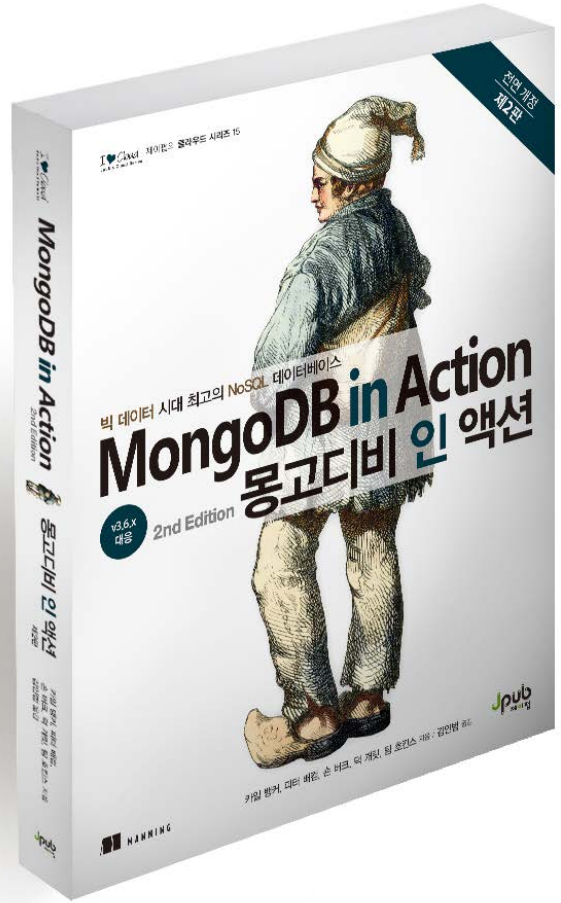
[몽고디비 인 액션] 1,2장

19. 9. 21 (토)



[몽고디비 인 액션 2nd Edition]

github.com/bakks/mongo-in-action-code

1장

최신 웹을 위한 도큐먼트 데이터베이스

RDBMS vs. NoSQL

RDBMS

- 관계 데이터베이스(relational database)
- 전통적인 데이터베이스
- RDBMS와 상호작용하는 쿼리 언어를 SQL(Structured Query Language)이라고 부름
- 대표적으로 MySQL이 있음

NoSQL

- 비관계 데이터베이스(nonrelational database)
- RDBMS의 대안으로 등장
- SQL과 다른 쿼리를 사용하기 때문에 포괄적 용어로 NoSQL(Not Only SQL)이라고 부름
- 그중 MongoDB는 가장 인기 있는 NoSQL 데이터베이스

MongoDB

- MongoDB란?

: 웹 애플리케이션과 인터넷 기반을 위해 설계된 DBMS

- MongoDB를 사용하는 이유?

① 높은 읽기·쓰기 효율

② 자동 장애조치(failover)를 통한 확장의 용이성

③ **직관적인 데이터 모델** ✓

직관적인 데이터 모델?

- 정보를 행(row) 대신 도큐먼트(document)에 저장한다.
  - 도큐먼트 형식은 JSON(JavaScript Object Notation)에 기반한다.
  - 키(key)와 키에 대한 값(value)으로 이루어져 있으며, 중첩에 제한이 없다.

- 풍부하고 계층적인 구조의 데이터를 표현할 수 있다.
  - 복잡한 조인 연산이 없어도 된다.
  - 데이터 변경 시에 스키마를 맞춰야 할 필요가 없다.
  - 대부분의 정보를 하나의 도큐먼트로 쉽게 표현할 수 있다.
  - 도큐먼트에 대해 질의, 조작 연산도 할 수 있다.

1.1 MongoDB의 역사

- software platform-as-a-service
  - 2007년 10gen이라는 신생 기업에서 개발한 소프트웨어 플랫폼
  - 웹 애플리케이션을 호스트하고 필요하면 확장할 수 있는 애플리케이션 서버, 그리고 데이터베이스로 구성
  - 이후에 (MongoDB가 된) 데이터베이스에만 집중해서 개발

- 매우 간단하지만 유연한 웹 애플리케이션 계층의 일부

1.2 MongoDB의 핵심 기능

1.2.1 도큐먼트 데이터 모델
1.2.2 애드혹 쿼리
1.2.3 인덱스 => 8장
1.2.4 복제 => 8장
1.2.5 속도와 내구성 => 11장
1.2.6 확장 => 12장

