5. Document DB 데이터 모델링

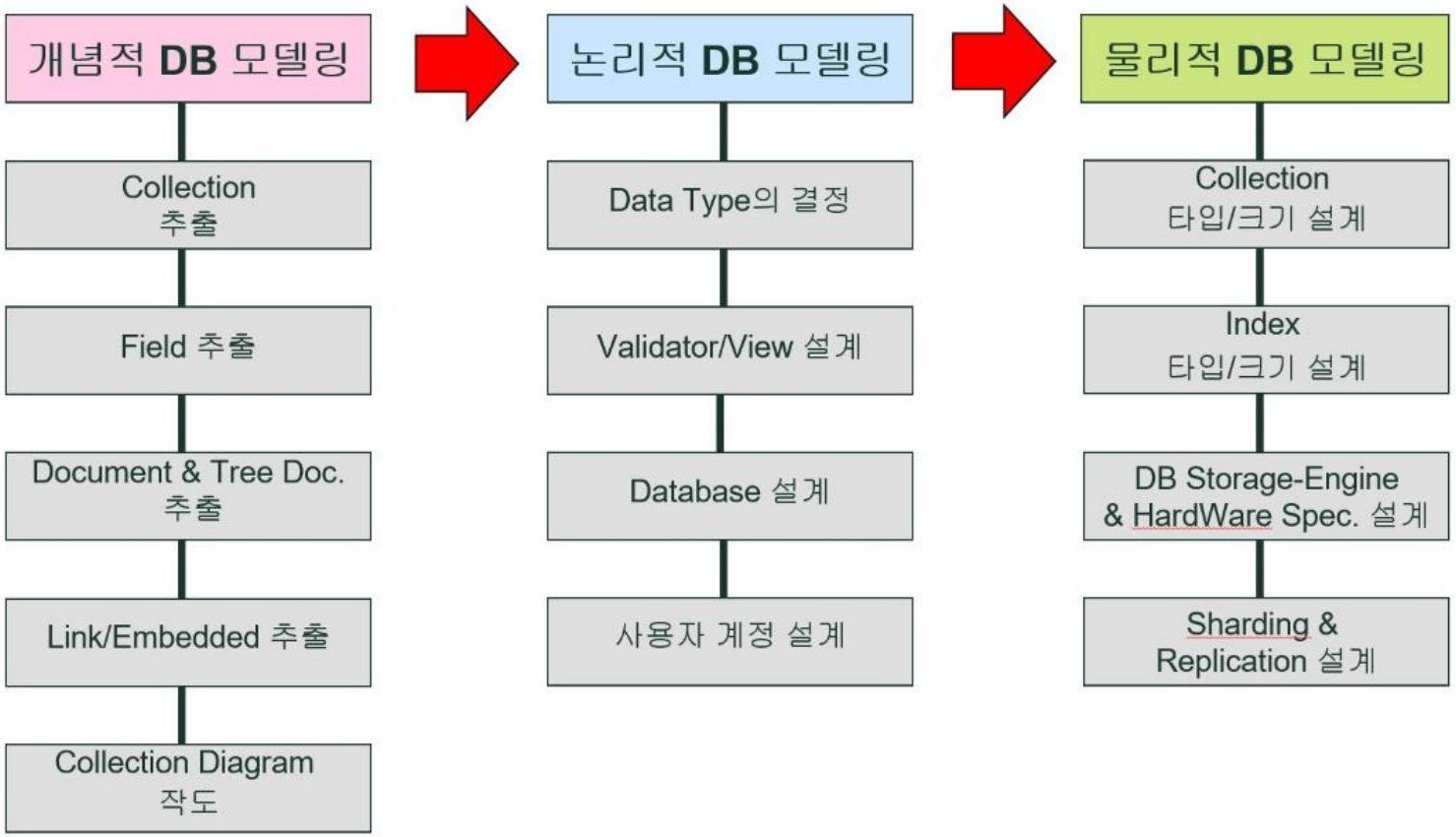
차례

1. MongoDB 데이터 모델링

- 데이터 모델링
  - 실세계의 내용을 그림과 도형으로 나타내는 것
  - 데이터가 저장되어 있는 모양
- 모델링 절차
  - 개념적 모델링 -> 논리적 모델링 -> 물리적 모델링
    - 개념적 모델링: 비즈니스 영역으로부터 데이터를 추출하는 단계
    - 논리적 모델링 : Document 구조에 맞게 분석 설계하는 과정
    - 물리적 모데링: MongoDB의 물리적 구조에 맞게 설계하는 단계

1. MongoDB 데이터 모델링

- 분석/설계 단계



2. MongoDB 설계의 주요 특징

- Data& 프로세스 설계의 중심
  - HOST 환경기반 **파일 시스템 : 프로세스 중심**의 데이터 구조설계
  - **클라이언트/서버 환경의 관계형 DB : 트랜잭션의 효율적인처리를 위한 Data 중심의 설계 기법**을 지향
  - 클라우드 컴퓨팅 환경의 **NoSQL : Data와 프로세스 모두를 설계의 중심에 둠**. (Big Data의 수집 및 저장이 중심)
- Rich Document Structure
  - **관계형 DB**는 **정규화**를 통해 **데이터 중복**을 제거하며 **무결성**을 보장하는설계 기법을 지향.
  - **NoSQL**은 데이터의 **중복을 허용하며 역정규화된 설계를 지향**.
