기본 데이터 처리

ObjectId

다큐먼트를 유일하게 식별하는 id 값으로 다음과 같은 특징을 가진다.

- 쉬운 생성
- 작은 데이터
- 유일성
- 순차성

유일하고 순차적인 것을 보장하면서 사용사가 _id 를 통해 값을 정의 할 수 있는 부분에서 auto_increment 와 유사하다.

ObjectId 구성

- 4 byte
  - Unix epoch 기반의 초단위의 timestamp 값으로 objectid가 생성된 시간이다.
- 5 byte
  - 랜덤값
- 3 byte
  - 랜덤값으로 초기화되고 이후 증가하는 값

ObjectId 포맷

BSON 은 기본적으로 리틀엔디안이지만 ObjectId 의 timestamp 와 counter 는 빅엔디안이다. 값 자체에 바이트 시퀀스는 없지만 내부적인 관리측면에서 1바이트를 사용하는듯 하다.

ObjectId 사용

ObjectId 는 collection 의 각 다큐먼트가 저장될때 마다 _id 를 통해 사용되고 primary key 역할을 한다. 사용자가 값을 지정하지 않을 경우 내부에서 자동으로 ObjectId 를 생성해서 사용한다. 또한 update 에서도 "upsert : true" 를 사용할 경우에도 사용될 수 있다.

ObjectId 를 사용자가 정의하는 예시

```
> var newid = new ObjectId();
> print(newid)
ObjectId("6034a423b50e6cb6b15c288c")
// newid 에 임의의 숫자를 넣어도 가능
> db.users.insert( {_id: newid, name : "chlee" } )
WriteResult({ "nInserted" : 1 })
> db.users.find( { name : "chlee" } )
{ "_id" : ObjectId("6034a423b50e6cb6b15c288c"), "name" : "chlee" }
>
```

ObjectId 메소드

```
> newid.getTimestamp()
ISODate("2021-02-23T06:43:47Z")
> newid.toString()
ObjectId("6034a423b50e6cb6b15c288c")
> newid.valueOf()
6034a423b50e6cb6b15c288c
> newid.str
6034a423b50e6cb6b15c288c
>
```

CRUD 쓰기 쿼리의 기본 동작

Insert

insert 시점에서 collection 이 존재하지 않는다면 collection 을 생성한다.

_id 필드

ObjectId 는 collection 의 각 다큐먼트가 저장될때 마다 _id 를 통해 사용되고 primary key 역할을 한다. 사용자가 값을 지정하지 않을 경우 내부에서 자동으로 ObjectId 를 생성해서 사용한다.

원자성

mongodb 에서는 모든 쓰기연산에서 단일 다큐먼트 레벨의 원자성을 보장한다. (임베디드 다큐먼트도 포함)

쓰기 응답

WriteConern 설정을 통해 쓰기 동작에 대한 응답레벨을 정할 수 있다.

Update

_id 필드

한번 설정된 _id 필드 값은 다른 값으로 변경할 수 없다.

원자성

mongodb 에서는 모든 쓰기연산에서 단일 다큐먼트 레벨의 원자성을 보장한다. (임베디드 다큐먼트도 포함)

필드 순서

쓰기 연산을 이후 정해진 필드 순서는 다음 경우를 제외하고 변한지 않는다.

- _id 필드는 언제나 첫번째 필드 이다.
- 필드의 이름을 변경하면 순서가 바뀌 수 있다.

Upsert Option

updateOne(), updateMany(), replaceOne() 등에서 "upsert: true" 옵션을 명시한다면 filter 맞는 다큐먼트를 찾지 못했을때 insert 연산을 하게 된다.

쓰기 응답

WriteConern 설정을 통해 쓰기 동작에 대한 응답레벨을 정할 수 있다.

Delete

모든 다큐먼트가 삭제되더라도 인덱스를 삭제하지 않는다.

원자성

mongodb 에서는 모든 쓰기연산에서 단일 다큐먼트 레벨의 원자성을 보장한다. (임베디드 다큐먼트도 포함)

쓰기 응답

WriteConern 설정을 통해 쓰기 동작에 대한 응답레벨을 정할 수 있다.

WriteConcern

WriteConcern 은 쓰기 연산에 대한 응답레벨을 기술하는 것이다. shared cluster 에서는 mongos 에 의해 각 instance 에 전송된다.

> transaction 안에서 연산단위의 WriteConcern 은 설정하지 않는다.

mongodb 4.4 부터는 global WriteConcern 설정을 통해 replica set 이나 shared cluster 에 WriteConcern 을 적용할 수 있다.

setDefaultRWConcern 은 관리명령어로 global derault read or write concern 을 설정할 수 있으며 admin 데이터베이스에서만 실행 가능하다.

