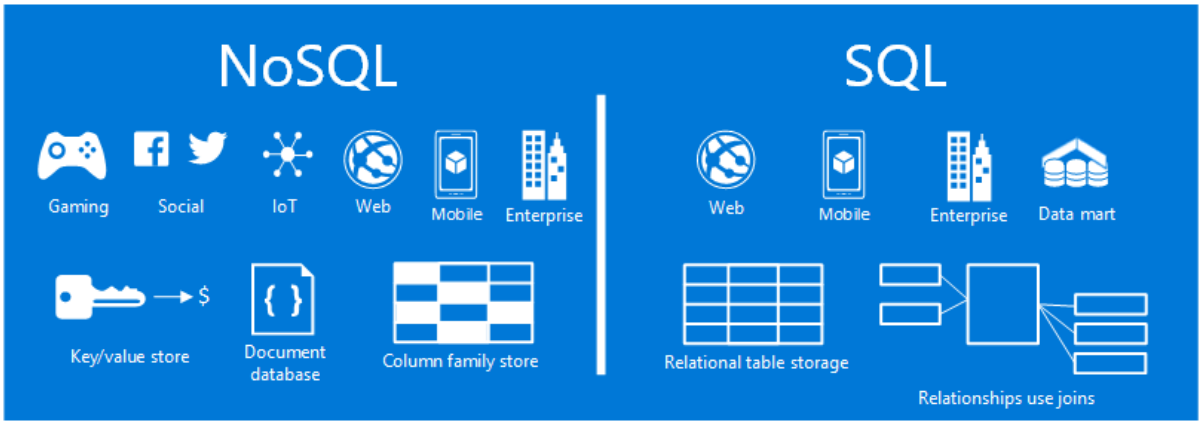
MongoDB는 NoSQL로 분류되는 크로스 플랫폼 도큐먼트 지향 데이터베이스 시스템이다. MySQL 처럼 전통적인 테이블-관계 기반의 RDBMS가 아니며 SQL을 사용하지 않는다. 이름의 mongo는 humongous를 줄인 표현이다. 즉 '겁나 큰 DB' 라는 뜻이다.

MongoDB는 MySQL(MariaDB)의 테이블과 같이 스키마가 고정된 구조 대신 JSON 형태의 동적 스키마형 문서를 사용하는데, 이를 MongoDB 에서는 BSON이라고 부른다.

MongoDB는 가장 기본적인 데이터를 Document 라고 부른다. 이는 MySQL(MariaDB)같은 RDBMS에서의 row 에 해당된다. 이 Document의 집합을 Collection이라고 하는데, RDBMS에서는 Table에 해당된다. Collection의 집합은 DB이며, 이것은 RDBMS에서도 동일하다.

RDBMS(SQL)	NoSQL
데이터베이스	데이터베이스
테이블	콜렉션
로우(행)	도큐먼트



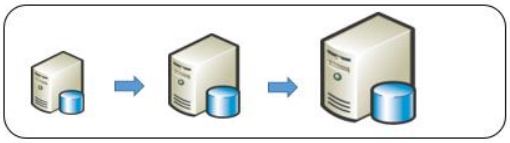
똑같은 조건으로 설계되었을 시 기존 RDBMS 보다 속도가 굉장히 빠르다는 장점이 있다. 이런 속도는 ACID(트랜젝션)를 포기한 댓가로 얻은 것이다. 따라서 데이터 consistency가 거의 필요 없고 조인 연산을 embed로 대체할 수 있는 경우에는 MongoDB가 확실한 대안이 될 수 있다. 반대로 저장되는 데이터가 은행 데이터 같이 consistency가 매우 중요한 작업에는 MongoDB를 쓰지 않는다.

SQL, NoSQL은 서로 반대의 개념도, 경쟁상대도 아니다. 많은 회사들이 두 타입 모두 동시에 사용하기도 하기 때문이다. 하나의 시스템이 모든 경우에 만족된 서비스를 제공하지 못하기 때문이다. 만약 데이터가 급격하게 늘어나는 형태라면 NoSQL이 도움이 될 수 있다.

'NoSQL이 스케일링에 좋다고 하던데요'라는 게 무슨 말일까?