1.2.1 도큐먼트 데이터 모델

- 도큐먼트 지향적인 데이터베이스
  - 도큐먼트(document) 속성의 이름과 값으로 이루어진 쌍의 집합
  - 속성의 값은 배열이나 다른 도큐먼트가 될 수 있음
  - 컬렉션(collection) 데이터를 컬렉션에 도큐먼트로 저장
  - 즉 도큐먼트 < 컬렉션 < 데이터베이스

- 기존 RDBMS의 특징
  - 정규화를 통해 하나의 데이터를 하나의 테이블로만 표현할 수 있다.
  - 정규화(normalization) 한 객체(object)의 데이터를 여러 테이블로 나누어 표현하는 기법
  - 즉 데이터를 정해진 스키마(ex. 테이블)에 집어넣는 과정이 필요하다.

1.2.1 도큐먼트 데이터 모델

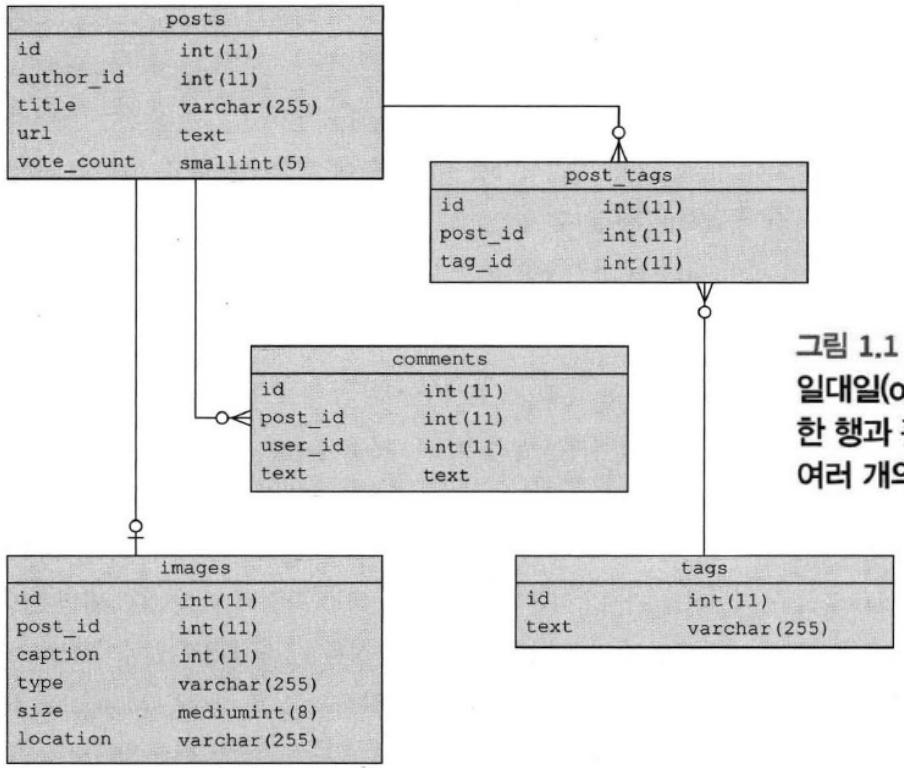


그림 1.1 소셜 뉴스 사이트에서 각 데이터에 대한 기본적인 관계 데이터 모델. 십자 표시처럼 보이는 줄 끝은 일대일(one-to-one) 대응 관계를 표현하며, 따라서 images 테이블에서 오직 하나의 행만이 posts 테이블의 한 행과 관계를 맺는다. 여러 개의 가지로 갈라진 줄 끝은 일대다 대응 관계를 표현하며, comments 테이블의 여러 개의 행은 posts 테이블의 하나의 행과 관계를 맺게 된다

- 내부적으로 Binary JSON 또는 BSON의 형태로 도큐먼트 저장

스키마가 없는 모델의 장점

- 데이터베이스가 아닌 애플리케이션이 데이터 구조를 정한다.
  - RDBMS에서 어떤 데이터 열에 필드 하나를 추가해야 할 경우에는, 테이블의 구조 전체 (허용되는 데이터 타입)를 바꿔줘야 한다.
  - 반면 MongoDB에서는 (스키마 없이) 그저 추가하면 된다.
  - 개발 초기단계에는 데이터의 구조를 자주 변경할 테니, 개발 속도가 빨라질 것이다.