  - 관계형 DB가 요구되었던 당시와 달리 저장장치의 비약적 발전과 저렴한 가격 요인도 설계에 중요한 요소임

2. MongoDB 설계의 주요 특징

- N:M 관계 구조를 설계 가능
  - 관계형 DB는 Entity 간의 N:M 관계 구조를 설계할 수 없지만 **NoSQL은 N:M 관계 구조를 설계할 수 있고 구축 가능**
- 불필요한 JOIN을 최소화 시킴
  - 관계형 DB는 Entity 간의 Relationship을 중심으로 데이터의 무결성을 보장하지만 불필요한 **JOIN**을 유발시킴으로써 코딩양을 증가시키고 검색성능을 저하시키는 원인을 제공.
  - **NoSQL은 중첩 데이터 구조를 설계할 수 있기 때문에 불필요한 JOIN을 최소화 시킴**

2. MongoDB 설계의 주요 특징

- Schema 없음
  - Document DB는 기본적으로 **Schema가 없기** 때문에 유연한 데이터구조를 설계 가능.
- 관계형 DB에 비해 빠른 쓰기/읽기
  - 기존의 업무 영역에서 처리할 수 없었던 비정형 데이터에 대한 저장과관리가 가능하며 **관계형 DB에 비해 빠른 쓰기와 읽기가 가능한** 데이터설계가 가능.
- 유연성이 있는 서버 구조 제공
  - 관계형 DBMS는 서버에 설치할때 DBMS전용 메모리 영역 활당하여 제한된 크기의 메모리만 사용 가능
  - NoSQL은 유연한 자원 할당 기술을 사용

3. MongoDB 설계 기준

- 데이터 조작은 어떻게 수행되는가?
- ACCESS PATTERN은 어떤가?
- SCHEMA 설계시 고려 사항은?

