NoSQL의 특징

1. 데이터 간의 관계를 정의하지 않음
    - 데이터의 관계를 외래키 등으로 정의해 이를 이용, 관계형 연산을 하는 RDBMS와 달리 NoSQL은 데이터 간의 관계를 정의하지 않음
    - RDBMS에 비해 훨씬 더 대용량의 데이터를 저장할 수 있음
        - 태생이 RDBMS의 복잡도와 용량 한계를 극복하기 위한 만큼, 페타바이트 급의 대용량 데이터 저장 가능

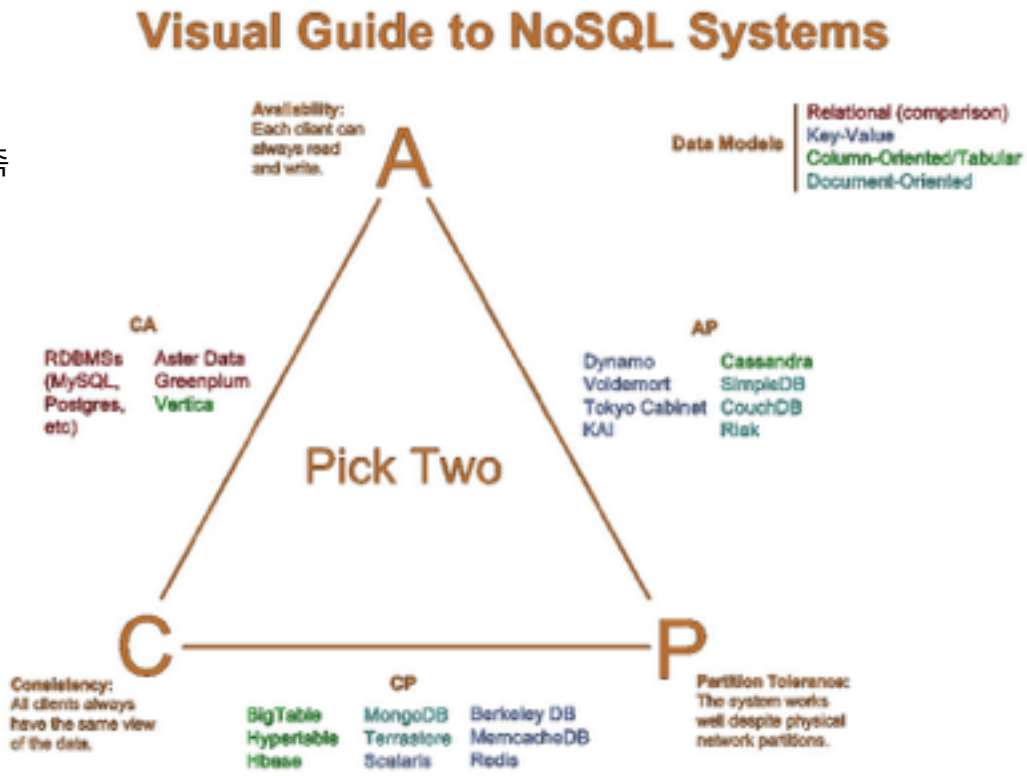
2. 분산형 구조
    - 일반적인 x86서버를 수십대 연결해 데이터를 저장 및 처리하는 구조
    - 분산 시 데이터를 상호 복제해 특정 서버의 장애에도 데이터 유실이나 서비스 중지가 없음

3. 고정되지 않은 테이블 스키마
    - 유동적인 스키마
    - 키 부분에만 타입이 동일하고 값에 해당하는 칼럼은 어떤 타입이든 어떤 이름이든 모두 허용

