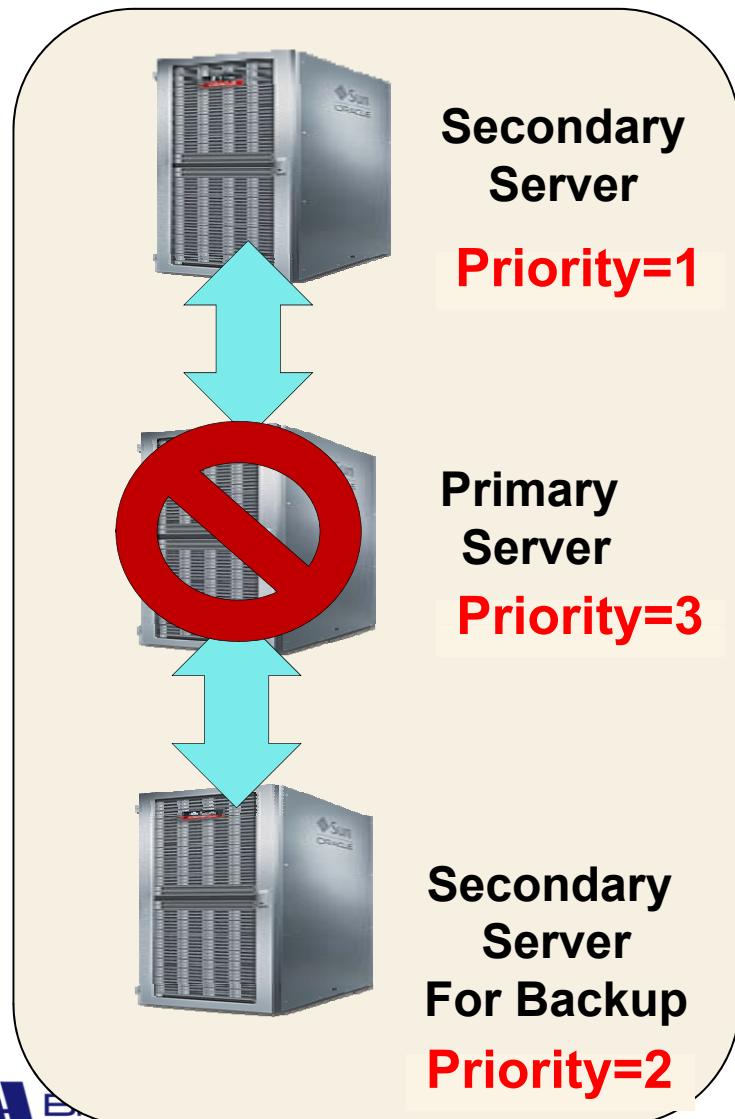
- 1) Heartbeat : 매 2초마다 **Secondary** 서버 상태를 체크한다.
- 2) **Secondary** 서버가 **Down** 되더라도 복제 만 중지될 뿐 **Primary** 서버에 대한 작업은 정상적이다.
- 3) **Primary** 서버가 다운되면 **Secondary** 서버 중 하나가 **Primary** 서버가 된다.

초기화 상태

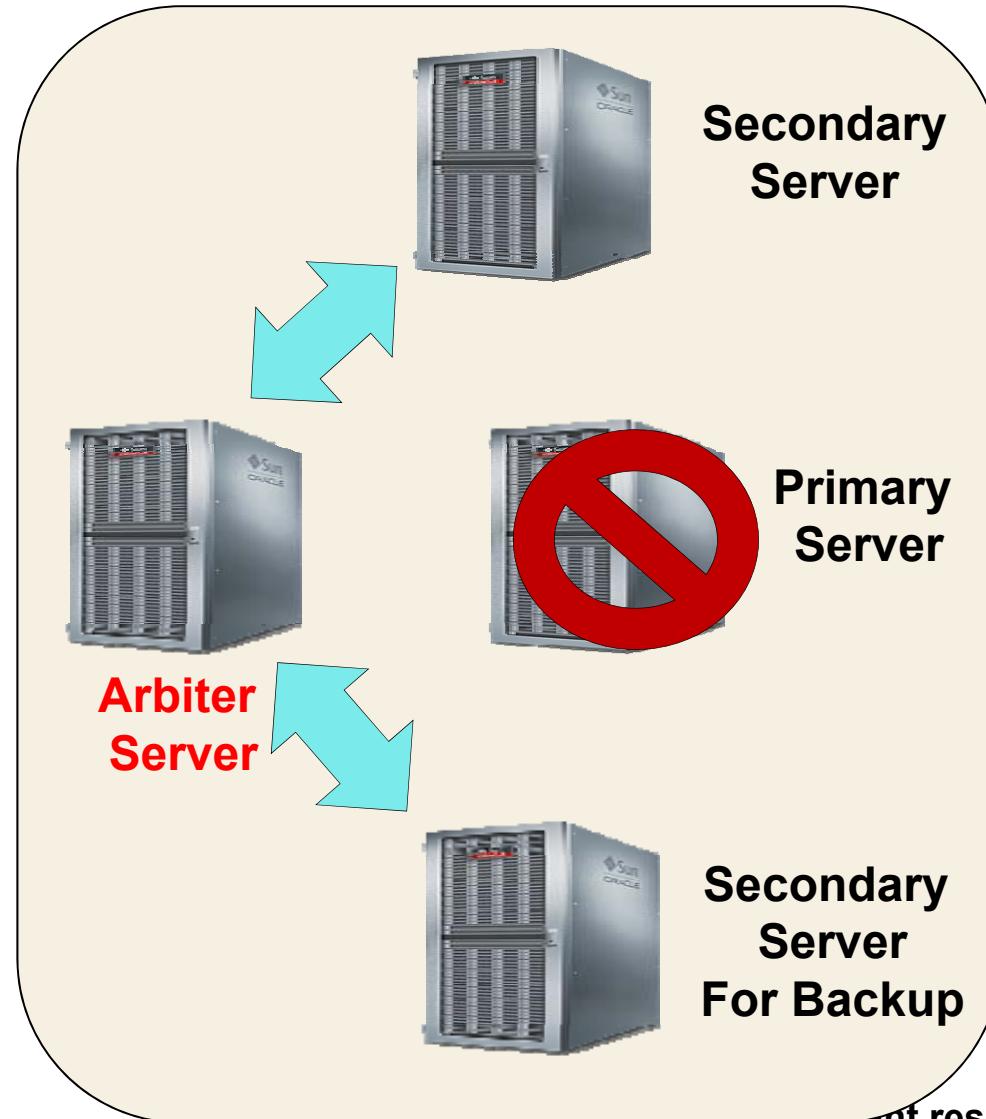


Primary 서버 선출 방법

1) Priority에 의한 선출



2) Arbiter 서버에 의한 전자 선출



Demonstration

11 차시

Hbase & 데이터 처리

NoSQL 제품군

1. Key-Value Database

- 1) Amazon's Dynamo Paper
- 2) Data Model : Collection of K-V pairs
- 3) 제품유형 : Riak, Voldemort, Tokyo*

3. Document Database

- 1) Lotus Notes
- 2) Data Model : Collection of K-V collection
- 3) 제품유형 : Mongo DB, Cough DB

2. BigTable Database

- 1) Google's BigTable paper
- 2) Data Model : Column Families
- 3) 제품유형 : Hbase, Casandra, Hypertable

4. Graph Database

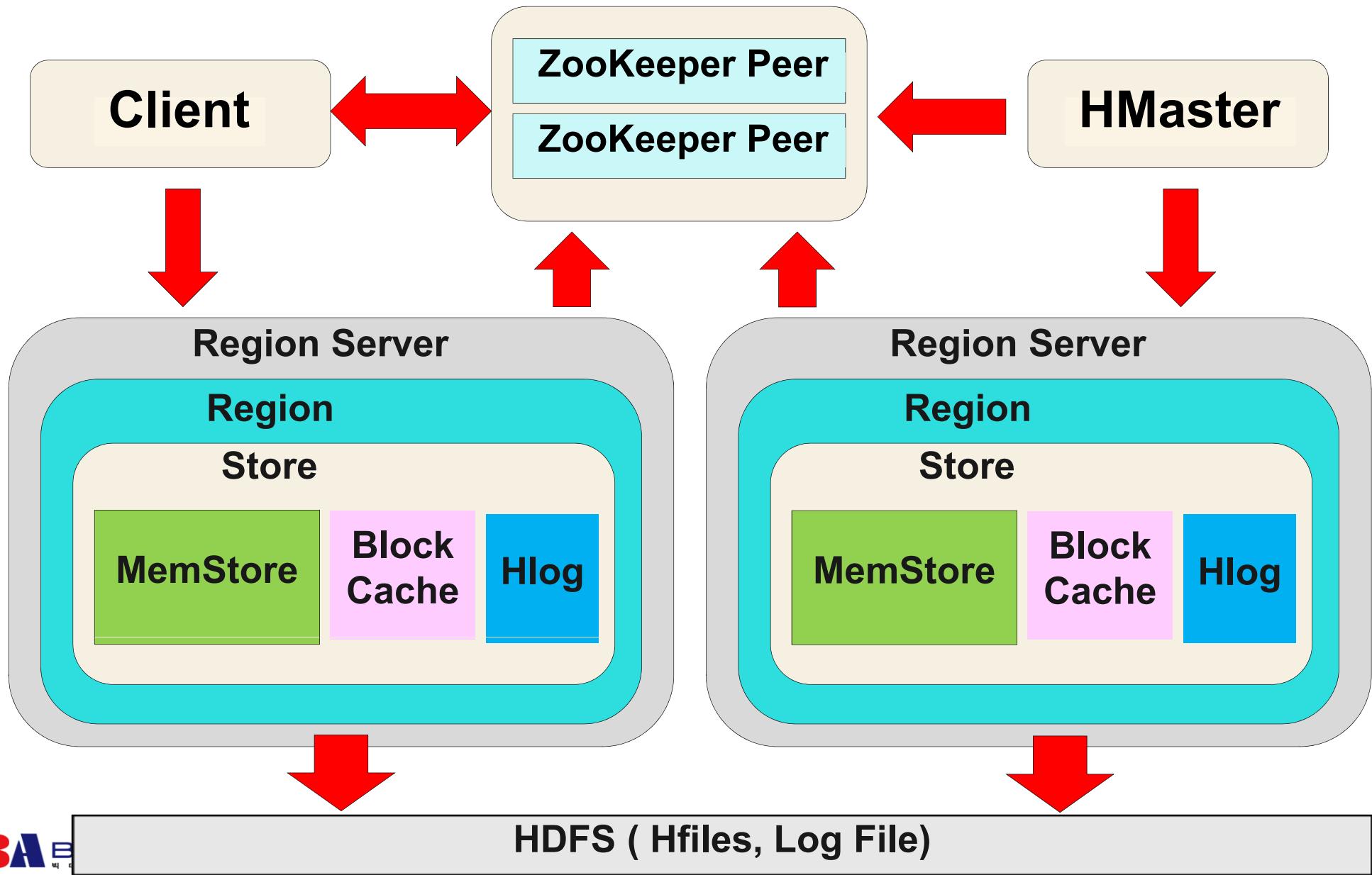
- 1) Euler & Graph Theory
- 2) Data Model : nodes, rels, K-V on both
- 3) 제품유형 : Neo4J, Sones

