MongoDB

MongoDB

Review (1/4)

- Database 생성
  - use database_name 으로 생성
  - 1개 이상의 collection이 존재해야 database 리스트에서 확인
- Database 조회
  - show dbs : database 리스트 확인
  - db : 현재 사용 중인 database 확인

Review (2/4)

- Collection 생성
  - db.createCollection(name, [option]) 으로 collection 생성
  - name은 collection 이름이고, option은 document 타입으로 구성된 해당 collection의 설정값
  - option 객체의 속성들
    - capped : Boolean타입, 이 값을 true로 설정하면 capped collection을 활성화, Capped collection 이란 고정된 크기를 가진 collection으로, size가 초과되면 가장 오래된 데이터를 덮어씀, 이 값을 true로 설정하면 size 값을 꼭 설정
    - size : number 타입, capped collection을 위해 해당 collection 최대 사이즈를 ~bytes로 지정
    - max : number 타입, 해당 collection에 추가할 수 있는 최대 document 개수를 설정

Review (3/4)

- Collection 조회
  - show collections로 collection 리스트 확인
- Collection 제거
  - db.collection명.drop()으로 collection 제거
- Document 생성
  - db.collection명.insert(document)로 document 추가
  - 배열형식으로 전달하면 여러 document를 bulk 형식으로 추가
- Document 조회
  - db.collection명.find([query, projection]) 로 collection의 document 리스트 확인
  - 한 줄이 너무 길어 불편할 때는 끝에 .pretty()를 붙이면 사용자가 보기 편하게 결과 출력

Review (4/4)

- Document 제거
  - db.collection명.remove(criteria[, justOne])로 document 제거
  - 매개변수로 들어가는 객체의 속성들은 아래와 같음
    - criteria : document 타입, 데이터의 기준 값으로 일치하면 기본적으로 다 삭제, 이 값이 { } 이면 collection의 모든 데이터를 제거, 반드시 넣어야 함
    - justOne : Boolean 타입, optional 매개변수이며, 이 값이 true이면 1개의 document만 제거, 이 매개변수가 생략되면 기본값을 false이고, criteria에 해당되는 모든 document 제거

  - MongoDB 종료
    - 몽고쉘에서 데몬 종료

```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
```

SQL vs. MongoDB 용어 정리

SQL 용어	MongoDB 용어
database	Database
table	Collection (도큐먼트들의 집합)
row	도큐먼트 or BSON document (필드: 값의 쌍으로 이루어짐)
column	field
index	index
table joins	embedded document (내장 도큐먼트)
primary key	primary key
하나의 열 또는 여러 열의 조합으로 기본 키를 생성	기본키를 지정할 필요 없이 _id 필드를 자동으로 기본키로 설정

SQL vs. MongoDB

SQL SELECT Statement	MongoDB find() Statement
SELECT * FROM samples	db.samples.find()
SELECT id, user_id, status FROM samples	db.samples.find({ }, { user_id: 1, status: 1 })
SELECT user_id, status FROM samples	db.samples.find({ }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM samples WHERE status = "B"	db.samples.find({ status: "A" })
SELECT user_id, status FROM samples WHERE status = "A"	db.samples.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM samples WHERE status != "A"	db.samples.find({ status: { $ne: "A" } })
SELECT * FROM samples WHERE status = "A" AND age = 50	db.samples.find({ status: "A", age: 50 })
SELECT * FROM samples WHERE status = "A" OR age = 50	db.samples.find({ $or: [{ status: "A" } , { age: 50 }] })
SELECT * FROM samples WHERE age > 25	db.samples.find({ age: { $gt: 25 } })
SELECT * FROM samples WHERE age < 25	db.samples.find({ age: { $lt: 25 } })

SQL vs. MongoDB

SQL SELECT Statement	MongoDB find() Statement
SELECT * FROM samples WHERE age > 25 AND age <= 50	db.samples.find({ age: { $gt: 25, $lte: 50 } })
SELECT * FROM samples WHERE user_id like "%bc%"	db.samples.find({ user_id: /bc/ }) 또는 db.samples.find({ user_id: { $regex: /bc/ } })
SELECT * FROM samples WHERE user_id like "bc%"	db.samples.find({ user_id: /^bc/ }) 또는 db.samples.find({ user_id: { $regex: /^bc/ } })
SELECT * FROM samples WHERE status = "A" ORDER BY user_id ASC	db.samples.find({ status: "A" }).sort({ user_id: 1 })
SELECT * FROM samples WHERE status = "A" ORDER BY user_id DESC	db. samples.find({ status: "A" }).sort({ user_id: -1 })
SELECT COUNT(*) FROM samples	db.samples.count() or db. samples. find().count()
SELECT COUNT(user_id) FROM samples	db.samples.count({ user_id: { $exists: true } }) 또는 db.samples.find({ user_id: { $exists: true } }).count()
SELECT COUNT(*) FROM samples WHERE age > 30	db.samples.count({ age: { $gt: 30 } }) 또는 db.samples.find({ age: { $gt: 30 } }).count()
SELECT * FROM samples LIMIT 1	db. samples.findOne() 또는 db. samples.find().limit(1)
EXPLAIN SELECT * FROM samples WHERE status = "A"	db. samples. find({ status: "A" }).explain()