- replica set 을 위해서는 prinary mongod 에서 실행
- shared cluster 를 위해서는 mongos 에서 실행

```
db.adminCommand(
  {
    setDefaultRWConcern : 1,
    defaultReadConcern: { <read concern> },
    defaultWriteConcern: { <write concern> },
    writeConcern: { <write concern> },
    comment: <any>
  }
)
```

사용되는 필드

```
{ w: <value>, j: <boolean>, wtimeout: <number> }
```

w option

쓰기 연산이 지정된 수 만큼의 instance 에 전달된 것을 보장 받는다는 의미이다. default 는 1 이며

0 이 설정 된 경우 응답을 기다리지 않지만 socket 오류 등에는 error 를 반환하게 되며 secondary 에 전파가 되기 전에 primary step down (장애) 이 발생하는 경우 roll back 된다. 설정이 0일 지라도 "j:true" 가 우선시 된다.

> delayed sedondary 의 경우 지정된 지연시간 이전에 쓰기 응답을 할 수 있다.

majority 가 설정된 경우 투표멤버의 절반 이상에 전달된 것을 보장 받는다는 의미이다. w:majority 설정하고 그 응답을 받은 client 만 쓰기에 대한 majority readconcern 을 사용할 수 있다.

j option

쓰기 연산이 journal 에 기록(디스크기록) 되었는지를 결정하는 옵션이다. true 일때 w: 만큼의 instance 에 기록되었는지 확인한다. journaling 을 활성화 하지 않았다면 오류가 발생된다.

wtimeout option

무한으로 대기하는것을 막기위한 timeout 옵션이다.

쓰기 명령어 사용법

insert

update

Updates with Aggregation Pipeline

mongodb 4.2 부터는 update 연산에 "Aggregation Pipeline" 을 사용할 수 있다. 사용 가능한 스테이지는 다음과 같다.

stage	desc
$addFields	기존에 존재하는 필드를 포함하고 새로운 필드가 포함된 다큐먼트를 출력한다.\n 기본필드를 명시할 경우 데이터를 갱신하게 된다.
$project	선택된 필드를 다음 스테이지에 전달하며 이때 필드는 새로운 값이나 기존의 필드를 계산한 값이 될 수 있다.
$set	새로운 필드를 추가하며 해당 기능은 $addFields 와 alias 이다.
$unset	지정된 필드를 remove/exclude 할 수 있고 $project와 alias 이다.
$replaceRoot	입력된 다큐먼트를 지정한 다큐먼트로 교체한다. \n replace 연산은 기존 모든 필드를 교체할 수 있으며 임베디드 다큐먼트를 최상위 레벨로 올리거나 새로운 다큐먼트를 승급시킬 수도 있다.
$replaceWith	(new in 4.2) $replaceRoot 와 alias 이며 기능은 같으나 shortcut 으로 일부 키워드가 생략된 것으로 추정된다.

Updates with Aggregation Pipeline 예제

```
// 예제 다큐먼트 입력
> db.students2.insertMany([
...    { "_id" : 1, quiz1: 8, test2: 100, quiz2: 9, modified: new
Date("01/05/2020") },
...    { "_id" : 2, quiz2: 5, test1: 80, test2: 89, modified: new
Date("01/05/2020") },
... ])
{ "acknowledged" : true, "insertedIds" : [ 1, 2 ] }
>
// update 수행
> db.students2.updateMany(
... {},   // filter, 전체 다큐먼트
...   [   // aggregation pipiline
...     // 첫번째 pipeline
...     { $replaceRoot: { newRoot:
...        { $mergeObjects: [ { quiz1: 0, quiz2: 0, test1: 0, test2: 0 }, "$$ROOT"
] }
...     } },
...     // 두번째 pipeline
...     { $set: { modified: "$$NOW"}  }
...   ]
... )
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
> db.students2.find()
{ "_id" : 1, "quiz1" : 8, "quiz2" : 9, "test1" : 0, "test2" : 100, "modified" :
ISODate("2021-02-24T04:51:28.309Z") }
{ "_id" : 2, "quiz1" : 0, "quiz2" : 5, "test1" : 80, "test2" : 89, "modified" :
ISODate("2021-02-24T04:51:28.309Z") }
>
```

upsert

update 연산에서 filter 에 매칭되는 다큐먼트를 찾지 못하는 경우 새로운 다큐먼트로 입력하는 옵션이다.

Capped Collection

update 연상이 다큐먼트의 사이즈를 변경하는 경우 실패하게 된다.