3. MongoDB 설계 주요 패턴

- Embedded Document(Rich Document)구조
- Extent Document(Rich Document) 구조
- Link 구조
- Inheritence (OODBMS) 구조
- 계층형 데이터 구조

3. MongoDB 설계 주요 패턴

주 문 전 표

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

항목번호	제 품 명	단 가	주문수량	금 액
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

3. MongoDB 설계 주요 패턴

- 관계형 DBMS

주문 테이블

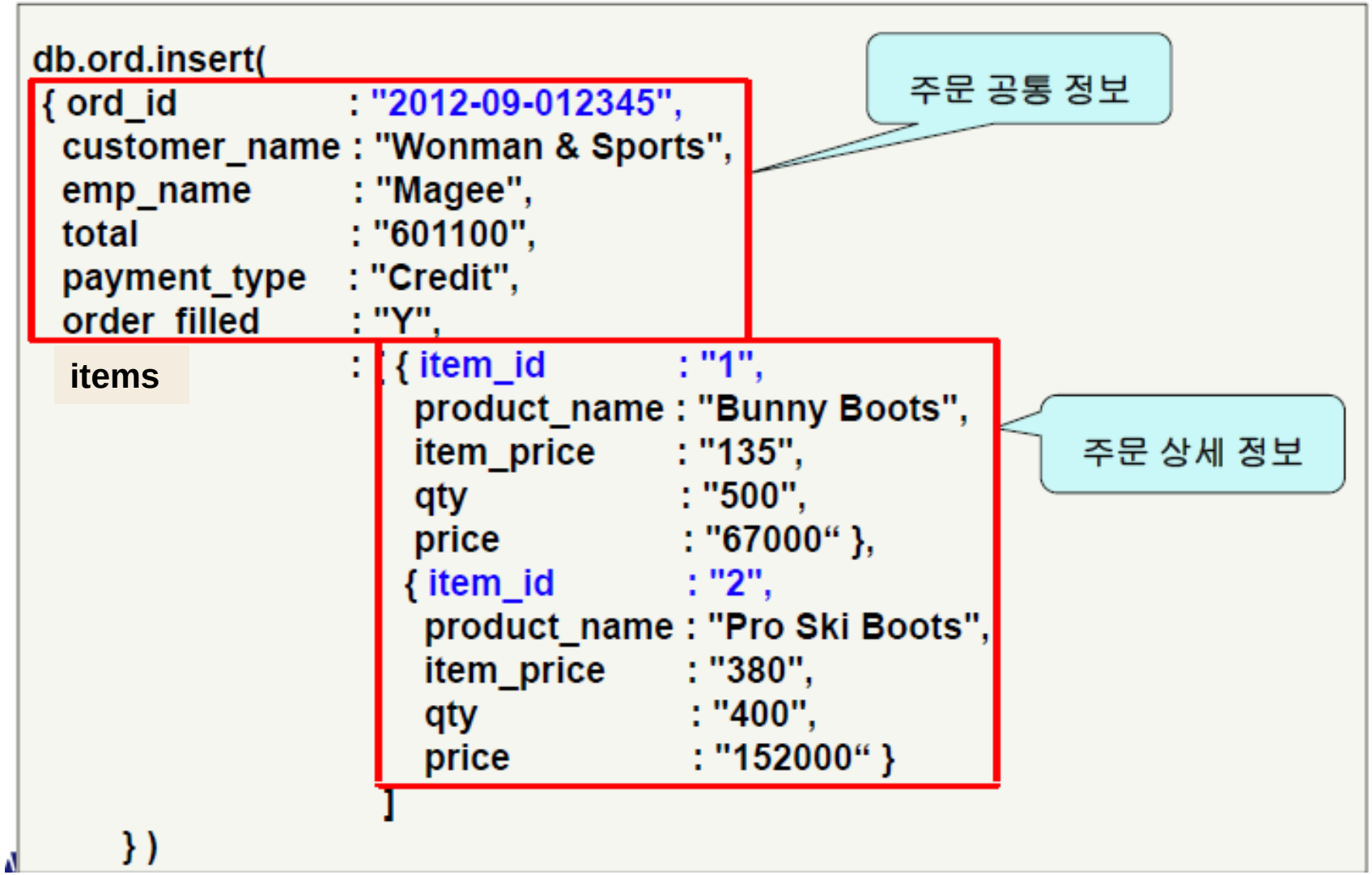
2012-09-012345, Wonman&Sports, Magee,601100, Credit, Y ….. …..	2012-09-012346, Man&Sports, Magee,34200, Credit, N ….. …..	2012-09-012347, Adidas, Magee,23100, Credit, Y ….. …..	2012-09-012348, Soleman, Magee,43100, Credit, N ….. …..

