MongoDB: CRUD operations

CS185C: Introduction to NoSQL Databases

Suneuy Kim

Viewing available databases and collections

```
vagrant@vagrant-ubuntu-trusty-64:~$ mongo
MongoDB shell version: 3.2.11
connecting to: test
> show dbs
blog      0.078GB
comments  0.078GB
foo       0.078GB
local     0.078GB
media     0.078GB
test      0.078GB
> use test
switched to db test
> show collections
activities
audit100
blog
comments
foo
```

CRUD: Insert

CRUD operations

```
> post = {"title" : "My Blog Post","content" : "Here's my blog post.","date" : new Date()}
{
        "title" : "My Blog Post",
        "content" : "Here's my blog post.",
        "date" : ISODate("2016-09-19T22:21:08.970Z")
}
> db.blog.insertOne(post)
```

CRUD operations

```
{ "_id" : ObjectId("57e06509e233ba37d48215fa"), ← added
"title" : "My Blog Post",
"content" : "Here's my blog post.",
"date" : ISODate("2016-09-19T22:21:08.970Z")
}
```

insert(), insertOne(), insertMany

insert()

returns a WriteResult object for single inserts and a BulkWriteResult object for bulk inserts.

```
> db.foo.insert({"b": 3})
WriteResult({ "nInserted" : 1 })
```

```
> db.foo.insert([ {x:10 },{y:20} ] )
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

insert(), insertOne(), insertMany

- inserts a document/multiple documents into a collection and returns a document.

```
> db.foo.insertOne({a:20})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("57fd68c403f3d96b696f1e0b")
}
> db.foo.insertMany([{i:15},{j:25}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("57fd68fa03f3d96b696f1e0c"),
                ObjectId("57fd68fa03f3d96b696f1e0d")
        ]
}
```

insert(), insertOne(), insertMany

- The insert() method is deprecated in major driver so use insertOne() method and insertMany().
- The same thing applies to updateOne, updateMany, deleteOne, deleteMany, findOneAndDelete, findOneAndUpdate and findOneAndReplace.

Insert Validation

- The maximum BSON document size is 16 MB.
- To store documents larger than the maximum size, MongoDB provides the GridFS API
- To check the BSON size (in byte): Object.bsonsize(doc)

```
>doc = {"a":"b"}
{ "a" : "b" }
>Object.bsonsize(doc)
14
```

CRUD:Query

db.collection.find(query, projection)

Parameter	Type	Description
query	document	Specifies selection filter using query operators.
projection	document	Specifies the fields to return in the documents that match the query filter.
Returns:	A cursor to the document that matches to the query criteria.	

https://docs.mongodb.com/manual/reference/method/db.collection.find/

Data to test Query operations

```
> document = { "Type" : "Book",
"Title" : "Definitive Guide to MongoDB 3rd ed., The",
"ISBN" : "978-1-4842-1183-0",
"Publisher" : "Apress",
"Author" : ["Hows, David", "Plugge, Eelco", "Membrey, Peter", "Hawkins, Tim"]
}

> db.media.insertOne(document)
```

Data to test Query operations

```
db.media.insertOne(
{ "Type" : "CD",
"Artist" : "Nirvana",
"Title" : "Nevermind",
 "Tracklist" : [
 {
 "Track" : "1",
 "Title" : "Smells Like Teen Spirit",
 "Length" : "5:02"
 },
 {
 "Track" : "2",
 "Title" : "In Bloom",
 "Length" : "4:15"
 }
 ]
}
)
```

Data to test Query operations

```
> dvd1 = ( { "Type" : "DVD",
            "Title" : "Matrix, The", "Released" : 1999,
            "Cast" : ["Keanu Reeves","Carrie-Anne Moss", "Laurence Fishburne",
           "Hugo Weaving","Gloria Foster","Joe Pantoliano"] } )
> dvd2 = ( { "Type" : "DVD", Title : "Blade Runner", Released : 1982 } )
> dvd3 = ( { "Type" : "DVD", Title : "Toy Story 3", Released : 2010 } )
> db.media.insertOne(dvd1)
> db.media.insertOne(dvd2)
> db.media.insertOne(dvd3)
```

find()

> db.media.find()

Finds all of the documents in media

> db.media.find({Artist:"Nirvana"})

Finds all of documents where Artist is Nirvana.

> db.media.find({Artist:"Nirvana"},{Title:1})

To specify the name of the keys to be returned (e.g. Title), followed by a 1. (Only information from the Title field should be returned.)

> db.media.find({Artist:"Nirvana"},{Title:0})

To specify the name of the keys to be excluded. (All information except for the Type field.)

find()