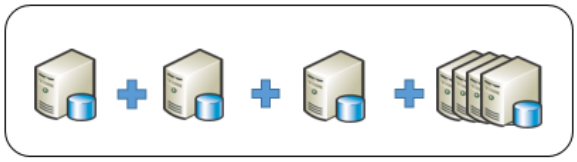
SQL의 경우엔 시스템이 커져가면서 Scale-Up의 형태로 DB를 증설하게 된다. 관계를 가지는 테이블들이 수평적으로 더 커진다는 말이다. 그렇게 되면 기존에 사용하던 DB시스템보다 고성능의 DB시스템이 필요할 수 있고 이는 굉장히 덩치가 큰 DB 시스템이 된다는 말이며 비용도 많이 증가될 뿐만 아니라 관리가 어려워질 수 있다.

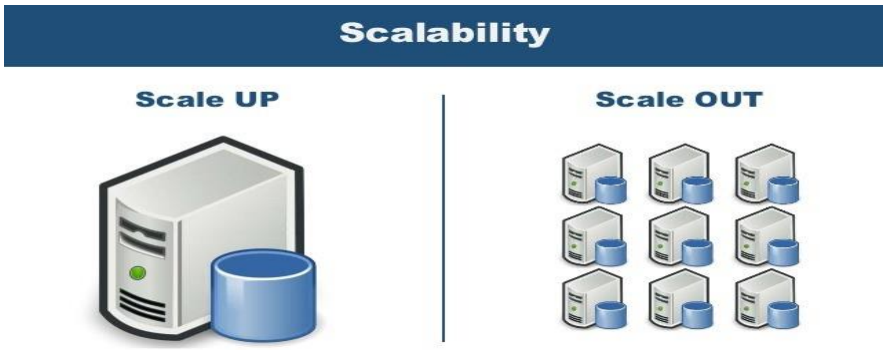
스케일업

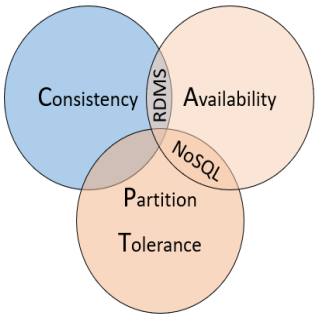


반면, NoSQL의 경우 Scale-Out의 형태로 증설을 할 수 있게 되는데 고성능의 DB를 갖추는 게 아니라 여러 DB 시스템으로 추가할 수 있다는 말이다. 숫자는 무한대로 늘려갈 수 있는데 이는 SQL의 Scale-Up이 무제한적으로 늘려갈 수 없는 것과 비교해 장점이 될 수 있다.

스케일아웃



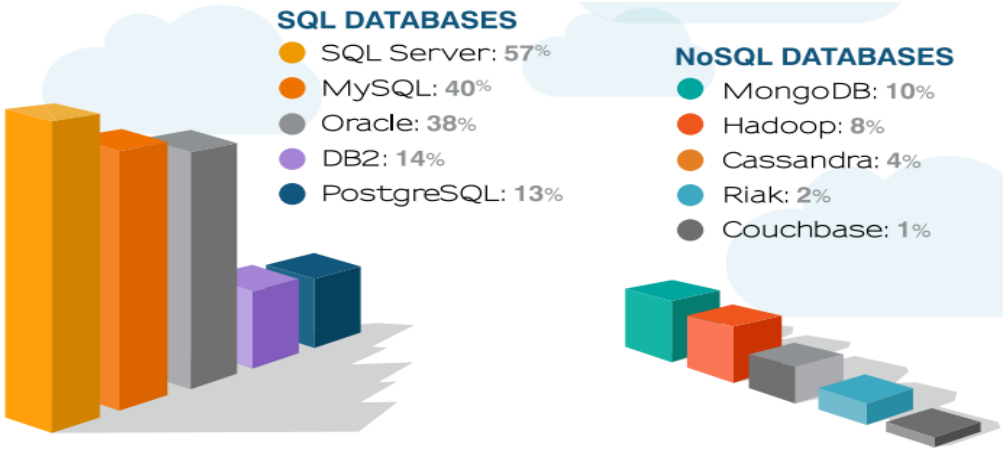




일관성(Consistency) : 모든 노드가 같은 순간에 동일한 데이터를 볼 수 있다.

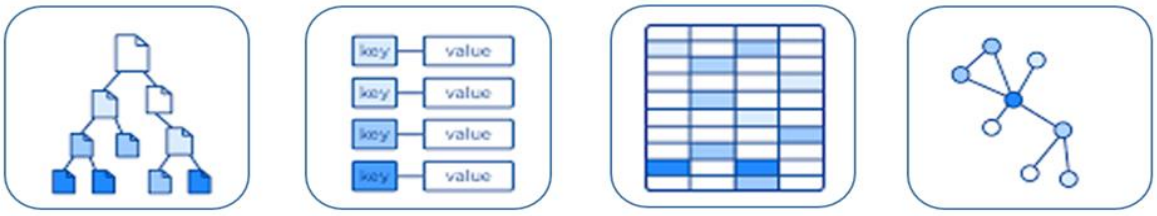
가용성(Availability) : 모든 요청이 성공 또는 실패 결과를 반환할 수 있다.