- 가변적인 속성을 갖는 데이터를 표현할 수 있다.
  - 추후에 필요할 데이터 필드가 무엇일지 걱정할 필요가 없다.
  - RDBMS와 달리 일대다 관계가 가능하다.
  - 각각의 도큐먼트들이 서로 완전히 다른 구조를 갖는 것도 가능하다.

1.2.2 애드혹 쿼리

- 애드혹 쿼리(ad hoc query)를 지원한다?
  - 질의를 미리 정의하지 않아도 된다!
  - 즉 질의가 어떠한 조건을 갖더라도 올바른 구조이기만 하면 쿼리가 실행된다.

- 사실 이러한 동적 질의는 RDBMS의 특징적인 쿼리 기능
  - RDBMS에서 매우 필수적인 쿼리 언어 성능을 MongoDB에서도 유지하고자 지원
  - 서로 구조는 다르지만, 여러 속성을 조합할 수 있다는 게 장점

1.2.3 인덱스

- 인덱스(Index)
  - 효과적으로 데이터를 검색할 수 있는 방법
  - 예를 들어 대부분의 책에는 키워드와 페이지 번호를 연결해 놓은 인덱스가 있다.
  - MongoDB에서 인덱스는 B-트리로 구현
  - 3.2 버전 이후 WiredTiger 엔진이 도입되며 LSM(Log-Structured Merge-trees) 지원

- 세컨더리 인덱스(secondary index)
  - 각 도큐먼트(행)는 고유의 식별자로서 프라이머리 키(primary key) 부여됨
  - 여러 개의 세컨더리 인덱스를 허용하여 쿼리 최적화 가능
  - 많은 NoSQL DB에서 세컨더리 인덱스를 허용하지 않는다는 점에서 중요한 특징
  - 최대 64개까지 세컨더리 인덱스 생성 가능

1.2.4 복제

- 복제(replication) 기능
  - 데이터베이스 안 데이터를 여러 대의 서버에 분산하는 것
  - 서버와 네트워크 장애가 발생할 경우를 대비해 중복성과 자동 장애조치

- 복제 세트(replica set)
  - 복제 세트는 많은 MongoDB 서버로 구성되며, 각각 노드(Node)라고 불림
  - 노드는 보통 분리된 물리 장비에 별도로 존재함

- 프라이머리 노드(primary node), 세컨더리 노드(secondary node)
  - 복제 세트는 하나의 프라이머리 노드와 하나 이상의 세컨더리 노드로 구성
  - 프라이머리 노드는 읽기/쓰기, 세컨더리 노드는 읽기 지원
  - 프라이머리 노드에 장애 발생 시, 자동으로 세컨더리 노드 중 하나를 프라이머리로 설정하고, 이후 이전 프라이머리 노드가 복구되면 세컨더리 노드로 작동