CAP 이론

분산 컴퓨팅 환경 3가지 특성

- 2002년 버클리 대학의 에릭 브루어 교수가 발표
- 일관성, 가용성, 분산 허용의 3가지 특징 중 두가지만 만족 할 수 있다는 이론

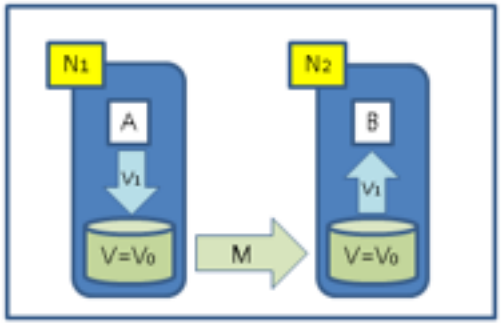


Consistency: 분산된 노드 중 어느 노드로 접근하더라도 데이터 값이 같음

Availability: 클러스터링 노드 중 하나 이상의 노드가 실패 하더라도 정상적으로 요청을 처리할 수 있음

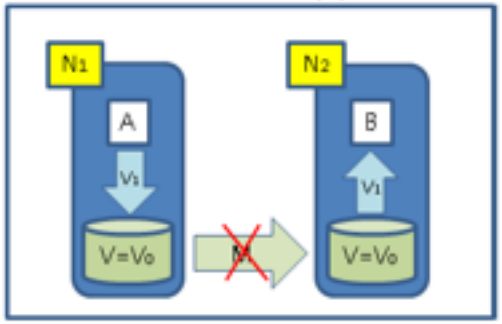
Partition Tolerance: 클러스터링 노드 간에 통신하는 네트워크가 장애가 나더라도 정상적으로 서비스를 수행할 수 있음

CAP 이론



– 네트워크가 N1, N2로 구분된 분산환경이다.
– 각 DB 노드는 V=V0이라는 값을 가지고 있다.
– 각 네트워크에는 A, B라는 클라이언트가 존재한다.
– A는 V=V1이라고 쓰고 B가 그것을 읽는다.

이런 환경에서 메시지 전달 과정(M)에서 문제가 생겼을 때..



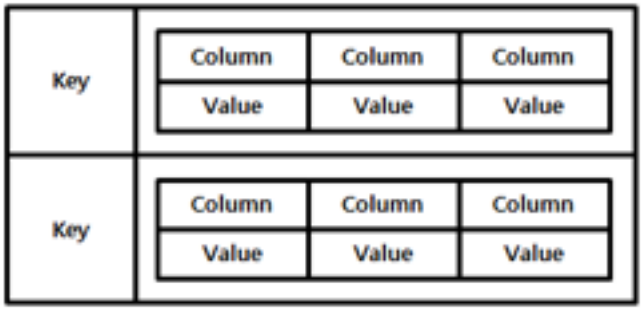
1. "C"가 꼭 필요한 상황인 경우
– A가 V1이라고 썼기 때문에 B는 V1이라고 읽을 수 있어야만 한다.
– A의 쓰기 동작은 M이 복구되기 전까지는 성공할 수 없다.
– M이 복구되기 전까지는 A의 Write는 block되거나 실패해야 한다. = Availability가 없음 = CP
– M이 문제가 생길 수 없도록 구성 = Partition-Tolerance가 필요 없음 = CA

2. "A"가 꼭 필요한 상황인 경우
– 어떤 경우에도 서비스가 Unavailable하면 안된다.
– A와 B가 꼭 동일한 데이터를 읽을 필요는 없음 = AP
– M이 문제가 생길 수 없도록 구성 = Partition-Tolerance가 필요 없음 = CA

3. "P"가 꼭 필요한 상황인 경우
– 메시지 전달 과정(M)에서 문제가 생기더라도 시스템에 영향이 가서는 안된다.
– A와 B가 꼭 동일한 데이터를 읽을 필요는 없음 = AP
– A의 쓰기 동작은 M이 복구되기를 기다린다. = 그동안 쓰기 서비스 불가능 = Availability가 없음 = CP

NoSQL의 분류

1. Key/Value Store
   - 가장 기본적인 패턴, 대부분의 NoSQL이 다른 데이터 모델을 지원하더라도 기본적으로 Key/Value 개념을 지원
   - 고유한 Key에 하나의 값을 가진 형태
   - Put(Key, Value), Value := get(Key) 형태의 API로 접근
   - Column Family
     - ➢ Key 안에 (column, value) 조합으로 여러 개의 필드를 갖음(ex: Key-이름, Column-주소, Value-값)