```
> db.media.find({"Tracklist.Title": "In Bloom"})
```

The dot after the key name is to find information embedded in a document.

```
> db.media.find({"Author":"Membrey, Peter"})
```

When the value of the Author key is an array. The above query will find books of which authors include "Membrey, Peter".

```
> db.media.find({"Tracklist": {"Track":"1"}})
```

Sub-objects must match exactly. The above query does not return anything from the example collection due to Title and Length fields in addition to {"Track":"1"}.

sort, limit, and skip functions

```
> db.media.find().sort ( { Title:-1} ).limit(10).skip(20)
```

After skipping the first 20 result, return next 10 items only in the descending order of Title field.

Cursor Manipulation in Mongo Shell

```
> var myCursor = db.media.find( );
> while (myCursor.hasNext()) {
      var myDocument = myCursor.next();
      var title = myDocument.Title;
      print(title);
}
```

```
Definitive Guide to MongoDB 3rd ed., The
Definitive Guide to MongoDB 3rd ed., The
Matrix, The
Blade Runner
Toy Story 3
>
```

Capped Collection

- Normal collections
  - Created dynamically and automatically grow in size
- Capped collections
  - Created in advance and is fixed in size

  db.createCollection("audit100", {capped:true, size:20480, max:100})
  - Behave like circular queue: If it runs out of space, the new document will over write the oldest document.
  - Documents are stored in insertion order because

    Documents cannot be removed

    Updates causing documents to grow in size cannot be done
  - Good for logging and auto archiving data

Natural order

- Natural sort: documents in the order that they appear on disk.
  - Normal collection: natural order may != insertion order
  - Capped collection: natural order = insertion order
- With sort($natural:1), MongoDB returns documents in forwarding natural order. For a capped collection, this gives documents from oldest to newest.
- To get newer documents first from a capped collection,

```
db.aCappedCollection.find().sort({$natural:-1}
```

findOne()

> db.media.findOne()

It is generally advised to use the findOne() if only on result is expected – not to waste CPU time and memory.

Query and Projection Operators

- Refer to

https://docs.mongodb.com/manual/reference/operator/query/

for the category and details of these operators.

- Following slides present some of the representative operators with examples.

Comparison: $gt, $gte, $lt, $lte, and $ne

```
> db.media.find({Released:{$gte:2000}},{"Cast":0})
{ "_id" : ObjectId("57eeab366c2488577bd8795e"), "Type" : "DVD", "Title" : "Toy Story 3", "Released" : 2010 }
> db.media.find( {Released : {$gte: 1990, $lt : 2010}}, { "Cast" : 0 })
{ "_id" : ObjectId("57eea97b6c2488577bd8795c"), "Type" : "DVD", "Title" : "Matrix, The", "Released" : 1999 }
> db.media.find( { Type : "Book", Author: {$ne : "Plugge, Eelco"}})
```

To find a list of all books where the author does not include Eelco Plugge.

Logical: $or, $and, $not, $nor

- $in is used for a single key

> db.media.find ({ Released : {$in : ["2010","2009"] } }, { "Cast" : 0 })

- $or can be used for multiple keys

> db.media.find({ $or : [{ "Title" : "Toy Story 3" }, { "ISBN" :"978-1-4842-1183-0" }] })

- To combine $or operator with another query parameter. First documents matching to the first parameter are returned. And then $or condition is applied.

> db.media.find({ "Type" : "DVD", $or : [{ "Title" : "Toy Story 3" }, {

"ISBN" : "978-1-4842-1183-0" }] })

Logical: $or, $and, $not, $nor

```
db.media.find ( { Tracklist :   { "$elemMatch" : {Track:"1",
Title:"Smells Like Teen Spirit"} } }  )
```

v.s.