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

HBase 주요 특징

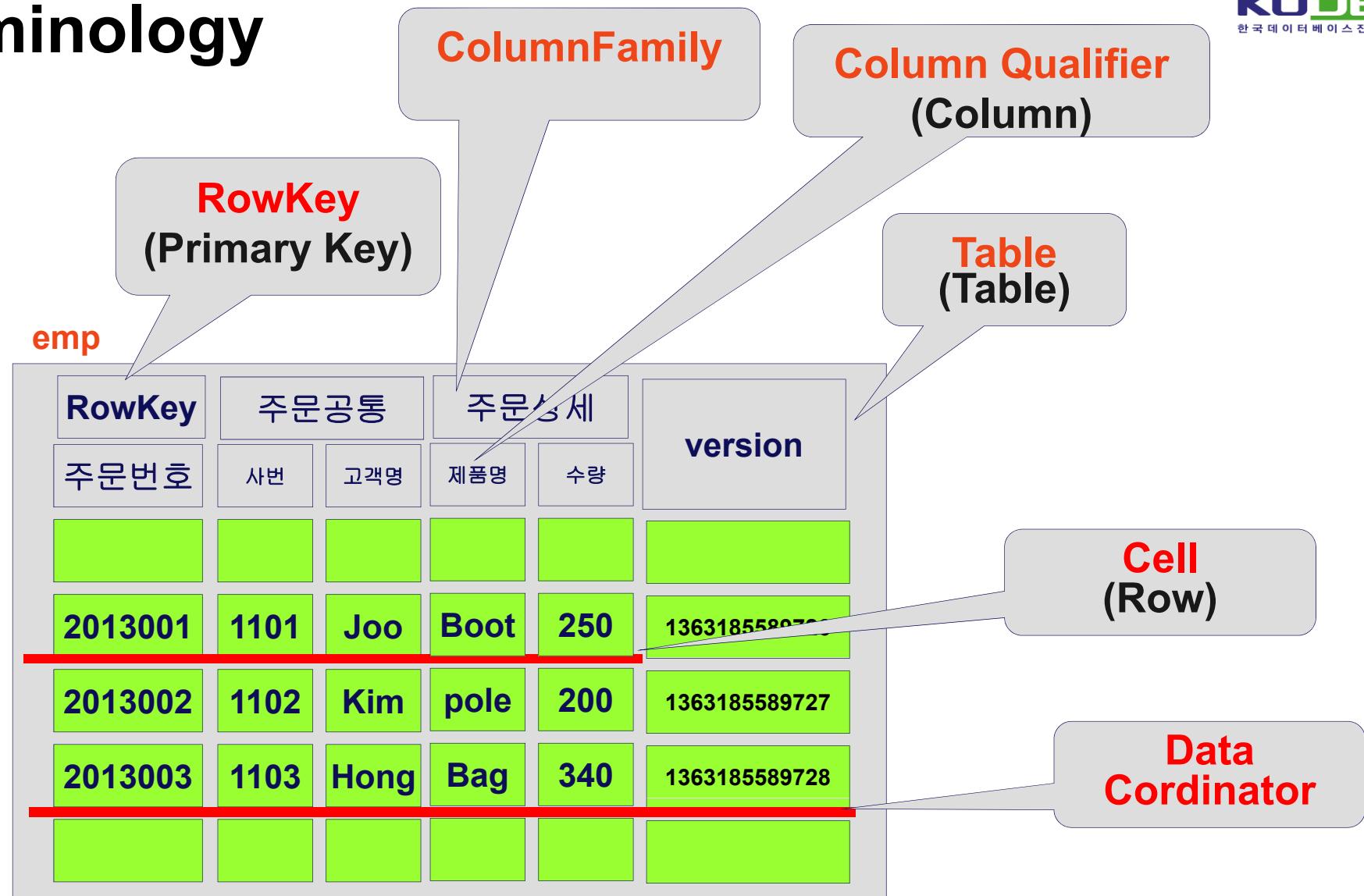
- 1) 구글 파일 시스템을 기반으로 하는 구글 **BigTable**(데이터 저장 기술)을 모델로 만들어졌으며 2008년도 **Hadoop**의 서브 프로젝트로 편입되어 개발되었다.
- 2) **Hadoop HDFS**와 **HDPS**를 기반으로 개발되었으며 **Zookeeper**를 이용한 고 가용성과 확장성이 보장된다.
- 3) 1000개 이상의 멀티 클러스터 노드를 확장할 수 있다.
- 4) **KeyValue** 중심의 데이터 저장기술로 구현되며 빅 데이터의 빠른 **WRITE/READ**에 적합하다.
- 5) **StandAlone** 모드, **Pseudo Distributed** 모드, **Full Distributed** 모드로 운영할 수 있다.

Hbase Architecture



Demonstration

Terminology



Hbase Command

```
$ ./bin/start-hbase.sh
```

```
starting Master, logging to logs/hbase-user-master-example.org.out
```

```
$ ./bin/hbase shell
```

```
HBase Shell; enter 'help<RETURN>' for list of supported commands.
```

```
Type "exit<RETURN>" to leave the HBase Shell
```

```
Version: 0.90.0, r1001068, Fri Sep 24 13:55:42 PDT 2010
```

```
hbase(main):001:0>
```

```
hbase(main):003:0> create 'test', 'cf1'
```

```
0 row(s) in 1.2200 seconds
```

```
hbase(main):003:0> list 'test'
```

```
test .....
```

```
1 row(s) in 0.0550 seconds
```

Copyright reserved.

hbase(main):004:0> **put 'test', 'r1', 'cf1:c1', '1101'**

0 row(s) in 0.0560 seconds

hbase(main):005:0> **put 'test', 'r2', 'cf1:c2', '1102'**

0 row(s) in 0.0370 seconds

hbase(main):006:0> **put 'test', 'r3', 'cf1:c3', '1103'**

0 row(s) in 0.0450 seconds

hbase(main):007:0> **scan 'test'**

ROW COLUMN+CELL

r1 column=cf1:c1, timestamp=1288380727188, value=1101

r2 column=cf1:c2, timestamp=1288380738440, value=1102

r3 column=cf1:c3, timestamp=1288380747365, value=1103

3 row(s) in 0.0590 seconds

hbase(main):008:0> get 'test', 'r1'

COLUMN CELL

cf1:c1 timestamp=1288380727188, value=1101

1 row(s) in 0.0400 seconds

hbase(main):012:0> disable 'test'

0 row(s) in 1.0930 seconds

hbase(main):013:0> drop 'test'

0 row(s) in 0.0770 seconds

주문전표

주문번호	2012-09-012345	담당사원	Magee		
고객명	Womansport (주)				
주문날짜	2012-09-20	선적날짜	2012-09-20	선적여부	Y
주문 총금액	601,100	지불방법	현금 30일 이내		

항목번호	제품명	단가	주문수량	금액
1	Bunny Boot	135	500	67,500
2	Pro Ski Boot	380	400	152,000
3	Bunny Ski Pole	14	500	7,000
4	Pro Ski Pole	36	400	14,400
5	Himalaya Bicycle	582	600	349,200
6	New Air Pump	20	450	9,000
7	Prostar 10Pd.Weight	8	250	2,000

SUMMIT2 (주)

Nested Entity

Row Key	Column Family	
ord_id	ord	ord_detail
2012-09-012345	ord:custom_name = W&Sports ord:emp_name = Magee ord:total = 601100 ord:payment_type = credit ord:order_filled = Y	ord_detail:item_id1= <prod_nm>Bunny Boots</prod_nm><item_price>135</item_price><qty>500</qty><price>67000</price> ord_detail:item_id2= <prod_nm>Pro Ski Boots</prod_nm><item_price>380</item_price><qty>400</qty><price>152000</price>