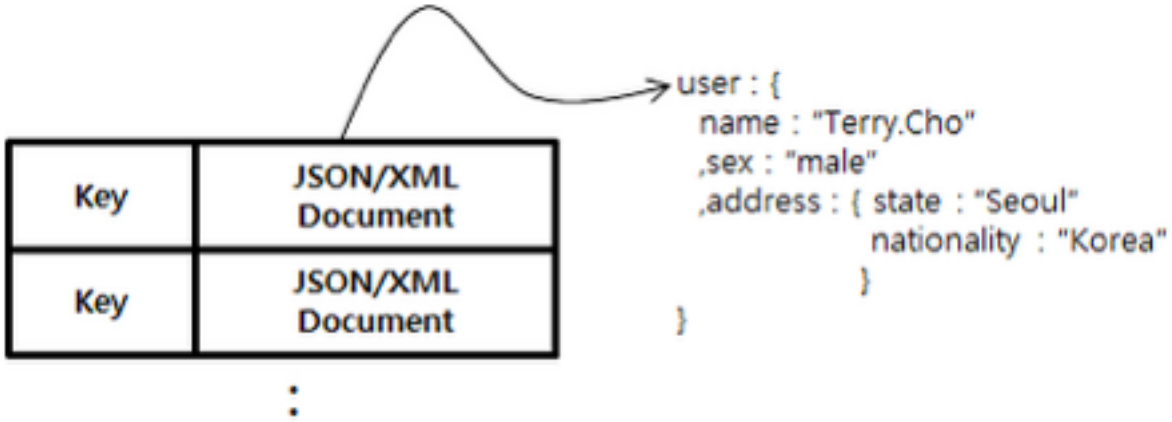


[Column family]

2. Ordered Key/Value Store
   - Key/Value Store의 확장된 형태로 데이터 저장 방식은 동일하나, 내부적으로 키를 순서로 정렬해서 저장
   - NoSQL은 RDBMS의 ORDER BY와 같은 기능을 제공하지 않기 때문에 결과 값을 업데이트 날짜 등으로 정렬해서 보여줄 때 Ordered Key/Value Store가 절대적으로 유리
   - 대표적인 제품으로 Hbase, Cassandra

NoSQL의 분류

3. Document Key/Value Store
   - Key/Value Store의 확장된 형태
   - 키에 해당하는 값의 데이터 타입으로 Document라는 타입을 사용
   - Document는 XML, JSON, YAML과 같이 구조화된 데이터 타입을 일컬음
   - 복잡한 계층 구조를 표현할 수 있음
   - Document Store 기반의 NoSQL은 제품에 다소 차이가 있으나 대부분 Sorting, Join, Grouping등의 추가적인 기능을 제공함 (MongoDB, CouchDB, Riak)
   - 그래프나 트리 구조와 같은 데이터 구조를 저장 하는데 최적화된 Neo4j같은 NoSQL도 있음



NoSQL과 기존 RDBMS와의 차이

1. NoSQL은 데이터를 저장하고 키에 대한 Put/Get만 지원
    1. Put: Insert into TABLE VALUES(KEY, value1, value2, …, valuen)
    2. GET: Select * from TABLE where KEY=“key”

2. 고민해야 하는 기능
    1. Sorting (SQL의 ORDER BY)
    2. Join (RDBMS에서 두 개 테이블의 외래 키를 이용하여 조인)
    3. Grouping (SQL의 Group BY)
    4. Range Query (where key > “start” and key < “end”와 같이 일정 범위 내의 내용 쿼리)
    5. Index (RDBMS에 인덱스를 지정하여 SELECT 쿼리 성능을 높이는 기능)