```
db.media.find ( { Tracklist : {$not:  { "$elemMatch" :
{Track:"1",  Title:"Smells Like Teen Spirit" } } } } )
```

$not may be performance heavy when the field of choice has many potential values.

Element: $exists and $type

```
> db.media.find ( { Author : {$exists : true } } )
```

Returns all documents with a key named Author

```
> db.media.find ( { Author : {$exists : false } } )
```

Returns all documents which do not have a key named Author

Element: $exists and $type

- Querying by data type is useful when dealing with highly unstructured data where data types are not predictable.
- To find based on the BSON Type

> db.media.find ({ Tracklist: { $type : 3 } }) → 3 means Embedded Object type

- When applied to arrays, $type matches any inner element that is of the specified BSON type.

> db.media.find({"Tracklist": {$type:4} })

{ "_id" : ObjectId("58a248d546a3cc91dfb83f37"), "Tracklist" : [[1, 2], [2, 3]] }

- BSON types: https://docs.mongodb.com/manual/reference/operator/query/type/

Evaluation: $mod, $regex and $text

```
> db.media.find ( { Released : { $mod: [2,0] } }, {"Cast" : 0 } )
```

Returns all documents of which Release field is an even-number integer. Only works for integers.

Evaluation: $mod, $regex and $text

Example data to test $regex

```
doc1 = { "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
doc2 = { "_id" : 101, "sku" : "abc789", "description" : "First line\nSecond line" }
doc3 = { "_id" : 102, "sku" : "xyz456", "description" : "Many spaces before     line" }
doc4 = { "_id" : 103, "sku" : "xyz789", "description" : "Multiple\nline description" }
db.products.insertOne(doc1);
db.products.insertOne(doc2);
db.products.insertOne(doc3);
db.products.insertOne(doc4);
```

Evaluation: $mod, $regex and $text

- Provides regular expression capabilities for pattern matching *strings* in queries
- MongoDB uses the Perl Compatible Regular Expression (PCRE).
- For case sensitive regular expression queries, if an index exists for the field, then MongoDB matches the regular expression against the values in the index.

```
> db.products.find( {sku: { $regex: /^ABC/i } } )
```

  - ^: prefix regular expression
  - ABC: pattern
  - i: case-insensitive option which can cause poor performance due to the number of searches to find the target in case-insensitive manner. Index cannot be used for case insensitive searches.

Evaluation: $mod, $regex and $text

- $regex vs. /pattern/ syntax

> db.products.find({sku:/^ABC/i}) will do the same as

> db.products.find({sku: { $regex: /^ABC/i } })

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
{ "_id" : 101, "sku" : "abc789", "description" : "First line\nSecond line" }
```

- Cannot use $regex inside an $in:

{ name: { $in: [/^acme/i, /^ack/] } }

Evaluation: $mod, $regex and $text

- $text performs a text search on the content of the fields indexed with a text index. A $text expression has the following syntax:

> db.texttest.find({$text:{$search:"cook"}},{_id:0,body:1 })

Just text fields for brevity.

```
> db.texttest.find({$text:{$search:"cook"}},{_id:0,body:1 })
{ "body" : "i want to cook dinner" }
{ "body" : "i am cooking lunch" }
>
```

Specifying an Array of Matches: $in, $nin, and $all

```
> db.media.find( {Released : {$in : ["1999", "2008", "2009"] } } )
```

- There exists an element of Released that matches any of the specified values in the array.

```
> db.media.find( {Released : {$nin : ["1999", "2008", "2009"] }})
```

- There does not exist an element of Released that matches any of the specified values or Released itself does not exist.

```
> db.media.find ( { Released : {$all : ["2010","2009"] } } )
```

- The array Released contains all the specified values.
- The above statement is equivalent to

```
db.media.find({ $and: [ { Released: "2010" }, { Released: " 2009" } ] })
{ "_id" : … , "Released" : [ "2010", "2009" ] }
{ "_id" : … , "Released" : [ "2009", "2010", "2011" ] }
```

The target field is an array: $size, $slice and $elemMatch

```
> db.media.find ( { Tracklist : {$size : 2} } )
```

- The array Tracklist has the specified number of elements in it.
- You cannot use the $size operator to find a range of sizes. For example, you cannot use it to find arrays with more than one element in them.