분할내성(Partition tolerance) : 메시지 전달이 실패하거나 시스템 일부가 망가져도 시스템이 계속 동작할 수 있다.

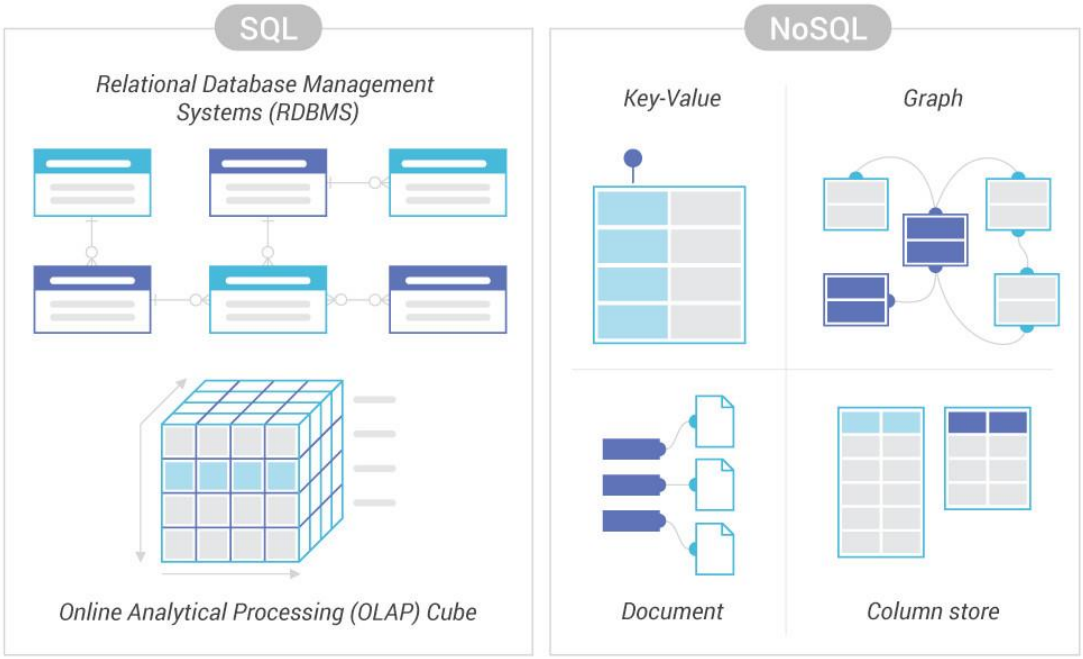


SQL	MongoDB
데이터베이스	데이터베이스
테이블	컬렉션
행	도큐먼트(BSON)
열	필드
테이블 조인	임베디드 도큐먼트 & 링킹(Linking)
인덱스	인덱스
주-키	주-키. 단 MongoDB에서는 주-키는 _id 필드로 자동 설정
집합(e.g. group by)	집합 프레임워크

[NoSQL의 데이터 모델]



모델	특징
문서 저장소	데이터 및 메타 데이터는 데이터베이스 내의 JSON 기반 문서에 계층적으로 저장
키 값 저장소	NoSQL 데이터베이스 중 가장 간단한 데이터는 키-값 쌍의 컬렉션으로 표시
Wide-Column 저장소	관련 데이터는 단일 열에 중첩 키/값 쌍의 집합으로 저장
그래프 저장소	데이터는 그래프 구조에 노드, 에지 및 데이터 속성으로 저장



[복습] MongoDB는.....

MongoDB는 C++로 작성된 오픈소스 문서지향(Document-Oriented) 적 Cross-platform 데이터베이스이며, 뛰어난 확장성과 성능을 자랑한다. 또한, 현존하는 NoSQL 데이터베이스 중 인지도 1위를 유지하고 있다.

NoSQL?

흔히 NoSQL이라고 해서 SQL이 없는 데이터베이스라고 생각 할 수도 있겠지만, 진짜 의미는 Not Only SQL 이다. 기존의 RDBMS의 한계를 극복하기 위해 만들어진 새로운 형태의 데이터저장소이다. 관계형 DB가 아니므로, RDMS처럼 고정된 스키마 및 JOIN 이 존재하지 않는다.