NoSQL 사용 시 주의할 점

1. 과연 NoSQL이 필요한가?
    - 교육과 개발 비용들이 절대 저렴하지 않음
    - NoSQL을 공부하는 데 시간이 들고 운영하면서 수많은 장애들이 발생할 수 있음
    - MySQL등 기존의 익숙한 기술을 사용하다 더 큰 용량이 필요할 때 전환을 고려하는 것도 좋은 접근
2. 적절한 NoSQL 제품군의 선정
    - RDBMS만 아니면 모두 NoSQL. 따라서 각 NoSQL의 특성을 잘 파악해 업무와 팀의 특성에 맞는 제품을 선정하는 것이 첫 걸음
3. 데이터모델링
    - RDBMS와 NoSQL의 데이터 모델링 방식은 정반대
    - NoSQL은 데이터 모델링이 설계의 90%
4. RDBMS와 적절한 혼합
    - NoSQL은 데이터를 저장하기는 하지만, 복잡한 데이터 저장이나 쿼리는 한없이 부족
    - 대용량 데이터를 빠르게 저장 및 쿼리하기 위해 최적화된 기술
5. 하드웨어 설계 병행
    - NoSQL을 사용할 때는 처음부터 Disk Raid 구성, Network등의 하드웨어 설계를 병행하는 것이 좋음
6. 운영 및 백업
    - 전문적인 관리자가 있어야 하며, 시스템에 대한 체계적인 모니터링 방안 등의 운영 체계 수립

NoSQL 데이터 모델링

1. NoSQL 디자인 패턴
    1. NoSQL의 데이터 모델링이란 NoSQL에 저장할 데이터들의 구조, 즉 테이블을 설계하는 것을 의미.
    2. RDBMS와는 그 특성이 매우 달라 다른 방식으로 접근 필요
2. NoSQL과 RDBMS의 데이터 모델링 차이

    1. NoSQL을 사용해서 데이터 모델링을 하려면 근본적인 사상 2가지를 바꿔야 함
        ① 개체 모델 지향에서 쿼리 결과 지향 모델링
            - 기존 RDBMS 도메인 모델 순서 [테이블 -> 쿼리]
            - NoSQL에서 도메인 모델 순서 [쿼리 -> 테이블]
            - NoSQL의 경우 복잡한 쿼리 기능이 없기 때문에 도메인 모델에서 어떤 쿼리 결과가 필요하지를 정의 후 쿼리 결과를 얻기 위한 데이터 저장 모델을 디자인 해야 함

        ② 정규화에서 역정규화
            - 기존 RDBMS 같은 데이터가 두 개 이상의 테이블에 중복 저장하는 것을 제거
            - NoSQL에서는 쿼리의 효율성을 위해 데이터를 정규화 하지 않고 의도적으로 중복된 데이터를 저장하는 등의 비정규화된 데이터 모델 설계 방식으로 접근

NoSQL 데이터 모델링

1. 기본적인 데이터 모델링 패턴
    1. 역정규화(Denormalization)
        - 데이터를 중복해서 저장하는 방식. 테이블 간의 조인을 없앨 수 있음
        - NoSQL은 조인을 지원하지 않기 때문에 어플리케이션에서 처리되어야 함

User Table

Name(PK)	Age	Sex	City (FK)

City Table

City(Pk)	State	ZipCode

역정규화

UserZipCode

User(PK)	Age	Sex	zipCode

RDBMS

→ select u.name, u.age, u.sex, c.zipcode
from user u, city c where u.city = c.city

NoSQL

→ select $city, name, age, sex from user where userid="사용자ID"