Nested Entity (문법)

create 'ord', 'ord', 'ord_detail'

put 'ord', '2012-09-012345, 'ord:**custom_name**', 'W&Sports'

put 'ord', '2012-09-012345, 'ord:**emp_name**', 'Magee'

put 'ord', '2012-09-012345, 'ord:**total**', '601100'

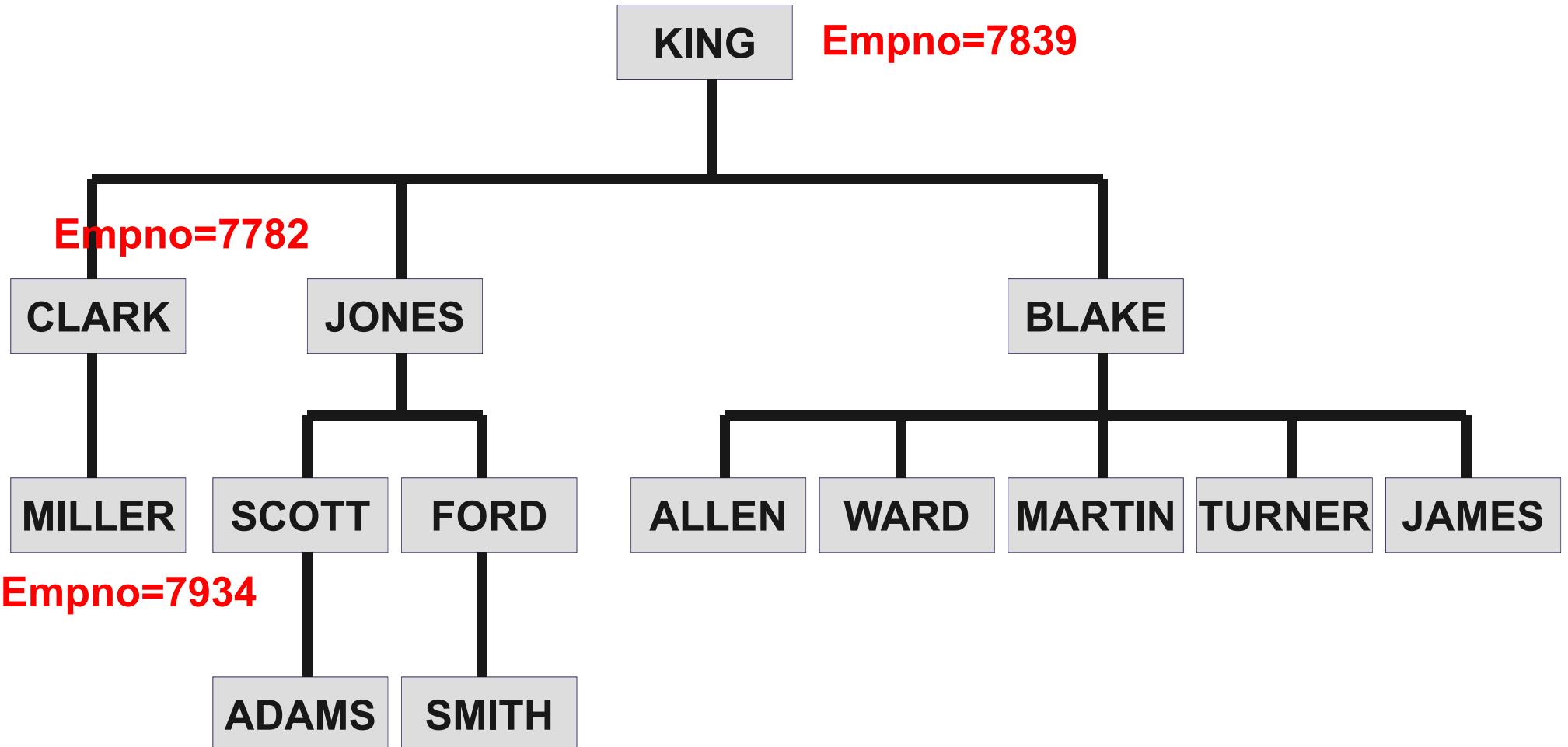
put 'ord', '2012-09-012345, 'ord:**payment_type**', 'Credit'

put 'ord', '2012-09-012345, 'ord:**order_filled**', 'Y'

put 'ord', '2012-09-012345, 'ord_detail:**item_id1**', '<**prod_nm</**prod_nm**><**item_price**>135</**item_price**><**qty**>500</**qty**><**price**>67000</**price**>**

put 'ord', '2012-09-012345, 'ord_detail:**item_id2**', '<**prod_nm</**prod_nm**><**item_price**>380</**item_price**><**qty**>400</**qty**><**price**>152000</**price**>**

계층 Structure



계층 Structure(Hbase)

Row Key	Column Family				
	id	info	ancestor	parent	child
7839		info:name='KING' Info:job='PRESIDENT'			child:1=7782 child:2=7934
7782		info:name='CLARK' Info:job='ANALYST'		parent:1=7839	child:1=7934
7934		info:name='MILLER' Info:job='CLERK'	ancestor:1=7839	parent:1=7782	

계층 Structure (문법)

create 'emp', 'info', 'ancestor', 'parent', 'child'

put 'emp', '7839', 'info:name', 'KING'

put 'emp', '7839', 'info:job', 'PRESIDENT'

put 'emp', '7839', 'child:1', '7782'

put 'emp', '7839', 'child:2', '7934'

put 'emp', '7782', 'info:name', 'CLARK'

put 'emp', '7782', 'info:job', 'ANALYST'

put 'emp', '7782', 'parent:1', '7839'

put 'emp', '7782', 'child:1', '7934'

put 'emp', '7934', 'info:name', 'MILLER'

put 'emp', '7934', 'info:job', 'CLERK'

put 'emp', '7934', 'ancestor:1', '7839'

put 'emp', '7934', 'parent:1', '7782'

served.

12 차시

Redis & 데이터 처리

Redis (Remote Directory System)



Commands Clients Documentation Community Download Issues License

Redis is an open source, BSD licensed, advanced **key-value store**. It is often referred to as a **data structure server** since keys can contain **strings, hashes, lists, sets** and **sorted sets**.

[Learn more →](#)

Try it

Ready for a test drive? Check this [interactive tutorial](#) that will walk you through the most important features of Redis.

Download it

[Redis 2.6.11 is the latest stable version.](#) Interested in release candidates or unstable versions? [Check the downloads page.](#)

What people are saying

redis-cached (0.0.1):
<http://t.co/dk45kyNnVN> A cache wrapper with redis

@ppiixx it's the jobs it runs. if you've got lots of jobs (e.g. scrapers). redis is a good place to store what needs scheduling...

@frabucus wouldn't you just have init/upstart/whatever start redis? why run it from cron?

Check out who is tweeting about: 'redis', here: <http://t.co/5slBbQlTDs>

@iriscouch any news on your Redis offering? The one at Nodejitsu doesn't seem stable.

[More...](#)



This website is open source software developed by Citrusbyte.
The Redis logo was designed by Carlos Prioglio. See more credits.

Sponsored by
vmware



<http://redis.io>

Copyright 2013 (주)플랜정보기술 All right reserved.

Key Value / Tuple Store

[DynamoDB](#) (will be inserted soon)

[Azure Table Storage](#) Collections of free form entities (row key, partition key, timestamp). Blob and Queue Storage available, 3 times redundant. Accessible via REST or ATOM.

[Couchbase Server](#) API: Memcached API+protocol (binary and ASCII) , most languages, Protocol: Memcached REST interface for cluster conf + management, Written in: C/C++ + Erlang (clustering), Replication: Peer to Peer, fully consistent, Misc: Transparent topology changes during operation, provides memcached-compatible caching buckets, commercially supported version available, Links: » [Wiki](#), » [Article](#)

[Riak](#) API: JSON, Protocol: REST, Query Method: MapReduce term matching , Scaling: Multiple Masters; Written in: Erlang, Concurrency: eventually consistent (stronger than MVCC via Vector Clocks), Misc: ... Links: talk »,

[Redis](#) API: Tons of languages, Written in: C, Concurrency: in memory and saves asynchronous disk after a defined time. Append only mode available. Different kinds of fsync policies. Replication: Master / Slave, Misc: also lists, sets, sorted sets, hashes, queues. Cheat-Sheet: » , great slides » Admin UI » From the Ground up »

[LevelDB](#) Fast & Batch updates. DB from Google. Written in C++. Blog » , hot Benchmark » , Article » (in