Document?

MongoDB에서의 Document는 RDMS의 record(row)와 비슷한 개념이다. 한 개 이상의 key-value pair 으로 이뤄져 있다. 다음은 MongoDB의 샘플 Document이다.

```
{
    "_id": ObjectId("5099803df3f4948bd2f98391"),
    "username": "velopert",
    "name": { first: "M.J.", last: "Kim" }
}
```

여기서 _id, username, name 은 key 이고 그 오른쪽에 있는 값들은 value 이다.

_id 는 12bytes의 hexadecimal 값으로서, 각 document의 유일함(uniqueness)을 제공한다.

이 값의 첫 4bytes는 현재 timestamp, 다음 3bytes는 machine id, 다음 2bytes는 MongoDB 서버의 프로세스id, 마지막 3bytes는 순차번호이다.

Document는 동적(dynamic)의 schema를 갖고 있다. 같은 Collection 안에 있는 Document끼리 서로 다른 schema 를 갖고 있을 수 있으며 서로 다른 데이터 (즉 다른 key) 들을 가지고 있을 수 있다.

Collection?

Collection은 MongoDB Document의 그룹으로서 Document들은 Collection내부에 위치하게 된다. RDMS의 table과 비슷한 개념이지만 RDMS와 달리 schema를 따로 가지고 있지 않다.

Database?

Database는 Collection들의 물리적인 컨테이너이며 각 Database는 파일시스템에 여러 파일들로 저장된다.

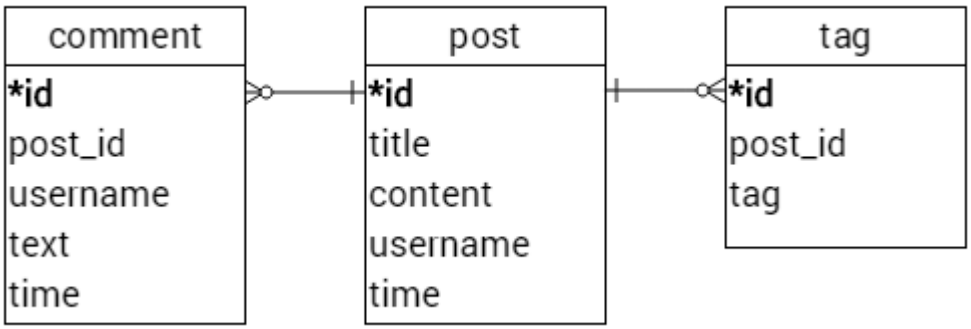
[schema 디자인 할 때 고려사항]

여러 객체들을 함께 사용해야 된다면 하나의 Document에 합쳐서 사용한다. (예: 게시물-댓글 과의 관계)

간단한 블로그를 위한 데이터베이스를 디자인한다고 가정해 본다.

- 게시글 : 작성자 이름, 제목, 내용
- 태그 : 각 게시글마다 0개 이상의 태그
- 댓글 : 각 게시글마다 0개 이상의 댓글을 가짐. 작성자 이름, 내용, 작성시간

RDMS에서 데이터베이스 schema를 디자인한다면 다음 그림과 비슷한 구조를 갖게 된다.



RDMS 의 경우에는 위와 같이 테이블을 3개로 나눠야 효율적이라고 부르지만, NoSQL 에서는 다음과 같이 모든 것을 하나의 Document 에 넣는다.

{

```
 _id: POST_ID,
 title: POST_TITLE,
 content: POST_CONTENT,
 username: POST_WRITER,
 tags: [ TAG1, TAG2, TAG3 ],
 time: POST_TIME
 comments: [
 {
 username: COMMENT_WRITER,
 mesage: COMMENT_MESSAGE,
 time: COMMENT_TIME
 },
 {
 username: COMMENT_WRITER,
 mesage: COMMENT_MESSAGE,
 time: COMMENT_TIME
 }
 ]
}
```

도큐먼트 페이지 : https://docs.mongodb.com/v4.0/reference/method/db.collection.find/

[Database 생성 : use DATABASE_NAME]

```
> use testdb_tutor
switched to db testdb_tutor
> db
testdb_tutor
> show dbs
```

본인의 리눅스 계정명

리스트에서 바로 전에 만든 데이터베이스를 보려면 최소 한개의 Document를 추가해야 한다.

```
> db.book.insert({"name": "python", "price": 10000})
> show dbs
> show collections
> db.book.insert({"name": "java", "price": 12000})
> db.book.insert({"name": "javascript", "price": 8000})
> db.book.insert({"name": "html5", "price": 8000})
> db.book.insert({"name": "django", "price": 20000})
```