→ select zipcode from city where city=$city

더 많은 IO가 발생

장점

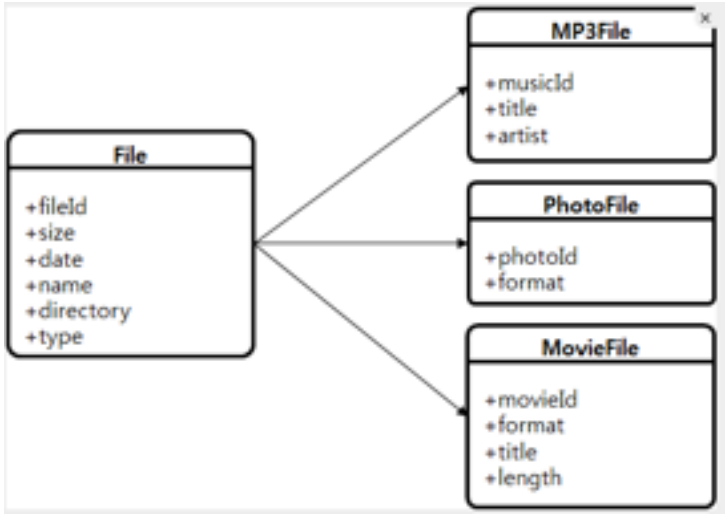
- 성능향상: 조인을 위한 쿼리 횟수 감소
- 쿼리 로직의 복잡도 감소

단점

- 데이터 일관성 문제 발생 가능
- 스토리지 용량 증가

NoSQL 데이터 모델링

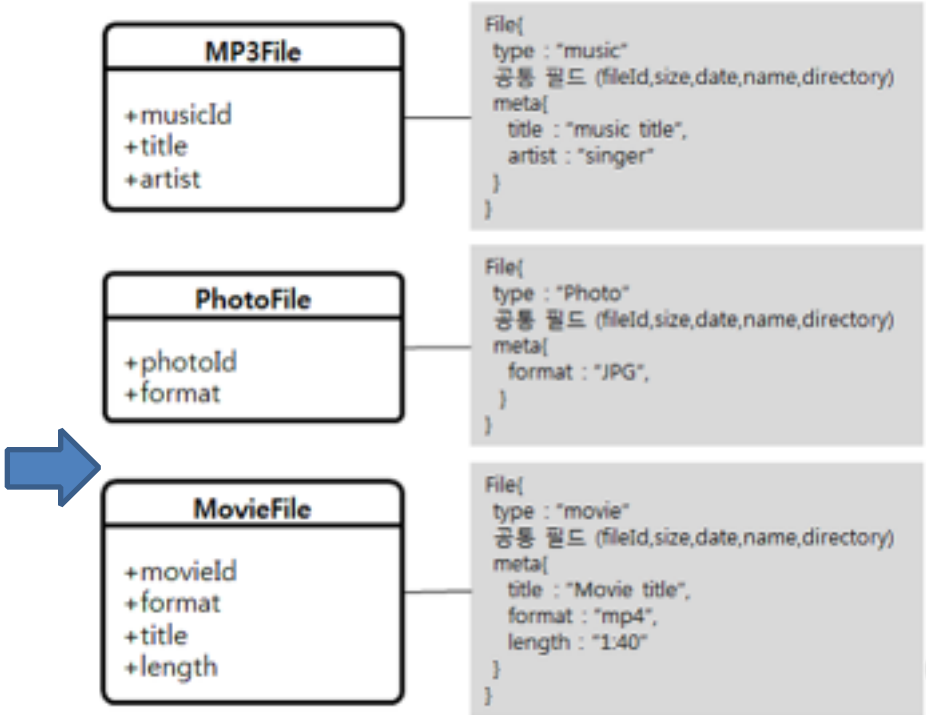
2. 집계(Aggregation)
    - NoSQL의 특성 중 하나인 Schemeless 또는 Soft Scheme
    - RDBMS와는 달리 키만 같다면 각 행은 제멋대로



드롭박스나 처럼 파일을 저장하는 개인 스토리지 서비스의 ERD

집계를 적용한 타입별 파일 테이블 구조

- 1:N과 같은 복잡한 엔티티들의 관계를 손쉽게 하나의 테이블로 변경
- 조인의 수를 줄여 쿼리성능 향상



NoSQL 데이터 모델링

1. 확장된 데이터 모델링 패턴
    1. Atomic Aggregation
        - NoSQL에서 원자성을 보장하기 위한 방법
        - 트랜잭션 보장을 통한 데이터 불일치성을 해결하기 위함
        - 구현 패턴상으로는 Aggregation과 동일하나 Aggregation은 Join을 없애기 위함

1번 테이블 업데이트 후 장애로 인해 2번 업데이트 불가

데이터 일관성 문제 발생

1번 테이블 업데이트 후 장애로 인해 2번 업데이트 불가

사용자 정보에 대한 Atomic Aggregation 패턴 예제

NoSQL 데이터 모델링

2. Index Table
    1. NoSQL은 RDBMS와 달리 인덱스가 없기 때문에 키 이외의 필드를 이용해 검색하면 테이블을 전체 스캔
    2. 인덱스를 위한 별도의 인덱스 테이블을 만들어 사용할 수 있음

FILE TABLE

fileID	filename	directory (index)	fileLocation

DIRECORY INDEX TABLE

directory	files
	fileid, fileid, fileid, fileid, fileid .....