1.2.4 복제

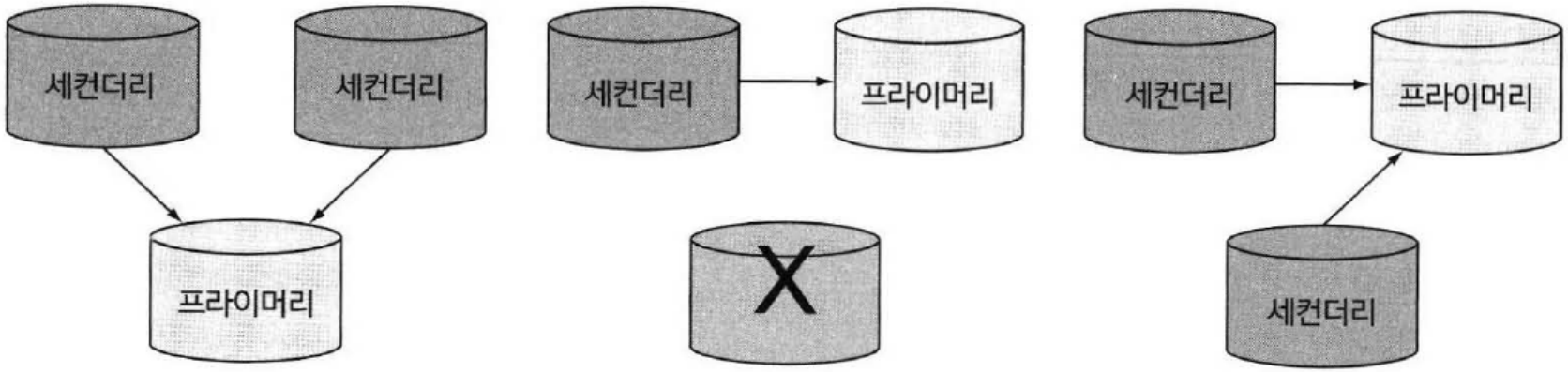


그림 1.3 복제 세트의 자동 장애복구

- 복제 세트를 통해 데이터베이스 복제기능 제공

1.2.5 속도와 내구성

- 쓰기 속도와 내구성은 trade-off

① 쓰기 속도(write speed)
  - 정해진 시간 내에 얼마나 많은 수의 삽입, 수정, 삭제 명령을 처리할 수 있는가

② 내구성(durability)
  - 이 쓰기 연산이 디스크에 제대로 이루어졌다는 것을 확신할 수 있는 정도

- MongoDB의 타협안

① 쓰기 시멘틱스(write semantics): 명령하고 잊어버리기(fire-and-forget) 모드
  - 설정 시 사용자에게 응답을 주기 전에 쓰기를 램에 안전하게 쓰는 것을 보장해준다.
  - 중요한 데이터의 경우에는 안전 모드 쓰기가 낫다.

② 저널링(journaling): 모든 쓰기에 대한 로그를 100ms마다 한 번씩 저널 파일에 기록

1.2.6 확장

① 수직적 확장(vertical scaling)
- = 상향식 확장(scaling up) = 단 하나의 노드를 업그레이드

② **수평적 확장(scaling horizontally)**
- = 외적 확장(scaling out) = 데이터베이스를 여러 대의 서버에 분산
- 상대적으로 비용 절감, 장애 피해 경감

- 샤딩(sharding) 범위 기반 파티션 메커니즘
  - 데이터를 여러 노드에 걸쳐 분산하는 것을 가능하게 해준다.
  - 샤드 노드를 추가하여 용량을 증설할 수 있으며 자동 장애조치도 된다.
  - 하나의 노드에 연결하듯 샤드 클러스터에 연결만 하면 된다.