The target field is an array: $size, $slice and $elemMatch

- To limit an array field to a subset of the array for each matching result.

> db.media.find({Title: "Matrix, The"}, {Cast : {$slice: 3}})

For each document matching to {"Title" : "Matrix, The"},

limit the array called “Cast” to the first 3.

- $slice :-3 → last 3
- $slice:[2,3] → skip the first 2 items and limit the results to 3 from that particular point.
- $slice:[-5,4] → skip the last 5 items and limit the results to 4
- Note: $slice can be used $push operator.

Dataset to study $elemMatch in the next page

```
>nirvana1 =
( { "Type" : "CD", "Artist" : "Nirvana", "Title" : "Nirvana",
   "Tracklist" :
     [ { "Track" : "1", "Title" : "You Know You're Right", "Length" : "3:38"},
       {"Track" : "5", "Title" : "Smells Like Teen Spirit", "Length" : "5:02" }
    ]
} )
> nirvana2 =
( { "Type" : "CD", "Artist" : "Nirvana", "Title" : "Nirvana",
   "Tracklist" :
     [ { "Track" : "1", "Title" : "Smells Like Teen Spirit ", "Length" : "3:38"},
       {"Track" : "4", "Title " : " School"}
    ]
} )

> db.media.insertOne(nirvana1)
> db.media.insertOne(nirvana2)
```

Projection from Array: $slice and $elemMatch

```
> db.media.find (
  { "Tracklist.Title" : "Smells Like Teen Spirit","Tracklist.Track" : "1" } )
```

There exists Title "Smells Like Teen Spirit“ and Track "1". Not necessarily the Title’s Track number is 1.

```
> db.media.find (
 { Tracklist: { "$elemMatch" : { Title:"Smells Like Teen Spirit", Track : "1" } } } )
```

To match the entire document within the array Tracklist.

A document is returned if the array element contains the entire

```
{     "Track" : "1",
      "Title" : "Smells Like Teen Spirit",
  . . .
},
```

Quiz

Suppose a collection contains the following documents:

```
> db.quiz.find()
{ "_id" : ObjectId("58a764be17038a1a3dcc04e4"), "x" : 5 }
{ "_id" : ObjectId("58a764c717038a1a3dcc04e5"), "x" : 15 }
{ "_id" : ObjectId("58a764cb17038a1a3dcc04e6"), "x" : 25 }
{ "_id" : ObjectId("58a764d317038a1a3dcc04e7"), "x" : [ 5, 25 ] }
```

[Q] Which of the following query finds all documents where "x" is between 10 and 20 ? The expected answer is { "_id" : ... , "x" : 15 }.

1. db.quiz.find({"x":{$gt: 10, $lt:20}})
2. db.quiz.find({"x":{$elemMatch:{"$gt":10, "$lt":20 } }})
3. None of the above

Quiz - Answer

```
> db.quiz.createIndex({"x":1})
> db.quiz.find({"x":{$gt: 10, $lt:20}}).min({"x":10}).max({"x":20})
{ "_id" : ObjectId("58a764c717038a1a3dcc04e5"), "x" : 15 }
```

Use min() and max() to limit the index range traversed by the query to your $gt and $lt values.

CR**U**D:Update

updateOne()

```
db.collection.updateOne(
  <filter>,
  <update>,
  {
    upsert: <boolean>,
    writeConcern: <document>,
    collation: <document>
  }
)
```

- Updates a single document within the collection based on the filter.
- Upsert: update a record if a document is present or to insert the record if it isn't.

```
> db.media.updateOne(
{ "Title" : "Matrix, The"},
{$set: {" review":"good"}},
{upsert:true}
)
```

updateMany()

- updateMany() updates all matching documents in the collection that match the filter, using the update criteria to apply modifications.