<http://nosql-database.org>

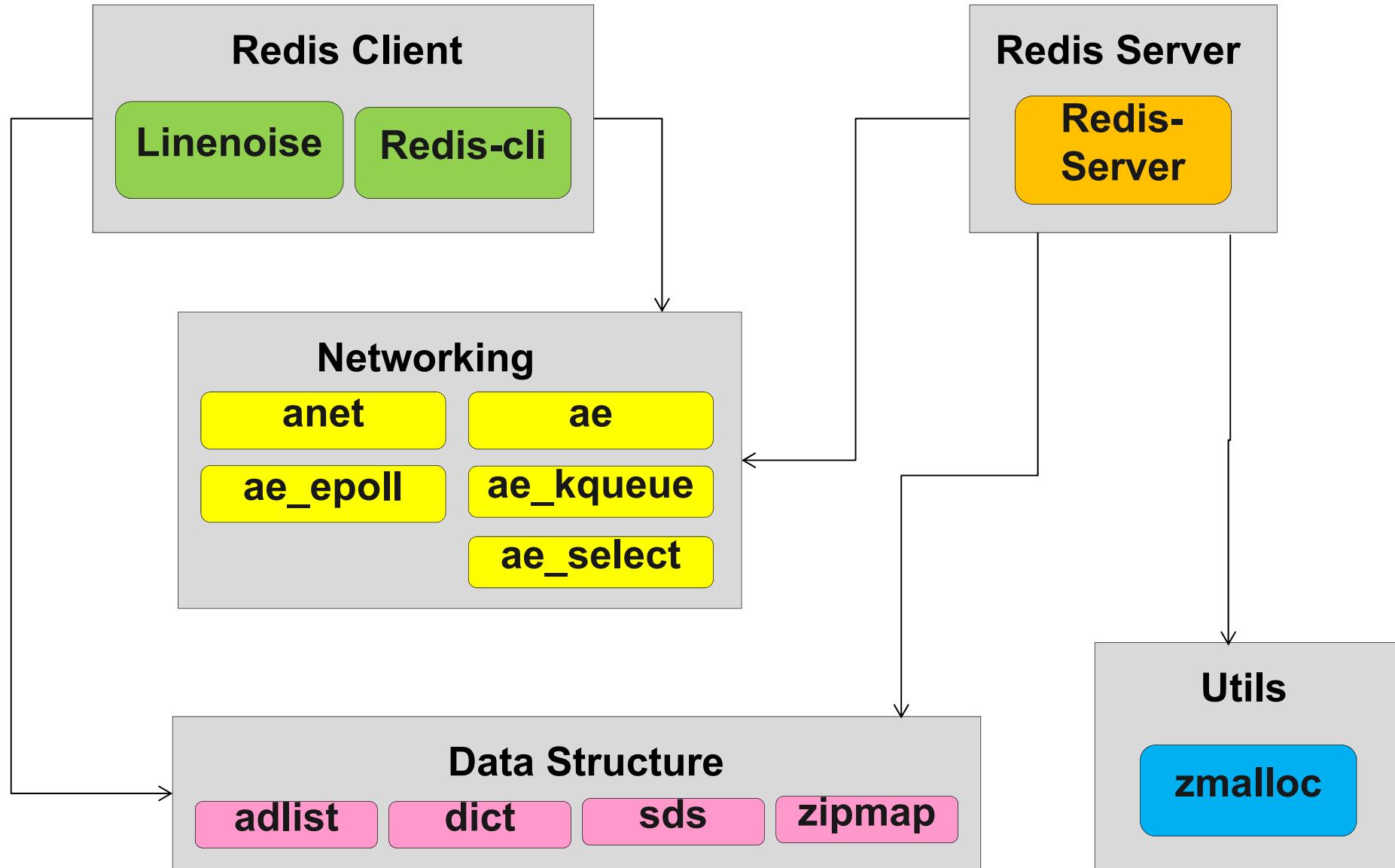
Redis 주요 특징

- 1) **KeyValue** 데이터베이스로 분류되는 **NoSQL**이며 2010년 **Vmware**에 의해 기술 지원되고 있다.
- 2) **In Memory** 상태를 유지하기 때문에 빠른 **Read/Write**가 특징이다.
- 3) **String, Set, Sorted Set, Hash Data value**를 저장할 수 있다.
- 4) **Snapshot**과 **AOF(Append Of File)** 방식으로 데이터를 저장한다.
- 5) **Master/Slave Replication** 기능을 제공한다.

Query Off Loading 기능을 제공한다. (**Master**에는 **Read/Write**를 수행하고 **Slave**에는 **Read**만 수행한다.)

- 6) **Sharding**을 통한 동적인 확장이 가능하다.
- 7) **Expiration** 기능은 일정 시간이 지나면 데이터를 자동 삭제한다.

Redis Architecture



Demonstration

Redis 설치 및 실행

1) Redis SW 설치

```
$ wget http://download.redis.io/releases/redis-2.8.17.tar.gz
$ tar xzf redis-2.8.17.tar.gz
$ mv redis-2.8.17 /home/mongodb/redis
$ cd /
$ cd redis
$ make
```

2) Redis Server 구동

```
$ cd /home/mongodb/redis/src
$ ./redis-server
```

3) Redis Client 실행

```
$ cd /home/mongodb/redis/src
$ ./redis-cli
redis>
```

Redis 환경 설정 (redis.conf)

Parameter	설명
port	TCP/IP 프로토콜을 위한 기본 Port 번호는 6379
database	Redis Server 내에 생성할 수 있는 데이터베이스의 수 (Default : 16)
save	변경되는 Key 당 디스크에 저장하는 Interval 설정 (Ex) SAVE 60 10000 (10,000 Key 가 변경될 때마다 60초에 한번 씩 저장)
dbfilename	디스크 상에 저장 될 DB 의 이름 설정 (Default : dump.rdb)
slaveof	Master –Slave 간에 데이터베이스 복제 시 복제 대상 서버를 지정
logfile	Redis Server 에서 발생하는 Log 데이터를 지정하는 경로에 생성

Redis 명령어

1) Set/Get 명령어

```
127.0.0.1:6379> set 1111 "<name>JM J00</name><mobile>01038641858</mobile>"  
OK  
127.0.0.1:6379> set 1112 "<name>YH J00</name><mobile>01075921858</mobile>"  
OK  
127.0.0.1:6379> get 1111  
"<name>JM J00</name><mobile>01038641858</mobile>"  
127.0.0.1:6379> get 1112  
"<name>YH J00</name><mobile>01075921858</mobile>"
```

2) Keys 명령어

```
127.0.0.1:6379> keys *  
1) "1111"  
2) "1112"  
127.0.0.1:6379>  
127.0.0.1:6379> keys *2*  
1) "1112"  
127.0.0.1:6379>  
127.0.0.1:6379> del 1112  
(integer) 1  
127.0.0.1:6379> keys *  
1) "1111"
```



3) Rename/Randomkey 명령어

```
127.0.0.1:6379> set 1112 "<name>YH J00</name><mobile>01075921858</mobile>"  
OK  
127.0.0.1:6379> set 1113 "<name>MH J00</name><mobile>01086777509</mobile>"  
OK  
127.0.0.1:6379> get 1113  
"<name>MH J00</name><mobile>01086777509</mobile>"  
127.0.0.1:6379> rename 1113 1116  
OK  
127.0.0.1:6379> randomkey  
"1116"  
127.0.0.1:6379> keys *  
1) "1111"  
2) "1116"  
3) "1112"
```

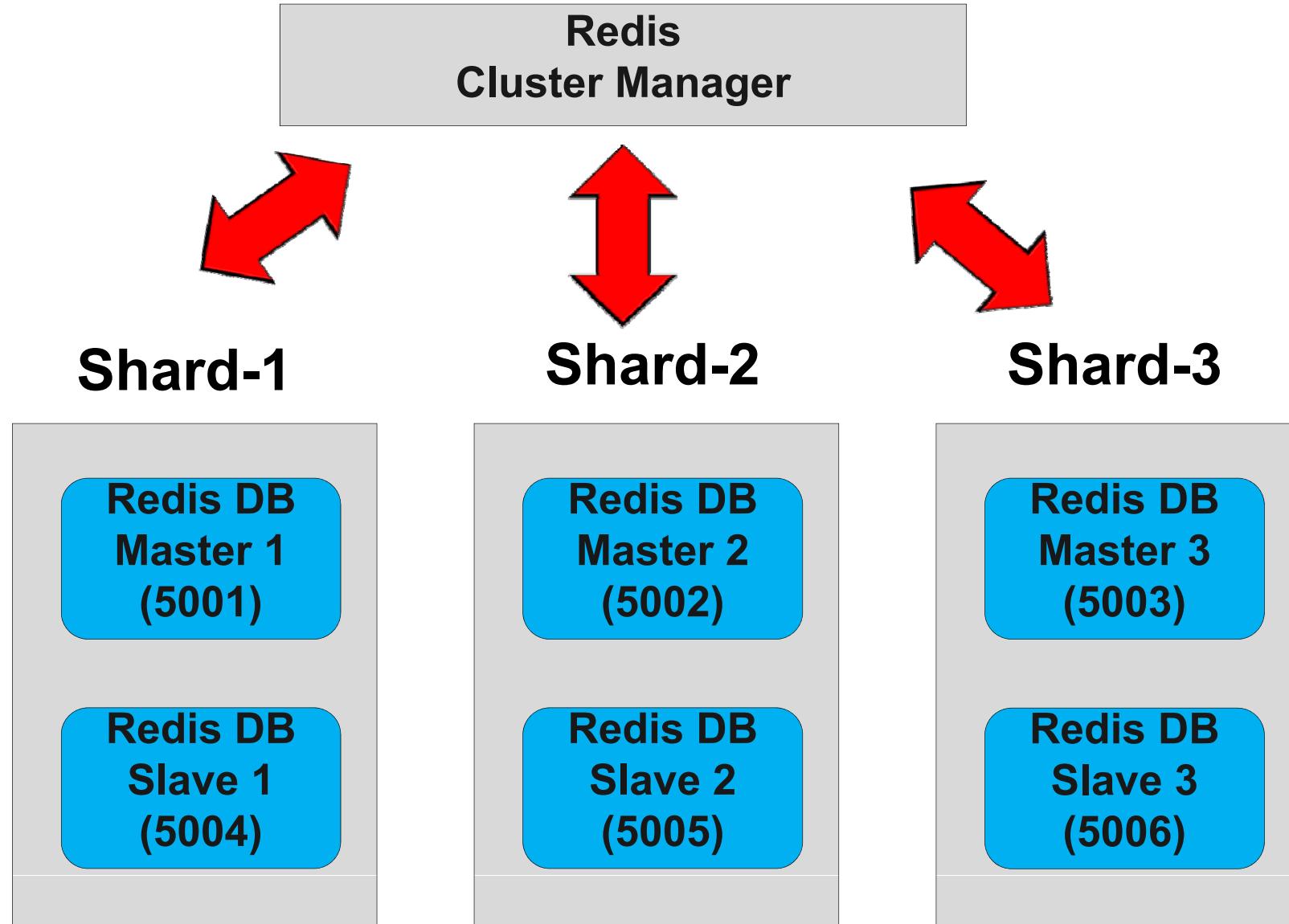
4) Exists 명령어

```
127.0.0.1:6379> keys *  
1) "1111"  
2) "1116"  
3) "1112"  
127.0.0.1:6379>  
127.0.0.1:6379> exists 1112  
(integer) 1  
127.0.0.1:6379> exists 1114  
(integer) 0
```