> db.book.insert({"name": "spark", "price": 25000})

> db.book.find()

> db.book.find().pretty()

> db.book.find().sort({"price": 1})

> db.book.find().sort({"price": -1})

> db.book.find().sort({"price": -1, "name": 1})

> db.book.find({"name": "javascript"})

> db.book.find({"name": { "$regex" : "^java"}})

> db.book.find({"price": 8000})

> db.book.find({"price": { "$gt" : 8000}})

> db.book.find({},{'name' : 1})

> db.book.find({},{'name' : 1, '_id' : 0})

[Document 조회 : db.COLLECTION_NAME.find(query, projection)]

parameter	Type	설명
query	document	Optional(선택적). 다큐먼트를 조회할 때 기준을 정한다. 기준이 없이 컬렉션에 있는 모든 다큐먼트를 조회할 때는 이 매개변수를 비우거나 비어있는 다큐먼트 { } 를 전달한다.
projection	document	Optional. 다큐먼트를 조회할 때 보여질 field를 정한다.

[반환(return) 값 : Cursor 객체]

query에 해당하는 Document들을 선택하여 cursor를 반환한다. cursor 는 query 요청의 결과값을 가르키는 pointer 이다. cursor 객체를 통하여 보이는 데이터의 수를 제한 할 수 있고, 데이터를 sort 할 수 도 있다. 이는 10분동안 사용되지 않으면 만료된다.

[Query 연산자]

프로그래밍 언어에서 >, <, <=, ==, != 등 연산자가 있는 것 처럼, MongoDB 에서도 원하는 데이터를 찾기 위해 연산자를 사용한다. 연산자의 종류는 비교(Comparison), 논리(Logical), 요소(Element), 배열(Array) 등 여러 종류가 있다.

비교(Comparison) 연산자

$eq (equals) 주어진 값과 일치하는 값

$gt (greater than) 주어진 값보다 큰 값

$gte (greather than or equals) 주어진 값보다 크거나 같은 값

$lt (less than) 주어진 값보다 작은 값

$lte (less than or equals) 주어진 값보다 작거나 같은 값

$ne (not equal) 주어진 값과 일치하지 않는 값

$in 주어진 배열 안에 속하는 값

$nin 주어빈 배열 안에 속하지 않는 값

논리 연산자

$or 주어진 조건중 하나라도 true 일 때 true

$and 주어진 모든 조건이 true 일 때 true

$not 주어진 조건이 false 일 때 true

$nor 주어진 모든 조건이 false 일때 true

$regex 정규표현식에 매칭되는 값

[Database 제거 : db.dropDatabase()]

이 명령어를 사용하기 전, use DATABASE_NAME 으로 삭제하고자 하는 데이터베이스를 선택한다.

> use 데이터베이스명

> db.dropDatabase();

> show dbs

[Collection 생성 : db.createCollection()]

Collection을 생성할 때는 db.createCollection(name, [options]) 명령어를 사용한다. 또한 콜렉션은 도큐먼트를 최초로 저장할 때 자동으로 생성되므로 직접 생성하지 않아도 된다.

> use test

switched to db test

> db.createCollection("game")

> show collections

[Collection 제거 : db.COLLECTION_NAME.drop()]

> use test

switched to db test

> show collections

> db.컬렉션이름.drop()

> show collections

[Document 추가 : db.COLLECTION_NAME.insert(document)]

> db.book.insert({"name": "CSS3", "price": 17000})

> db.book.insert([

{"name": "pandas", "price": 21000},

{"name": "R", "price": 23000}

])

> db.book.find()

[Document 제거: db.COLLECTION_NAME.remove(criteria, justOne)]

parameter	type	설명
*criteria	document	삭제할 데이터의 기준 값(criteria)이다. 이 값이 { } 이면 컬렉션의 모든 데이터를 제거한다.
justOne	boolean	선택적(Optional) 매개변수이며 이 값이 true 면 1개 의 다큐먼트만 제거한다. 이 매개변수가 생략되면 기본값은 false 로서, criteria에 해당되는 모든 다큐먼트를 제거한다.

> db.book.find({"name": "R"})

> db.book.remove({"name": "R"})

> db.book.find()

[CSV 파일을 MongoDB 에 컬렉션으로 저장하기]

mongoimport -d imsi -c 컬렉션명 --type csv --file csv파일명 --headerline

mongoimport -d imsi -c emp --type csv --file data/emp.csv --headerline