1.2.6 확장

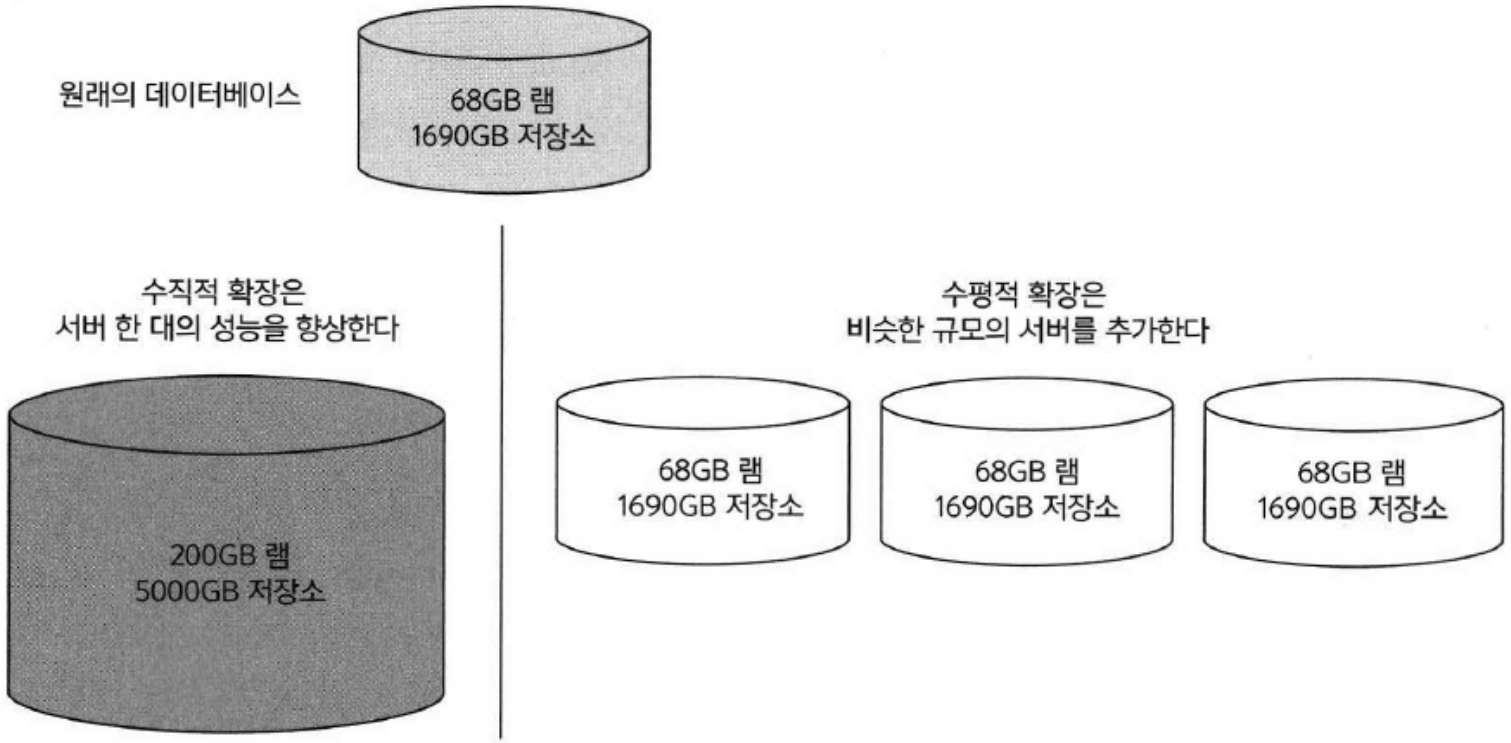


그림 1.4 수직적 확장 vs 수평적 확장

- MongoDB는 수평적 확장이 용이하도록 설계되어 있다.

1.3 MongoDB 코어 서버와 툴

1.3.1 코어 서버

- 코어 서버는 mongod(윈도우에서는 mongod.exe)로 구동
- Mongod 프로세스의 데이터 파일은 모두 같은 서버에 저장됨
  - /data/db (유닉스 등) 또는 C:\data\db (윈도우)

1.3.2 자바스크립트 셸

- MongoDB 명령어 셸은 자바스크립트에 기반한 툴
- 데이터베이스를 관리하고 데이터를 조작하는 데 사용

1.3.3 데이터베이스 드라이버

- 애플리케이션 상에서 DB 서버와 통신하기 위해 사용
- JavaScript, Python, Ruby 포함 대부분의 드라이버를 공식적으로 지원

MongoDB 툴

1.3.4 커맨드라인 툴

- mongodump, mongorestore 백업과 복구를 위한 유틸리티
- mongoexport, mongoimport JSON, CSV, TSV 타입의 데이터를 엑스포트(export)와 임포트(import)
- mongosniff 데이터베이스에 요청된 오퍼레이션을 보기 위한 와이어 스나이핑(wire-sniffing) 툴 (BSON을 셸 문장으로 변환)
- mongostat (iostat과 비슷하게) mongodb와 시스템을 계속 풀링(pool)해서 여러 유용한 통계 데이터 제공
- 이 밖에도 bsondump, monfiles 등 커맨드 라인 유틸리티를 제공한다.

1.4 MongoDB를 사용하는 이유

1) 설계자들의 주장

- RDBMS와 키-값 저장 시스템의 장점만 모았다.
  - 키-값 저장 시스템: 단순하므로 속도가 매우 빠르고 확장도 상대적으로 용이
  - RDBMS: 수평적인 확장은 어려운 반면, 다양한 데이터 모델과 강력한 쿼리 언어

2) 데이터베이스의 용도 관점

- 애플리케이션에서 일차 데이터 저장 시스템으로 적합
  - 웹, 분석과 로깅 애플리케이션, 중간 정도의 캐시(cache)를 필요로 하는 경우
- 스키마가 존재하지 않는 데이터를 저장하기가 용이하므로 구조가 미리 정해지지 않은 데이터 저장에 유용

다른 데이터베이스와의 비교

1) 간단한 키-값 저장 시스템
- *멤캐시디(Memcached)*
- 빠르고 확장성이 좋지만 오퍼레이션이 제한적

2) 정교한 키-값 저장 시스템
- *카산드라, 볼드모트 프로젝트, 리악 등*
- 초 대용량의 데이터를 갖는 여러 시스템을 관리하기 위해 개발
- 세컨더리 인덱스 없이는 질의하는 데 DB가 무용지물