NoSQL 데이터 모델링

3. Composite Key
    1. 특징
        1. 하나 이상의 필드를 쌍점 구분자(:)를 이용해 구분 지어 사용하는 방법
        2. NoSQL에서는 이 키를 어떻게 정의 하느냐가 매우 중요
        3. Ordered Key/Value Store의 경우 이를 이용해 ORDER BY와 같은 정렬 기능이나 그룹화를 구현할 수 있음
        4. RDBMS의 복합키와 같은 개념이나, RDBMS는 여러 개의 칼럼을 묶어서 PK로 지정하지만 NoSQL은 한 칼럼에 구분자를 이용해 여러 개의 묶어서 사용

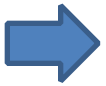
key	value
windows:etc	....
windows:programfile	....
windows:system32	....
windows:temp	....
msoffice:msword	....
msoffice:powerpoint	....
adobe	....

NoSQL의 특성상 N개의 클러스터 구성시 데이터는 키를 기준으로 N개의 서버에 나뉘어 저장
키 값을 선택할때 “특정 서버로의 몰림 현상”을 주의해야 함

NoSQL 데이터 모델링

4. Inverted Index
    1. 검색 엔진에서 많이 사용하는 방법
    2. Value에 검색 키워드들이 들어 있을 경우 효과적인 검색을 할 수 없기 때문에 검색 키워드를 키로, URL을 Value로 하는 테이블을 다시 만들어 사용

key	value
bcho.tistor.com/nosql	nosql,cassandra,riak
bcho.tistory.com/cloud	amazon,azure,google
facebook.com/group/serverside	amazon,google,riak
highscalability.com/bigdata	nosql,riak
www.basho.com/riak	riak



key	value
riak	bcho.tisoty.com/nosql facebook.com/group/serverside highscalability.com/bigdata www.basho.com/riak
nosql	highscalability.com/bigdata bcho.tisoty.com/nosql