6) Mset/Mget 명령어

```
127.0.0.1:6379> exists 1112
(integer) 1
127.0.0.1:6379> exists 1114
(integer) 0
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> mset 1113 "MongoDB User Group" 1114 "PIT"
OK
127.0.0.1:6379> get 1113
"MongoDB User Group"
127.0.0.1:6379> get 1114
"PIT"
127.0.0.1:6379> mget 1113 1114
1) "MongoDB User Group"
2) "PIT"
127.0.0.1:6379> mget 1113 1114 nonexisting
1) "MongoDB User Group"
2) "PIT"
3) (nil)
```

Redis Multi Cluster



13 차시

Neo4J & 데이터 처리

NoSQL 제품군

1. Key-Value Database

- 1) Amazon's Dynamo Paper
- 2) Data Model : Collection of K-V pairs
- 3) 제품유형 : Riak, Voldemort, Tokyo*

3. Document Database

- 1) Lotus Notes
- 2) Data Model : Collection of K-V collection
- 3) 제품유형 : Mongo DB, Cough DB

2. BigTable Database

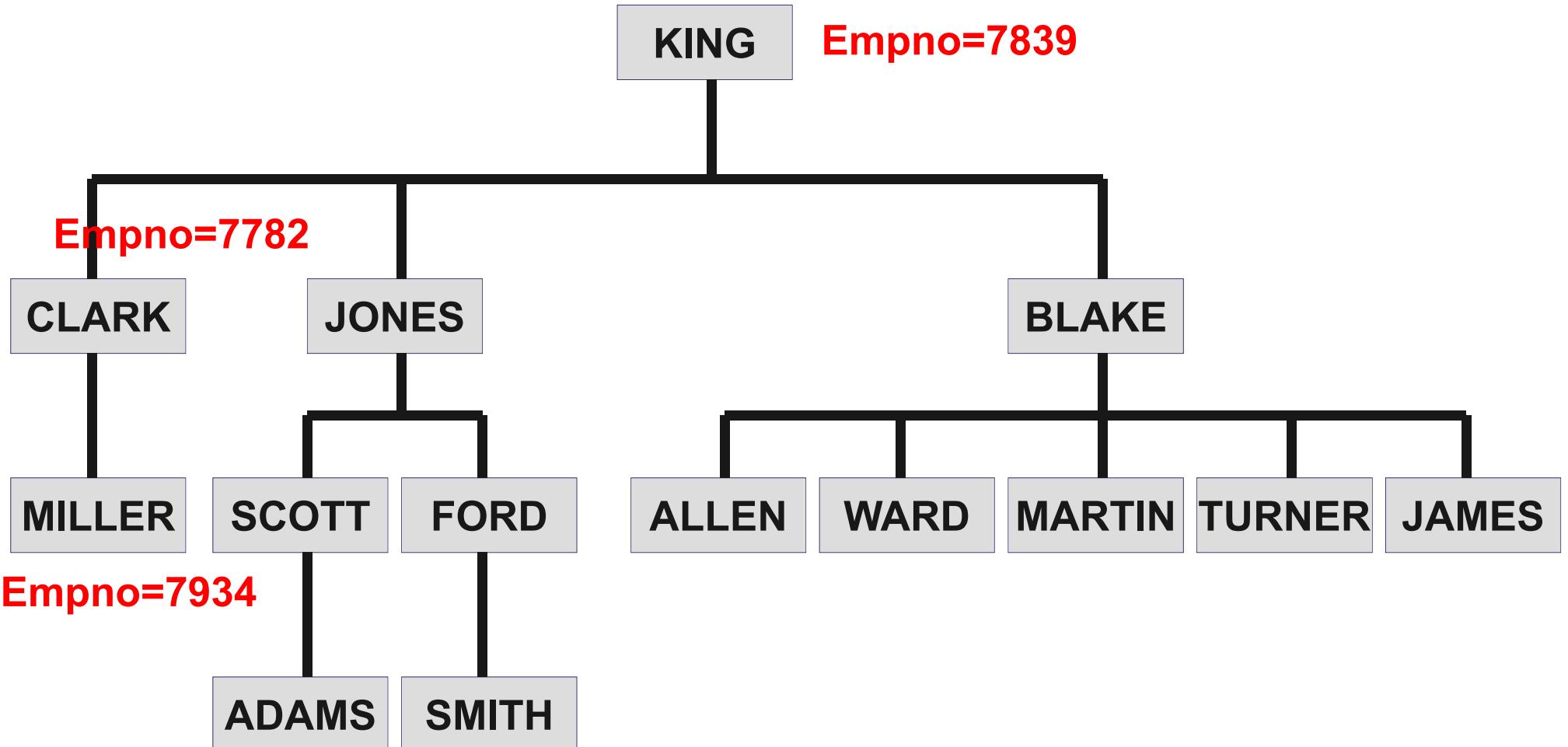
- 1) Google's BigTable paper
- 2) Data Model : Column Families
- 3) 제품유형 : Hbase, Casandra, Hypertable

4. Graph Database

- 1) Euler & Graph Theory
- 2) Data Model : nodes, rels, K-V on both
- 3) 제품유형 : Neo4J, Sones

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

4) 계층형 데이터 구조



Ancestor Reference (MongoDB)

7839 : KING

7782 : CLARK

7934 : MILLER

```
> db.emp.insert({ "_id" : "7839", "name" : "KING",    "job" : "PRESIDENT" })
> db.emp.insert({ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",
                  "PARENT"      : "7839" } )
> db.emp.insert({ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",
                  "ANCESTORS"  : "7839",
                  "PARENT"      : "7782" } )

> db.emp.find({"ANCESTORS" : "7839"})
{ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",
  "ANCESTORS" : [ "7839", "7782" ], "PARENT" : "7782" }

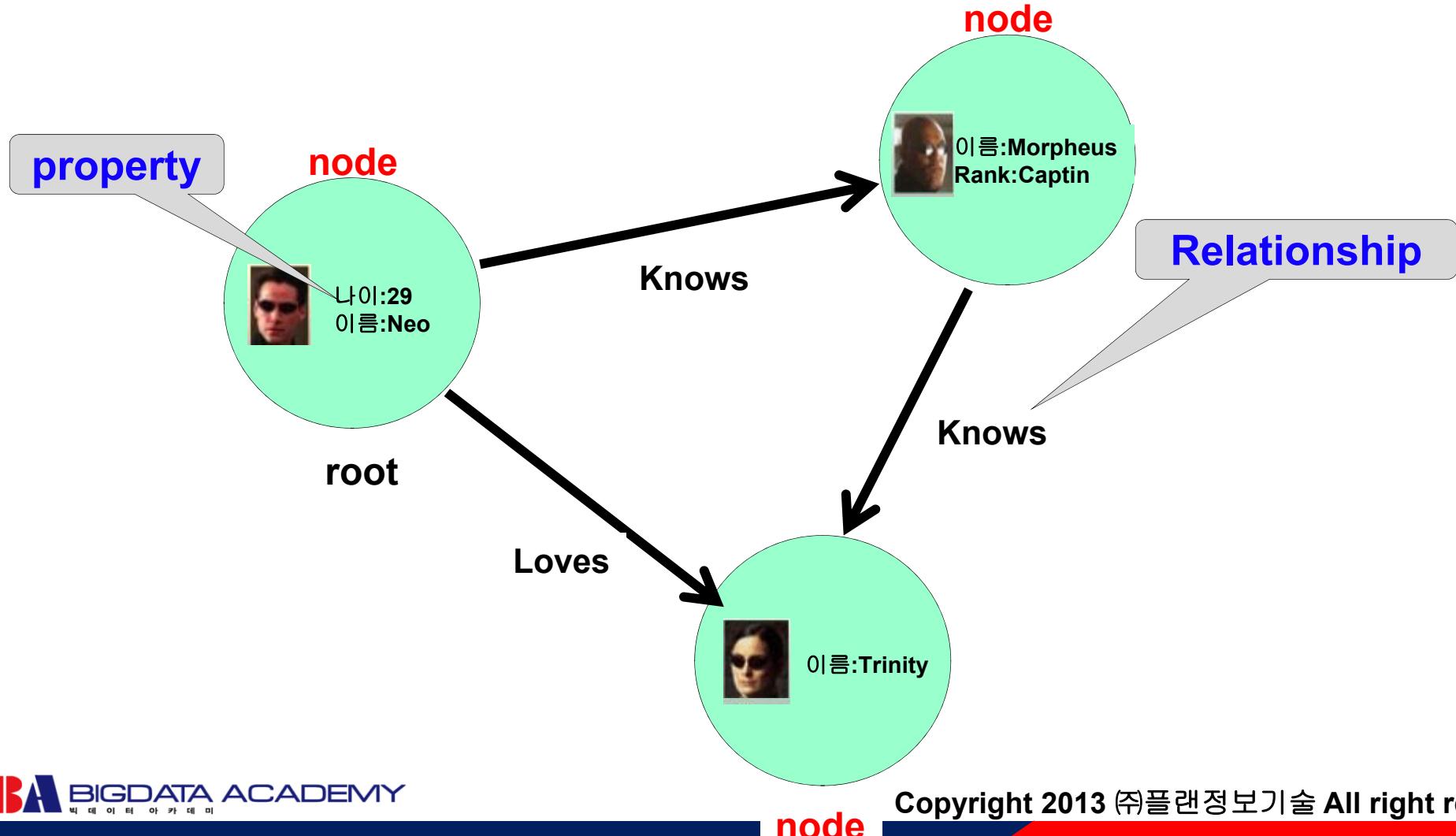
> db.emp.find({"PARENT" : "7839"})
{ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",
  "PARENT" : "7839" }
```