SQL vs. MongoDB

SQL Update Statements	MongoDB updateMany() Statements
UPDATE samples SET status = "C" WHERE age > 25	db.samples.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" } })
UPDATE samples SET age = age + 3 WHERE status = "A"	db.samples.updateMany({ status: "A" } , { $inc: { age: 3 } })

SQL Delete Statements	MongoDB deleteMany() Statements
DELETE FROM samples WHERE status = "D"	db.samples.deleteMany({ status: "D" })
DELETE FROM samples	db.samples.deleteMany({ })

커서 (Cursor) (1/2)

- 커서란?
  - 테이블에서 여러 개의 행을 쿼리한 후에, 쿼리의 결과인 행 집합을 한 행씩 처리하기 위한 방식
  - 커서의 초기값은 첫 번째 행 이전을 가리킴
  - next()를 이용하여 다음 행으로 이동하여 해당 행의 값을 반환

※ 본 페이지의 슬라이드 내용은 이해를 도모하기 위해 관계형 데이터베이스 기준으로 작성되었음

커서의 초기 위치

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	죽구의 역사	굿스포츠	7000
2	죽구 아는 여자	나무수	13000
3	죽구의 이해	대한미디어	22000

커서 (Cursor) (2/2)

도큐먼트 준비

```
for(i=0; i<100;i++){
   db.foo.insertOne({x : i});
}
```

커서 생성

```
var cursor = db.foo.find();
```

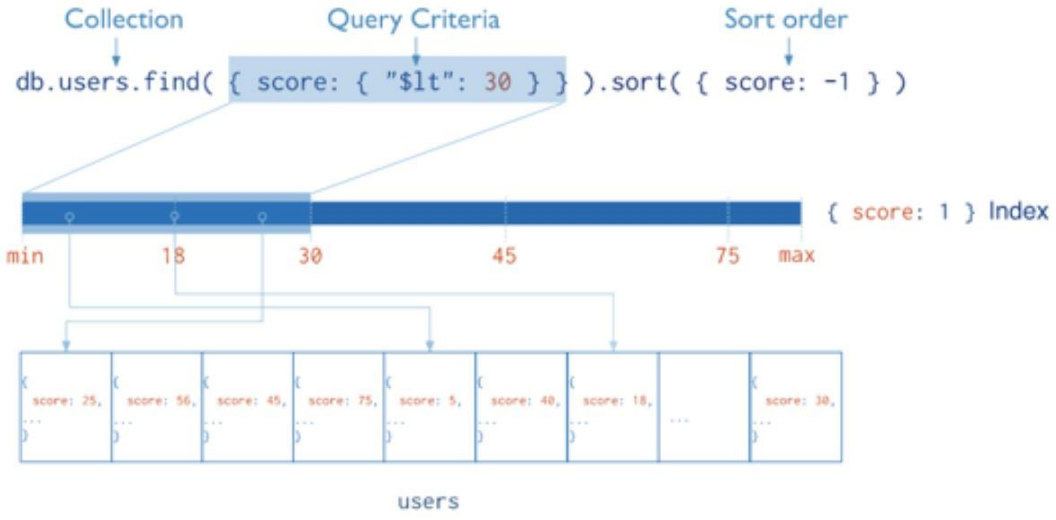
데이터 fetch

```
while(cursor.hasNext()){
      obj = cursor.next();
      print(obj.x);
}
```

0부터 99까지 콘솔에 출력

인덱싱 (1/4)

- Query를 더욱 효율적으로 할 수 있도록 documents에 기준(key)을 정해 정렬된 목록을 생성하는 것
- 인덱스가 없으면 쿼리 조건에 맞는 도큐먼트를 조회하기 위해 collection scan(컬렉션의 도큐먼트를 처음부터 끝까지 조회)을 실행



인덱싱 (2/4)

- 몽고디비 인덱스
  - DB의 검색을 빠르게 하기 위하여 데이터의 순서를 미리 정해두는 과정
  - 고정된 스키마는 없지만, 원하는 필드를 인덱스로 지정하여 검색 결과를 빠르게 함
  - index는 한 쿼리에 한 개의 index만 유효함 → 두 개의 index 가 필요하다면 복합 index를 사용
  - index는 어떤 데이터가 도큐먼트에 추가되거나 수정될 때(write 작업) 그 컬렉션에 생성되어 있는 index도 새로운 도큐먼트를 포함시켜 수정됨 → index 추가 시 wirte 작업은 느려질 수 있음 → index는 read 작업 위주의 애플리케션에서 유용하고 읽기보다 쓰기 작업이 많으면 index를 생성하는 것은 비효율적임