3) 관계 데이터베이스

2장

자바스크립트 셸을 통한 MongoDB

2.1 MongoDB 셸 경험하기

2.1.1 셸 시작하기

- 부록 A를 참고하여 MongoDB를 설치
- Download Center
- 결과적으로 다음 셸이 실행되어야 한다.

```
(base) C:₩Users₩rinseo>mongo --version
MongoDB shell version v4.2.0
git version: a4b751dcf51dd249c5865812b390cfd1c0129c30
allocator: tcmalloc
modules: none
build environment:
    distmod: 2012plus
    distarch: x86_64
    target_arch: x86_64
```

[Windows]

1. Download .msi
   - Choose your version / OS
2. 시스템 환경변수 설정
   - PATH에 'C:\Program Files\MongoDB\Server\4.2\bin' 추가
   - Type "mongod" in cmd (에러 발생!)
3. 데이터 디렉토리 생성
   - C:\data\db\를 직접 생성해주거나 --dbpath--옵션으로 변경
   - 재실행 후 localhost:27017에서 확인

2.1.2 데이터베이스, 컬렉션, 도큐먼트

- JSON 형태로 표현하는 도큐먼트에 저장

- 컬렉션(collection
  - 다른 유형의 도큐먼트를 별도 저장하고 싶을 때
  - RDBMS의 테이블과 같이 도큐먼트들을 그루핑(grouping)하는 방법
  - 별개의 데이터베이스(database)에 분리됨

- 질의하기 위해서는, 질의를 하기 위해 대상 도큐먼트가 존재하는 데이터베이스(또는 네임스페이스)와 컬렉션을 알아야 함
  - 데이터베이스 미지정시 test라는 이름의 기본 설정 데이터베이스에 연결

2.1.3 삽입과 질의

> db.users.insert({username: “smith”})

- 첫 번째 도큐먼트가 저장됨 (셸을 중단하거나 재가동해도 삽입이 보장)
- tutorial 데이터베이스와 users 컬렉션이 하드디스크에 생성됨

> db.users.find()

{ "_id" : ObjectId("5c8b07d34cc96c5cbff22610"), "username" : "smith" }

- “_id”필드는 도큐먼트의 프라이머리 키
- 도큐먼트 생성시 (이 필드가 없으면) 자동으로 추가됨

> db.users.insert({username: “jones”}) // 컬렉션에 두 번째 사용자 추가

> db.users.count() // 컬렉션 내 도큐먼트의 수 count

질의 술어 넘겨주기

> db.users.find()

- 컬렉션의 모든 도큐먼트를 볼 수 있음

> db.userss.find({username: “jones”})

- 존재하는 모든 도큐먼트에 대해 username 키 값이 jones와 일치하는지 검사
- 매개변수 없이 find 호출 = 비어 있는 술어를 넘김 (find() 는 find({})와 같음)

- 필드 사이에 암시적인 AND를 만들어 내기 위해 여러 개 필드 명시 가능
  - 또는 명시적으로 MongoDB의 $and 연산자 사용 가능
  - OR의 경우 $or 연산자만 이용하여 구현 가능

2.1.4 도큐먼트 업데이트

1) 연산자 업데이트

> db.users.update({username: “smith”}, {$set: {country: “Canada”}})

- 사용자 이름이 smith인 도큐먼트를 찾아서 country 속성의 값을 Canada로 수정

> db.users.update({username: “smith”}, {country: “Canada”})

- 위와 같은 쿼리를 실행하면, 도큐먼트의 username 필드는 제거됨
- 첫 번째 도큐먼트는 매칭을 위해서만 사용되고 두 번째 도큐먼트가 대체되는데 사용
- 반드시 $set 연산자를 사용하여 입력하기

> db.users.update({username: “smith”},{$unset: {country: 1}})

- $unset을 통해 원하지 않는 값 제거

$and 연산자

```
> db.users.find({
_id: ObjectId("5d841d85c8551f19c4ed5db3"),
username: "smith"})

> db.users.find({$and: [
{_id: ObjectId("5d841d85c8551f19c4ed5db3")},
{username: "smith"}
]
})
```

복잡한 데이터 업데이트

```
> db.users.update({username: “smith”},
{ $set: {
Favorites: {
cities: [“Chicago”, “Cheyenne”],
movies: [“Casablanca”, “For a Few Dollars More”, “The Sting”]
}
}
})
```