- 1) 기업에서 발생하는 데이터 구조 중에는 계층형 데이터 구조가 발생할 수 밖에 없는데 이런 경우 적용하면 가장 이상적인 데이터 모델이다.
- 2) ANCESTORS와 PARENT Field로 표현할 수 있으며 하나의 Document는 하나 이상의 ANCESTORS 와 PARENT를 가질 필요는 없다.

- 1) **Graph Database**는 **Node, Property, Relationship**으로 구성되는 데이터 저장 요소를 가진다.
- 2) **JAVA** 기반의 데이터베이스이며 **Embedded** 방식을 지원한다.
- 3) **INDEX** 및 노드 탐색을 지원한다.
- 4) 이중화를 통한 고가용성을 지원한다. (**Zookeeper**)
- 5) 트랜잭션 제어 및 백업/복구 기능을 지원한다.
- 6) 빠른 쓰기와 읽기가 요구되는 빅 데이터 처리 환경 보다는 소셜 네트워크 와 같은 **계층형 데이터 구조**를 처리하는데 적합하다.

Graph

. 하나의 Graph는 Node, property, Relationship으로 구성



Node 생성

GraphDatabaseService graphdb= ...

```
Node Neo = graphdb.createNode();
Neo.setProperty("name", "Neo");
Neo.setProperty("age", 29);
```

```
Node Morpheus = graphdb.createNode();
Morpheus.setProperty("name", "Morpheus");
Morpheus.setProperty("rank", "captain");
```

```
Neo.createRelationshipTo( Morpheus, RelTypes.Knows );
```

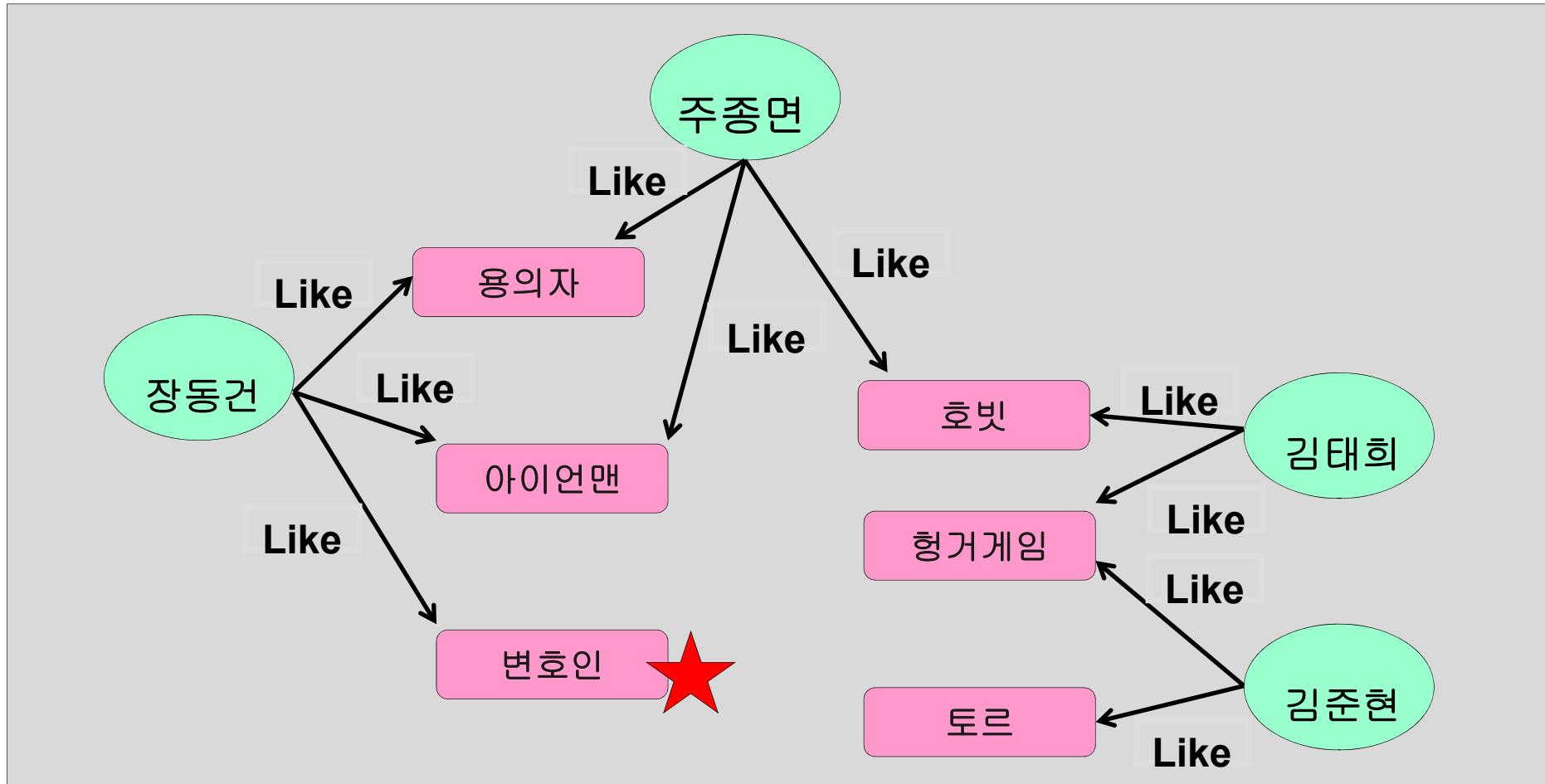
Cyper Query Language

```
create (Neo:Crew {name:'Neo', age:29}),  
       (Morpheus:Crew {name: 'Morpheus', rank:'captain'}),  
       (Trinity:Crew {name: 'Trinity'}),  
       (Neo)-[:Knows]→(Morpheus),  
       (Neo)-[:Loves]→(Trinity),  
       (Morpheus)-[:Knows]→(Trinity)
```

```
MATCH (n:Crew)-[r:KNOWS*]-m  
WHERE n.name='Neo'  
RETURN n AS Neo,r,m
```

Neo	r	m
(0:Crew {name:"Neo"})	[(0)-[0:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
(0:Crew {name:"Neo"})	[(0)-[0:KNOWS]->(1), (1)-[2:KNOWS]->(2)]	(2:Crew {name:"Trinity"})

Sample 1



```
CREATE ( movie1 : Movie : { title : '용의자' , year : 2013 } )
CREATE ( movie2 : Movie : { title : '아이언맨' , year : 2012 } )
CREATE ( movie3 : Movie : { title : '호빗' } )
CREATE ( movie4 : Movie : { title : '변호인' } )
CREATE ( movie5 : Movie : { title : '헝거게임' } )
CREATE ( movie6 : Movie : { title : '토르' } )
```

```
CREATE ( customer1:Customer { name : '주종면' } )
CREATE ( customer1)-[:Like]->(movie1)
CREATE ( customer1)-[:Like]->(movie2)
CREATE ( customer1)-[:Like]->(movie3)
```

```
CREATE ( customer2:Customer { name : '장동건' } )
CREATE ( customer2)-[:Like]->(movie1)
CREATE ( customer2)-[:Like]->(movie2)
CREATE ( customer2)-[:Like]->(movie4)
```

```
CREATE ( customer3:Customer { name : '김태희' } )
CREATE ( customer3)-[:Like]->(movie3)
CREATE ( customer3)-[:Like]->(movie5)
```

```
MATCH (:Movie { title: '용의자' })
RETURN title, year;
```

```
MATCH (:Customer)
RETURN name
ORDER BY name DESC;
```

```
MATCH (:Movie)
RETURN count(*) ;
```