인덱싱 (3/4)

- _id 인덱스
  - 컬렉션을 생성하는 동안_id 필드에 고유한 인덱스를 생성
  - _id 필드에 같은 값을 가진 2 개의 도큐먼트를 삽입할 수 없도록 함
  - _id 필드의 인덱스를 제거할 수 없음
- 인덱스 생성하기

```
db.collection.createIndex( <key and index type specification>, <options> )
```

- 인덱스의 기본 이름
  - 인덱스의 키와 방향의 조합
  - 예 : {item : 1, quantity : -1}에 생성된 이름 item_1_quantity_-1

인덱싱 (4/4)

- 샘플 데이터 생성하기

```
for (i =0; i<500000; i++){
        db.user.insert({
                "userid" : i,
                "name": "user"+i,
                "age": Math.floor(Math.random()*100),
                "score": Math.floor(Math.random()*100),
                "time": new Date()
        });
}
```

```
db.user.find({score:"23"}).explain("executionStats").executionStats.executionTimeMillis
28
```
인덱스 생성 전 검색 속도 측정

인덱스 종류 (1/2)

- 단일 필드 인덱스 (Single Field Index)
  - 하나의 필드 인덱스 사용

오름차순

```
db.user.createIndex( { score: 1 } )
```

{ score: 1 } Index

- 인덱스 유무 속도 비교

```
db.user.find({score:"23"}).explain("executionStats").executionStats.executionTimeMillis
```

28 생성 전

```
db.user.find({score:"23"}).explain("executionStats").executionStats.executionTimeMillis
```

3 생성 후

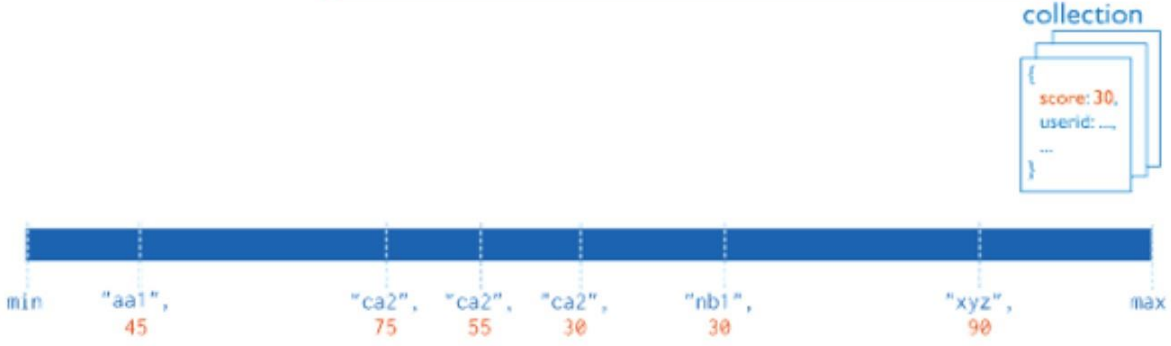
인덱스 종류 (2/2)

- 복합 인덱스 (Compound Index)
  - 두 개 이상의 필드를 사용하는 인덱스

```
db.collection.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )
```

  - 예와 같이 인덱스를 생성하면 userid는 오름차순으로, 같은 userid를 지니면 score로 내림차순 정렬함. 동일한 userid 인"ca2"는 score가 내림차순으로 정렬되어 있음

```
db.user.createIndex( {userid: 1, score: -1 } )
```



{ userid: 1, score: -1 } Index

인덱스 조회 및 삭제

- 인덱스 조회

```
db.user.getIndexes()
```

- 인덱스 삭제

```
db.users.dropIndex("userid_1_score_-1")
```

```
> db.users.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "userid" : 1,
                        "score" : -1
                },
                "name" : "userid_1_score_-1"
        }
]
```

텍스트 인덱스 (1/2)

- 제목, 설명 등 컬렉션 내에 있는 필드의 텍스트와 일치시키는 키워드 검색에 사용
  - 샘플 데이터

```
db.library.insert(
   [
     { _id: 101, name: "Java", description: "By ABC" },
     { _id: 102, name: "MongoDB", description: "By XYZ" },
     { _id: 103, name: "Python", description: "By ABCD" },
     { _id: 104, name: "Engineering Mathematics", description: "By *****" },
     { _id: 105, name: "Salesforce", description: "By Salesforce" }
   ]
 )
```

  - 텍스트 인덱스 생성

```
db.library.createIndex( { name: "text", description: "text" } )
```

텍스트 인덱스 (2/2)

- "Java"가 있는 도큐먼트 반환

```
db.library.find( { $text: { $search: "Java" } } )
```

- "Java Book"이 있는 도큐먼트 반환

```
db.library.find( { $text: { $search: "\"Java Book\"" } } )
```

- Sorting
  - 몽고디비가 반환하는 결과는 기본적으로 정렬되어 있지 않으므로, 정렬된 결과를 원하는 경우 sort() 사용