더 발전된 업데이트

- 하나의 값만 배열에 추가하고 싶을 때 $push 나 $addToSet을 사용
  - ($push와 달리) $addToSet는 값을 추가할 때 중복을 확인할 수 있다.

```
// $addToSet
db.users.update( {"favorites.movies": "Casablanca"},
    {$addToSet: {"favorites.movies": "The Maltese Falcon"} },
            false,    // insert if not found?
            true )    // update all found? (if false, updates just first it finds)
```

① 쿼리 셀렉터
② $addToSet 연산자를 이용해서 영화 추가
③ upset(update & insert) 허용 여부 = 해당되는 도큐먼트가 없을 때 update 연산이 도큐먼트를 입력해야 하는가?
④ 다중 업데이트 허용 여부 = 기본적으로 쿼리 셀렉터의 조건에 맞는 “첫 번째” 도큐먼트에 대해서만 업데이트 하도록 되어 있음

가독성 있게 확인하기

> db.users.find().pretty()

- 엄밀히 말하면 find() 명령은 반환하는 도큐먼트에 커서(cursor)를 반환
- pretty()는 실제로 cursor.pretty()가 됨

> db.users.find({favorites.movies": "Casablanca"})

- favorites와 movies 사이의 점(dot)은 쿼리 엔진으로 하여금 favorites 이름의 키를 찾도록 한다.
- 그리고 이 키의 값인 객체의 내부에서 다시 movies라는 이름의 키의 값이 되는 객체를 찾도록 함.

2.1.5 데이터 삭제

> db.foo.remove({})

- foo 컬렉션의 모든 도큐먼트를 삭제

> db.users.remove({“Favorites.cities”: “Cheyenne”})

- 쿼리 셀렉터를 통해 특정 정보만 삭제

> db.users.drop()

- 한 컬렉션을 모든 인덱스와 함께 삭제

2.2 인덱스 생성과 질의

2.2.1 대용량 컬렉션 생성

```
> for(i = 0; i < 20000; i++) {
db.numbers.save({num: i});
}
```

- numbers 컬렉션에 20,000개의 도큐먼트 추가
- save 메소드는 inser와 updat의 wrapper

- 범위 쿼리

```
> db.numbers.find( {num: {“$gt”: 20, “$lt”: 25}})
```

- $gt : greater than, $lt : less than
- $gte (greater than or equal to), $lte (less than or equal to), $ne (not equal to)

2.2.2 인덱싱과 explain()

- 쿼리 플랜 쿼리를 받았을 때 이를 실행하는 방법에 대한 계획

> db.numbers.find({num: {"$gt":19995}}).explain("executionStats")

  - 4개 결과 반환을 위해 20,000개 전부 스캔한 쿼리

- 인덱스의 필요성

> db.numbers.createIndex({num: 1})

  - 구 버전에서는 ensureIndex() 사용
  - {num:1}은 모든 도큐먼트의 num 키에 대해 오름차순 인덱스 생성 의미
  - 인덱스 성공 생성 여부는 .getIndexes()를 통해 확인.

> db.numbers.getIndexes()

  - 두개의 인덱스(기존 “_id”와 새로 생성한 num에 대한 인덱스, “num_1”)를 가짐

2.3 기본적인 관리

> show dbs
- 시스템상의 모든 데이터베이스 보여줌

> show collections
- 현재 사용 중인 데이터베이스에 정의된 모든 컬렉션 보여줌

> db.stats()
- 데이터베이스와 컬렉션에 대한 좀 더 하위 계층의 정보를 얻게 됨
- 각 컬렉션에 대해 stat() 실행 가능; db.numbers.stats()

2.3.2 명령어가 작동하는 방식

> db.runCommand({dbstats:1})

- db.stats()를 실행시키는 것과 동일
- runCommand에 매개변수로서 명령을 도큐먼트로 정의하여 넘기면 모든 명령 실행 가능

> db.runCommand({collstats: “numbers”})

- 컬렉션에 대해 stats 명령어 실행

2.4 도움말 얻기

> db.help()

- 데이터베이스에 대해 일반적으로 수행되는 메서드 리스트를 보여줌

> db.numbers.help()

- 컬렉션에 대해 수행되는 메서드를 보여줌

- 자동 탭 완성 기능
  - 첫 번째 문자 입력하고 탭 키 두 번
  - 메서드의 이름을 괄호 없이 입력하면 MongoDB 소스코드 확인 가능

The end.