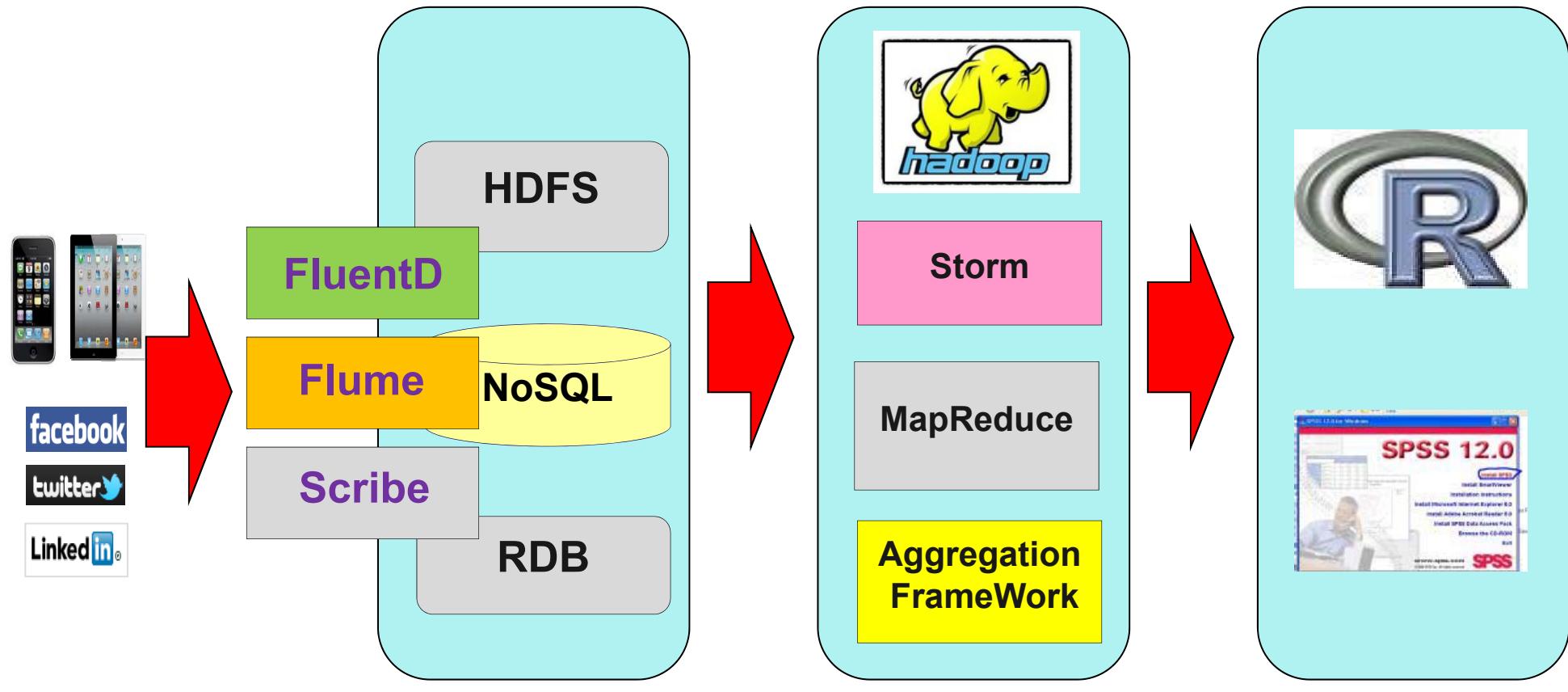
```
MATCH (:Movie { title: "아이언맨" })←[:Like]-(customer)-[:Like]→(movie1)
RETURN title, count(*) ;
```

```
MATCH (n)-[r]->(m)
RETURN n AS FROM , r AS `->` , m AS to;
```

14 차시

FlunetD & 데이터 수집

Big Data Processing Flow



The screenshot shows the official FluentD website. At the top left is the FluentD logo (a stylized butterfly icon next to the word "fluentd"). At the top right are links for Documentation, Plugins, MailingList, Issues, Source, and a Star button with the number 951. The main headline reads "Log everything in JSON". Below it are two buttons: "GET STARTED >" and "PLUGINS >". To the right, there's a large circular graphic containing a JSON configuration snippet. Two arrows point from the text below the heading to the "pluggable" and "lightweight" keys in the JSON. The JSON code is:

```
{  
  "type": "logging daemon",  
  "pluggable": true,  
  "lightweight": true,  
  "availability": "high"  
}
```

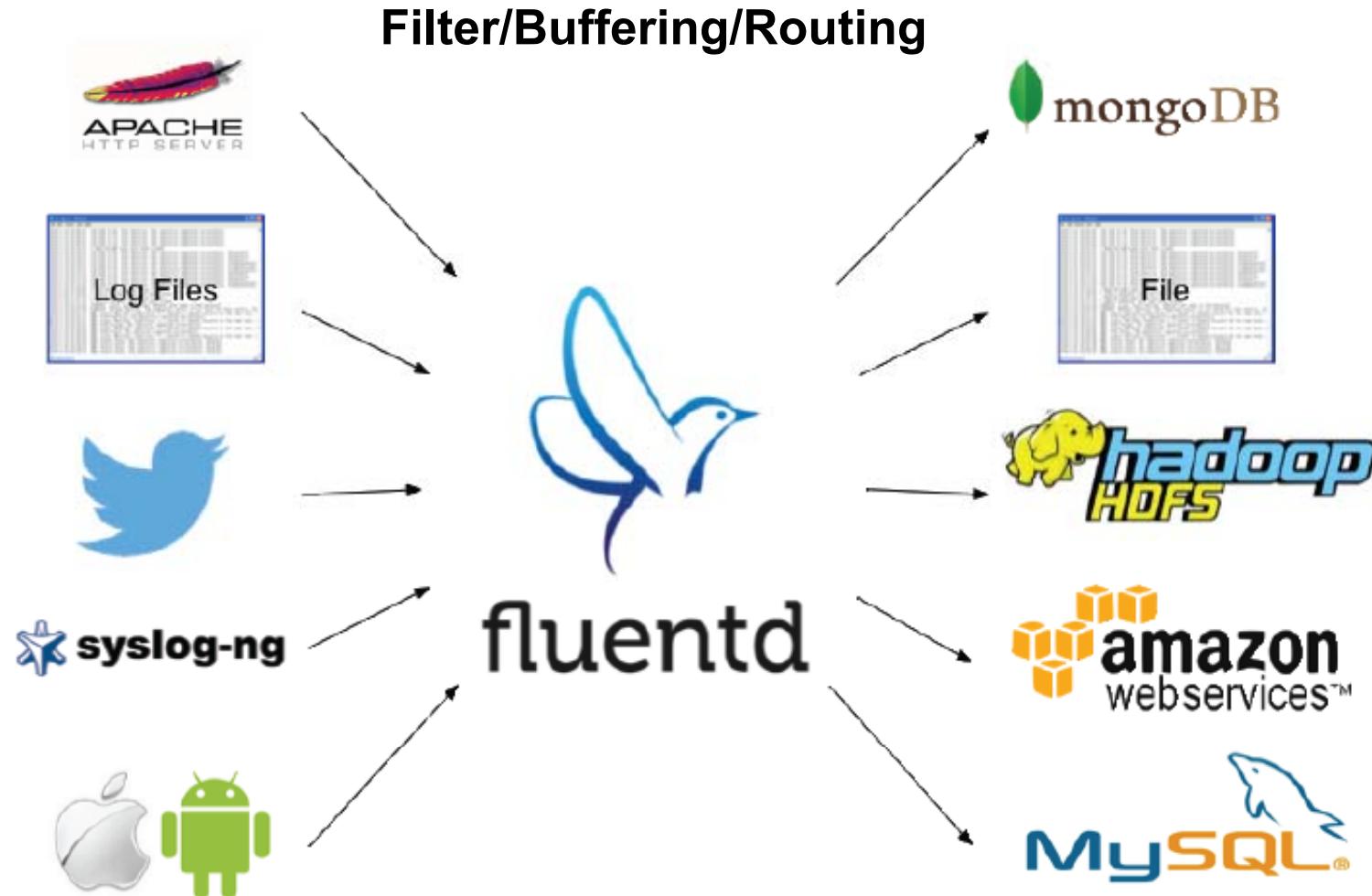
"Fluentd" is a OSS lightweight and flexible log collector. Fluentd receives logs as JSON streams, buffers them, and sends them to other systems like S3, MongoDB, Hadoop, or other Fluentds.