Sharded Collections

updateOne() 또는 updateMany() 에서 "upsert : true" 를 사용할 경우 shard key 를 filter 조건으로 사용해야 한다.

upsert 예제

```
// 예제 다큐먼트 입력
> db.students2.insertMany([
...    { "_id" : 1, quiz1: 8, test2: 100, quiz2: 9, modified: new
```

```
Date("01/05/2020") },
...    { "_id" : 2, quiz2: 5, test1: 80, test2: 89, modified: new
Date("01/05/2020") },
... ])
{ "acknowledged" : true, "insertedIds" : [ 1, 2 ] }
// _id:3 을 찾고 없다면 upsert : true 로 새로운 다큐먼트로 입력
> db.students2.update(
... { "_id":3 },
... { $set : { "_id": 3, "quiz1" : 0, "quiz2" : 0, "test1" : 0, "test2" : 0,
"modified" : "$$now"} },
... { upsert : true }
... )
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
// 새로운 다큐먼트 확인
> db.students2.find()
{ "_id" : 1, "quiz1" : 8, "test2" : 100, "quiz2" : 9, "modified" : ISODate("2020-
01-05T00:00:00Z") }
{ "_id" : 2, "quiz2" : 5, "test1" : 80, "test2" : 89, "modified" : ISODate("2020-
01-05T00:00:00Z") }
{ "_id" : 3, "modified" : "$$now", "quiz1" : 0, "quiz2" : 0, "test1" : 0, "test2"
: 0 }
>
```

remove

bulkwrite

db.collection.bulkWrite() 메소드는 bulk insert, update, remove 등을 지원한다.

Ordered vs Unordered Operations

Ordered 연산은 각 연산을 순차적으로 실행하면서 중간에 오류가 발생하는 경우 남은 연산을 수행하지 않고 결과를 반환한다.

Unordered 연산은 각 연산을 병렬로 처리하며 순서를 보장하지 않는다. 오류가 발생하는 경우 남은 연산을 모두 수행하고 결과를 반환한다.

기본설정은 ordered 이며 Unordered 가 처리 속도가 높다.

BulkWrite 에서 사용가능한 연산

- insertOne
- updateOne
- updateMany
- deleteOne
- deleteMany
- replaceOne

bulkWrite 예시

```
db.collection.bulkWrite(
   [
      { insertOne : <document> },
      { updateOne : <document> },
      { updateMany : <document> },
      { replaceOne : <document> },
      { deleteOne : <document> },
      { deleteMany : <document> }
   ],
   { ordered : false }
)
)
```

bulkWrite 내의 연산단위의 WriteConcern 은 설정할 수 없다.

$isolated 연산자

mongodb 는 multi-document 에 대한 쓰기 연산에 대해 각 다큐먼트에 대한 원자성을 보장하는데 $isolated 연산자를 통해 이를 격리하여 multi-document 쓰기 연산의 원자성을 보장할 수 있다.

하지만 다음과 같은 단점을 가지고 있다.

- 대상 collection 에 쓰기 잠금이 걸린다.
- 중간에 실패할 경우 원자성을 보장하지 않는다.

> mongodb 4.0 부터 지원을 중단하였으며 대시 transaction 을 통해 multi-document 의 원자성을 보장해야 한다.

배열(Array)

연산자

배열에 대한 쓰기연산을 위한 연산자

name	desc
$	쿼리 조건과 일치하는 첫번째 자리 표시자 역할
$[]	쿼리 조건과 일치하는 문서에 대한 배열의 모든 요소를 업데이트하는 자리 표시 자 역할
$[]	쿼리 조건과 일치하는 문서에 대해 arrayFilters 조건과 일치하는 모든 요소를 업데이트하는 자리 표시 자 역할
$addToSet	array 에 추가하려는 요소가 없는 경우에만 추가 한다.
$pop	array 의 처음 혹은 마지막 요소를 제거한다.
$pull	쿼리 조건에 일치하는 array 모든 요소를 삭제 한다.
$push	array 에 요소를 추가한다.

name	desc
$pullAll	쿼리 조건에 일치하는 array 중 매칭되는 요소 전부(복수) 를 삭제 한다.

연산자 수정자

배열 연산자 사용시 연산자와 함께 쓰이며 연산에 대해 수정을 할 수 있다.

name	desc
$each	$push 와 $addToSet 연사자를 에 대해 각 array 에 복수의 요소를 추가할 수 있다.
$position	$push 연산자에 대해 요소가 추가되는 위치를 지정할 수 있다.
$slice	$push 연산자에 대해 숫자 (0,negative,positive) 를 정해 수정되는 array 의 크기를 제한할 수 있다.
$sort	$push 연산자에 대해 추가되는 요소들을 정렬한다.

예제

```
> db.students2.update(
... { "_id":1 , score: 2},
... {$set : {"score.$" : 0} }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.students2.find()
{ "_id" : 1, "score" : [ 1, 0, 3 ] }
{ "_id" : 2, "score" : [ 4, 5, 6 ] }
>

> db.students2.update(
... { "_id":1 },
... {$push : { "addfield" : 100 } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.students2.find()
{ "_id" : 1, "score" : [ 1, 2, 3 ], "addfield" : [ 100 ] }
{ "_id" : 2, "score" : [ 4, 5, 6 ] }
>

> db.students2.update(
...     { _id: 1 },
...     { $addToSet: { colors: "c" } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.students2.find()
{ "_id" : 1, "score" : [ 1, 0, 3 ], "colors" : [ "c" ] }
{ "_id" : 2, "score" : [ 4, 5, 6 ] }
```