주문 상세 테이블

2012-09-012345, 1, Bunny Boot, 135,500,67000	2012-09-012345, 2, Pro Ski Boots, 380,400,152000		

3. MongoDB 설계 주요 패턴

- Embeded Document(Rich Document)



3. MongoDB 설계 주요 패턴(Extent Document)

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

주문정보

```
db.ord.update(
{ord_id : "2012-09-012345"},
{$set :{items :[{item_no: "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" }]}})
```

주문상세정보

3. MongoDB 설계 주요 패턴

- Rich Document 구조의 장단점
  - Query가 단순함
  - Join문 실행할 필요가 없기 때문에 도큐먼트 단위의 데이터 저장에 효과적이고 빠른 성능 보장
  - 데이터 보안에 효과적
  - Embedded되는 도큐먼트의 크기는 최대 16MB범위에서 가능
  - Embedded 되는 도큐먼트가 존재하지 않는 컬렉션에는 부적합

3. MongoDB 설계 주요 패턴(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" } )

> 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 : o._id })

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

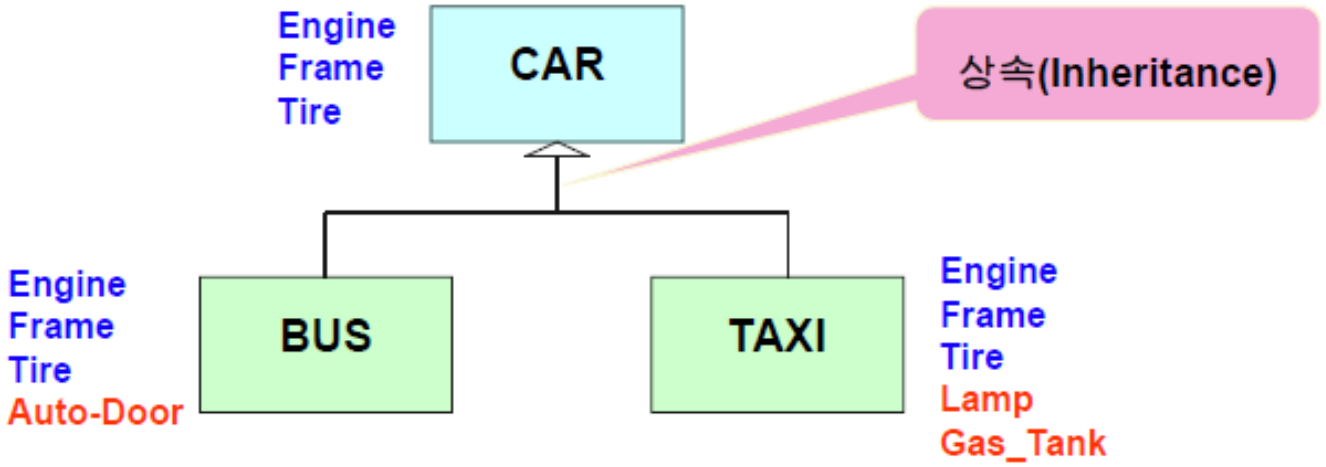
주문 공통 정보

주문 상세 정보

. MongoDB 설계 주요 패턴

- Link 구조의 장단점
  - 별도의 논리적 구조로 저장되기 때문에 도큐먼트 크기에 제한 받지 않는다.
  - 비즈니스 룰 상 별도로 처리되는 데이터 구조에 적합
  - 매번 논리적 구조 간에 Link 해야하기 때문에 Embedded보다 성능이 떨어짐
  - 컬렉션 개수가 증가하며 관리 비용이 높아짐