 **Install Fluentd**

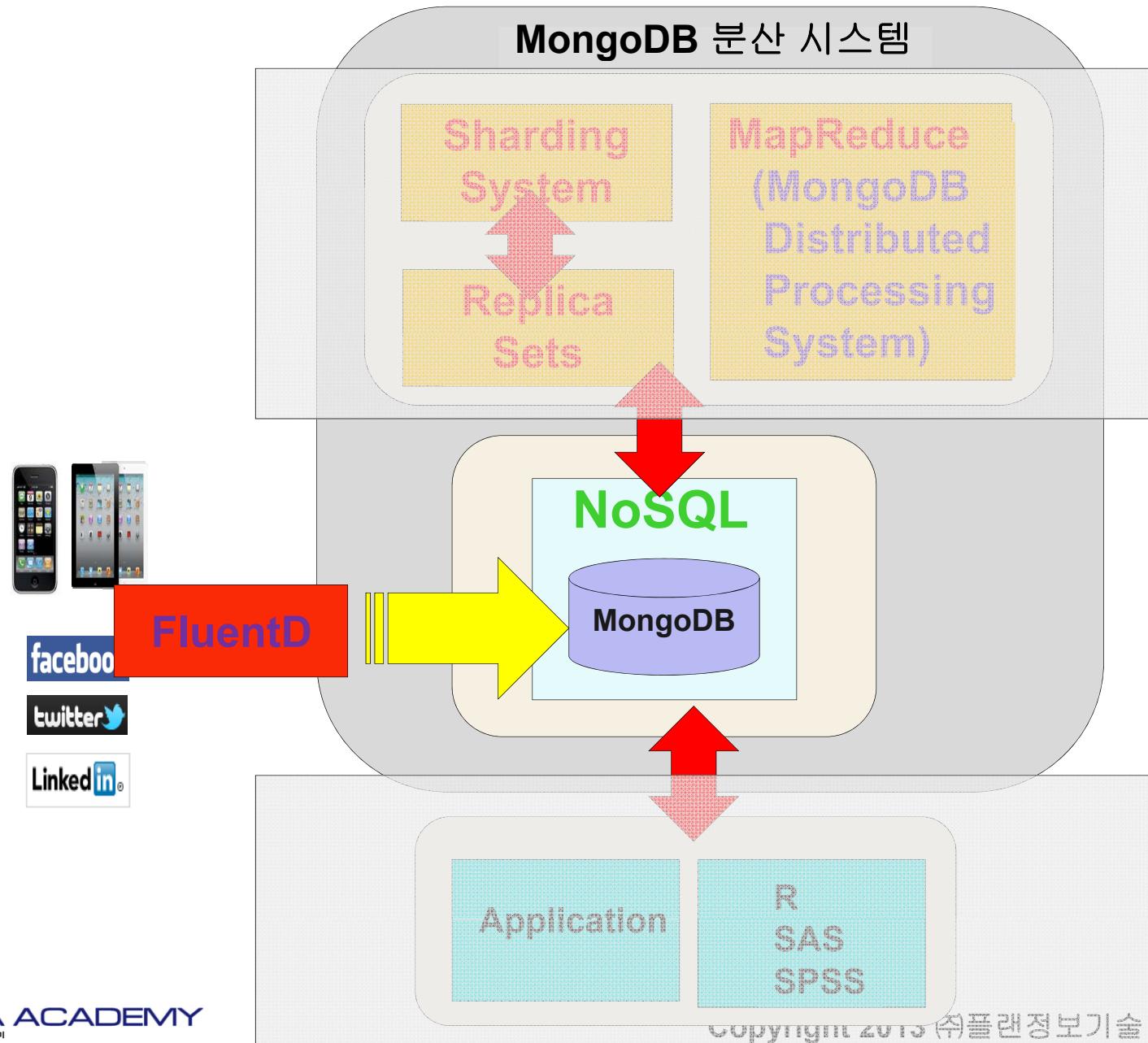
RPM package (.rpm) Debian package (.deb) gem install fluentd

Fluentd in the Industry

FluentD



FluentD & MongoDB



FlunetD 설치

Ubuntu Precise

Please add the Treasure Data apt packages repository to your sources list. This command will set it up for you:

```
$ sudo apt-add-repository 'deb http://packages.treasure-data.com/precise/ precise contrib'
```

Ubuntu Lucid

Then, please update your apt repositories and install td-agent package:

```
$ apt-get update  
$ apt-get install td-agent
```

To upgrade the existing agent, run the following command:

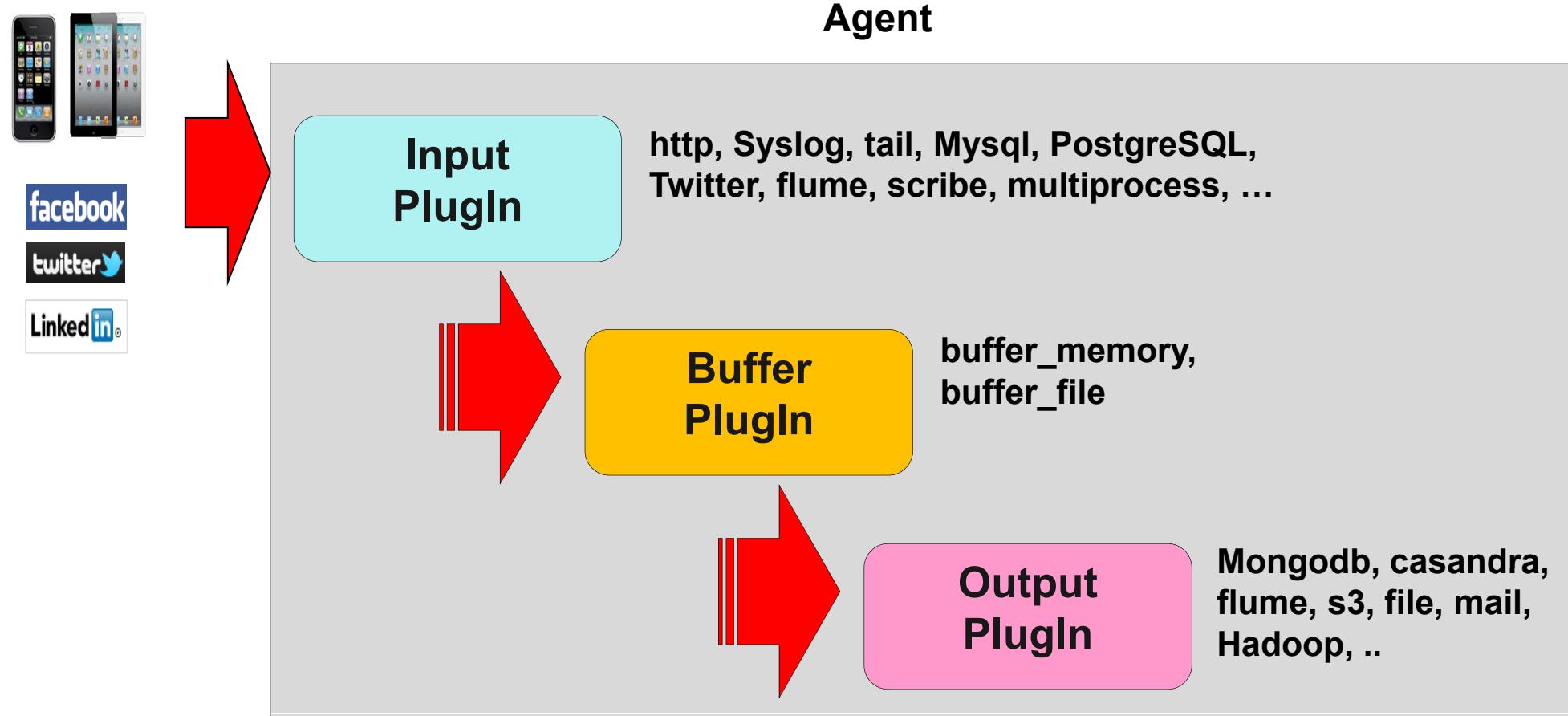
```
$ apt-get update && apt-get install td-agent
```

/etc/init.d/td-agent script is also provided to start, stop, or restart the agent.

```
$ /etc/init.d/td-agent start  
$ /etc/init.d/td-agent stop  
$ /etc/init.d/td-agent restart
```

Demonstration

FluentD Architecture



td-agent.conf

Tail Input

```
<source>
  type tail
  format apache2
  path /var/log/apache2/access_log
  pos_file /var/log/td-agent/apache2.access_log.pos
  tag mongo.apache.access
</source>
```

Http Input

```
<source>
  type http
  port 8888
</source>
```

Scribe Input

```
<source>
  type scribe
  port 1463
  bind 0.0.0.0
  msg_format json
</source>
```

Mongo Output

```
<match mongo.*.*>
  # plugin type
  type mongo

  # mongodb db + collection
  database apache
  collection access

  # mongodb host + port
  host localhost
  port 27017

  # interval
  flush_interval 10s
</match>
```

Mongo_Replaset Output

```
# Single MongoDB
<match mongo.*.*>
  type mongo_replset
  database fluentd
  collection test
  nodes localhost:27017,localhost:27018,localhost:27019

  # flush
  flush_interval 10s
</match>
```

Stdout Output

```
<match pattern>
  type stdout
</match>
```

```
<match hdfs.*.*>
  type webhdfs
  host namenode.your.cluster.local
  port 50070
  path /log/%Y%m%d_%H/access.log.${hostname}
  flush_interval 10s
</match>
```

HDFS Output

Example

td-agent.conf

```
<source>
  type http
  port 8888
</source>

<match hdfs.*.*>
  type webhdfs
  host namenode.your.cluster.local
  port 50070
  path /log/%Y%m%d_%H/access.log.${hostname}
  flush_interval 10s
</match>
```

hdfs-site.xml

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.support.append</name>
  <value>true</value>
</property>

<property>
  <name>dfs.support.broken.append</name>
  <value>true</value>
</property>
```

```
$ curl -X POST -d 'json={"action":"login","user":2}' \
http://localhost:8888/hdfs.access.test
$ kill -USR1 `cat /var/run/td-agent/td-agent.pid`
```

We can then access HDFS to see the stored data.

```
$ sudo -u hdfs hadoop fs -lsr /log/
drwxr-xr-x  -  supergroup          0 2012-10-22 09:40 /log/20121022_14/access.log.dev
```

감사합니다.

참고문헌: “NoSQL & mongoDB”
도서출판 DataBook. 주종면 저.

MongoDB Master 주종면
www.pitmongo.co.kr
jina6678@daum.net
010-3864-1858