5. Enumerable Keys
    1. RDBMS의 Sequence와 같은 기능을 제공하는 것
    2. 키에 대해서 자동으로 카운터를 올려주는 역할

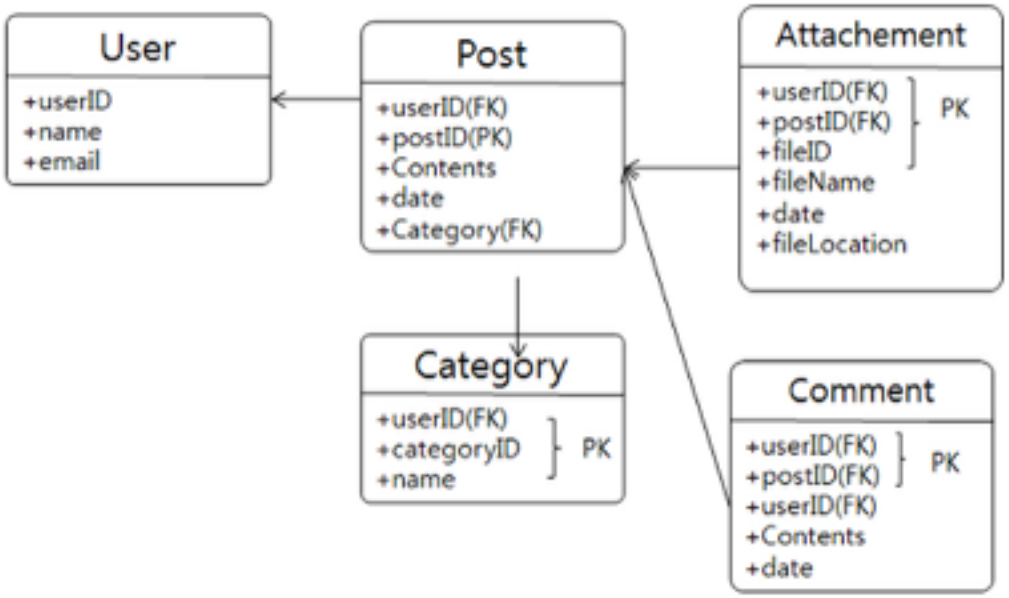
NoSQL 데이터 모델링

6. NoSQL을 이용한 시스템 개발시 주의 사항

- 1. 데이터 모델링이 80% 이상. 선정한 NoSQL과 애플리케이션의 특성에 맞는 데이터 모델링에 집중
- 2. NoSQL은 어떤 솔루션이 좋다, 나쁘다가 없다. 반드시 데이터 모델과 내부 아키텍처 두 가지를 파악한 후 애플리 케이션의 특성에 맞는 NoSQL을 선정
- 3. 하나의 NoSQL로 전체 데이터를 저장하려고 하지 마라.

NoSQL 데이터 모델링 절차

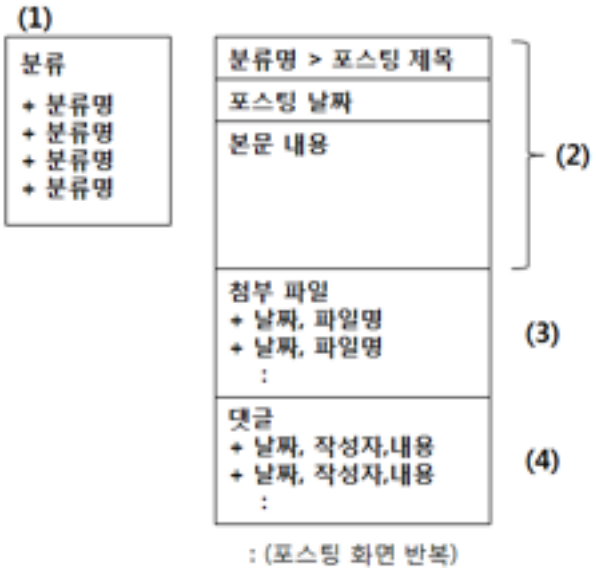
1. 도메인 모델 파악 예
    1. 블로그 시스템의 데이터 모델
        1. 사용자 ID 기반으로 블로그의 분류(Category)를 가지고 있고
        2. 분류별로 글을 작성할 수 있으며
        3. 글에 파일을 첨부할 수 있고
        4. 댓글을 달 수 있는 블로그이다



블로그 시스템의 데이터 저장 구조 ERD

NoSQL 데이터 모델링 절차

2. 쿼리 결과 디자인(데이터 출력 형태 디자인)
    1. 블로그 시스템의 데이터 모델



1. 왼쪽 화면 위에는 블로그 사용자의 포스팅 분류명들을 목록식으로 출력

2. 상단에 포스팅의 분류명과 제목을 출력하고, 다음 포스팅 날짜, 본문을 출력

```
Select po.postID, po.Contentes, po.date, ca.name
from Category ca, Post po
where userID="사용자ID" order by date desc
```

3. 첨부 파일 업로드 날짜와 파일명을 차례대로 출력하고 파일에 대한 링크 출력

```
Select fileID, filename, date, fileLocation
From Attachment
where userID="사용자ID" and postID="현재 포스팅 ID"
Order by date desc
```

4. 댓글에는 작성일, 작성자 이름과 댓글 내용을 출력하고 이름에는 이메일 링크

```
Select userID, email, Contents, date
From Comment
Whhere userID="사용자ID" and postID="현재 포스팅  ID"
Order by date desc
```

Key : userID

CategoryID	Name (분류명)

Key : userID

po.postID	ca.name(분류명)	po.date(포스팅날짜)	Po.Conents(내용)

내림차순
(날짜)

Key : userId,postId

fileID	filename	date	fileLocation

내림차순
(날짜)

Key : userId,postId

userId(댓글)	email	Contents(댓글)	date

내림차순
(날짜)