Inheritence (OODBMS)

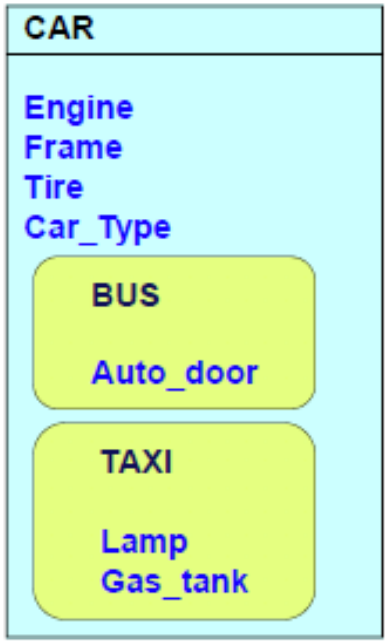



```
CREATE TYPE car  AS OBJECT
(engine      NUMBER(9)     Primary  Key,
 frame       VARCHAR(30),
 tire        VARCHAR(30))  NOT  FINAL;

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 Inheritence (RDBMS)




```
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

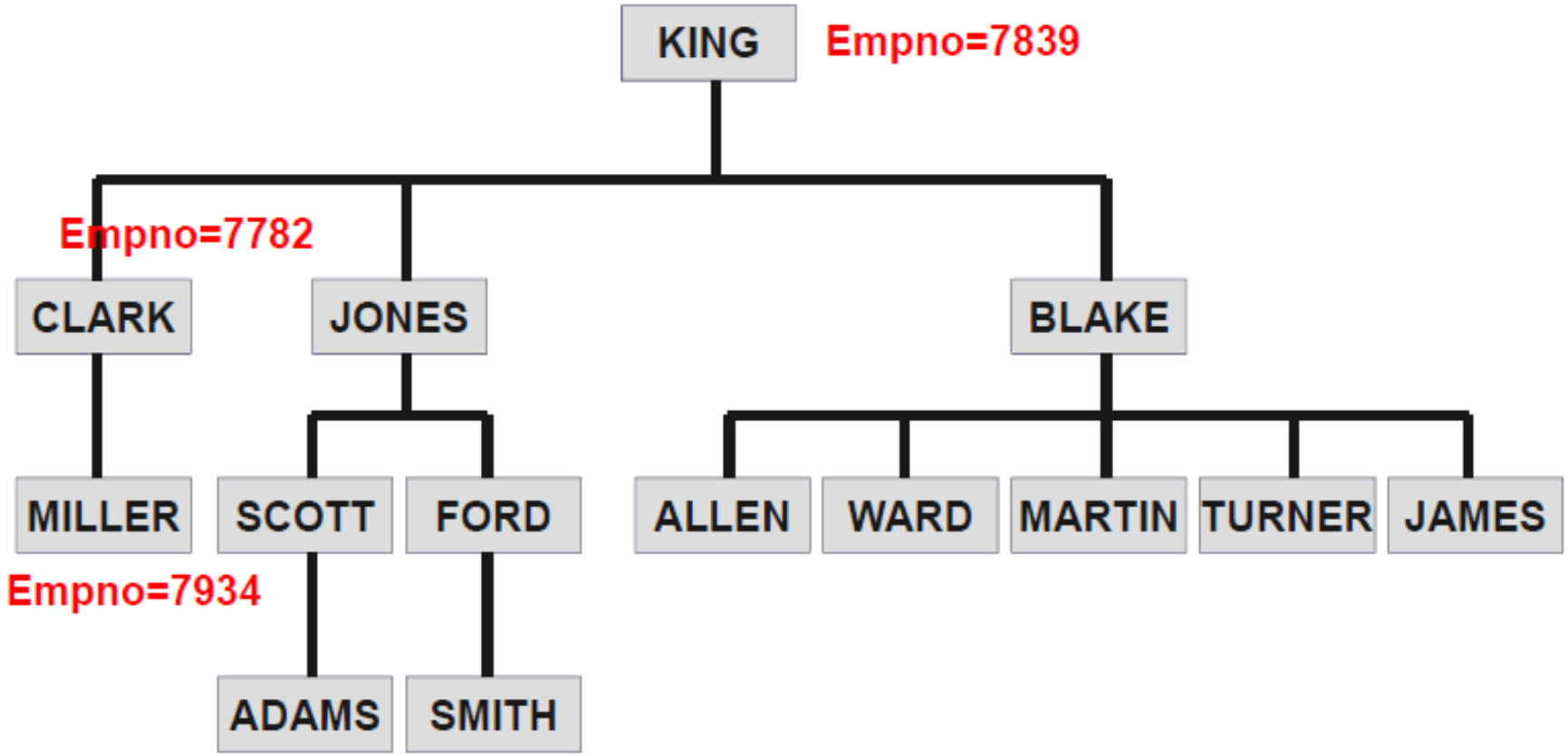
Single Table Inheritence (MongoDB)

```
> 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

계층형 데이터 구조

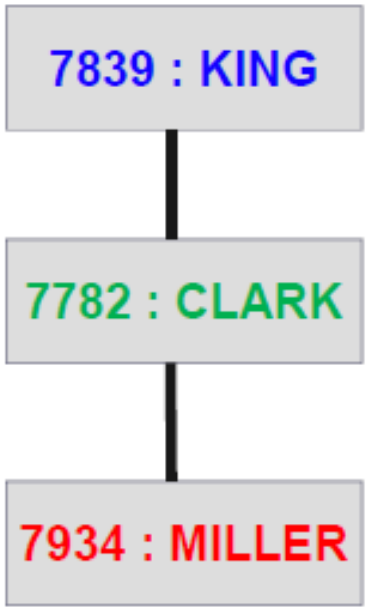

KING
Empno=7839
Empno=7782
CLARK
JONES
BLAKE
MILLER
SCOTT
FORD
ALLEN
WARD
MARTIN
TURNER
JAMES
Empno=7934
ADAMS
SMITH

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)




```
> 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 모델.

2) ANCESTORS와 **PARENT Field**로 표현할 수 있으며 하나의 **Document**는 하나 이상의 **ANCESTORS 와 PARENT를 가질 필요는 없다**

Vallidator

- RDBMS의 제약조건 역할
- MongoDB v3.2에 추가

Constraints	Validators
Primary key	Object_ID
Foreign key	Link/Embedded/Document
Not null	Validator
check	Validator
unique	Unique index

```
db.createCollection("emp",
{ validator : { $and : [{ empno : {$type :"string"}},
                         { deptno : {$in : [10,20,30]}}]}})

db.emp.insert({empno : "1111", ename : "JMJOO", deptno : 10})

db.emp.insert({empno : "1112", ename : "JMJOO", deptno : 20})
db.emp.insert({empno : "1113", ename : "JMJOO", deptno : 30})
db.emp.insert({empno : "1114", ename : "JMJOO", deptno : 40})


ddb.createCollection("emp",
{ validator : { $and : [{ empno : {$type :"double"}},
                         { ename : {$type :"string"}},
                         { job : {$type :"string"}},
                         { sal : {$type :"double"}},
                         { hiredate : {$type : "date"}},
                         { deptno : {$type : "double"}},
                         { deptno : {$in : [10,20,30]}}]}})

db.emp.insert({empno : 1111, ename : "JMJOO", job : "MANAGER",
          sal : 1200, hiredate : ISODate(), deptno : 10 })
db.emp.insert({empno : 2222, ename : "JMJ", job : "MANAGER",
                      sal : 1200, hiredate : ISODate(), deptno : 40 })
```