> db.media.updateMany({ "Title" : "Matrix, The"},

{$set: {"comment":"my comment"}}, {upsert:true}) upserts all the occurrences

Atomic updates

- Updating a document is atomic: if two updates happen at the same time, whichever one reaches the server first will be applied, and then the next one will be applied.
- The last update will " win ".

Modifiers

- $inc
- $set
- Array Modifiers (cannot use for non-array)
  - $push
  - $pop
  - $pull, $pullAll
  - Positional array modification: $
  - $addToSet (using arrays as sets)

Modifiers - $inc, $set, $unset

> manga = ({ "Type" : "Manga", "Title" : "One Piece", "Volumes" : 612,"Read" : 520 })

> db.media.insertOne(manga)

> db.media.updateOne ({ "Title" : "One Piece"}, {$inc: {"Read" : 4} })

Increase the value of "Read" by 4. If "Read" doesn't exist, it will be created.

> db.media.updateMany({ "Title" : "Matrix, The" }, {$set : {"Genre":"Sci-Fi" } })

Replaces the value of Genre with "Sci-Fi". If Genre doesn't exist, it will be created.

> db.media.updateMany ({"Title": "Matrix, The"}, {$unset : { "Genre" : 1 } })
Deletes the Genre field from the document. If Genre doesn't exist, nothing is deleted.

Array Modifiers - $push

> db.media.updateOne({"ISBN" : "978-1-4842-1183-0"}, {$push: { Author : "Griffin, Stewie"} }) pushes "Griffin, Stewie" to the array.

> db.media.updateOne({"ISBN" : "978-1-4842-1183-0"}, {$push: { Title :"This isn't an array"} }) does not work for non-array.

> db.media.updateOne({ "ISBN" : "978-1-4842-1183-0" }, { $push: { Author : { $each: ["Griffin, Peter", "Griffin, Brian"] } } }) pushes multiple elements to the array.

> db.media.updateOne({ "ISBN" : "978-1-4842-1183-0" }, { $push: { Author : { $each: ["Griffin, Meg", "Griffin, Louis"], $slice: -3 } } }) pushes multiple elements to the array and keeps the last 3 elements in the array.

Array Modifiers - Using an array as a set $addToSet

> db.media.updateOne({ "ISBN" : "978-1-4842-1183-0" }, {$addToSet : { Author :"Griffin, Brian" } }) will not do anything if Author :"Griffin, Brian" exists.

> db.media.updateOne({ "ISBN" : "978-1-4842-1183-0" }, {$addToSet : { Author :{ $each : ["Griffin, Brian", "Griffin, Meg"] } } }) will add "Griffin, Meg“ since it does not exist in the array Author.

Array modifier - Removing from an array
$pop, $pull, $pullAll

> db.media.updateOne({ "ISBN" : "978-1-4842-1183-0" }, {$pop : {Author : 1 } }) removes the last element from the array.

> db.media.updateOne({ "ISBN" : "978-1-4842-1183-0" }, {$pop : {Author : -1 } }) removes the first element from the array.

Note:

- Any 0 or positive value removes the last element.
- Any negative value removes the first element.

Array modifier - Removing from an array
$pop, $pull, $pullAll

The $pull operator removes from an existing array all instances of a value or values that match a specified condition.

```
{  _id: 1,  fruits: [ "apples", "pears", "oranges", "grapes", "bananas" ],
   vegetables: [ "carrots", "celery", "squash", "carrots" ]}
> db.stores.updateMany({ },  { $pull: { fruits: { $in: [ "apples", "oranges" ] },
                                        vegetables: "carrots" } } )
> db.stores.find()
{ "_id" : 1, "fruits" : [ "pears", "grapes", "bananas" ], "vegetables" : [ "celery",
"squash" ] }
```

Array modifier - Removing from an array
$pop, $pull, $pullAll

- The $pullAll operator removes all instances of the specified values from an existing array.
- Unlike the $pull operator that removes elements by specifying a query, $pullAll removes elements that match the listed values.

{ _id: 1, scores: [0, 2, 5, 5, 1, 0] }

db.survey.updateOne({ _id: 1 }, { $pullAll: { scores: [0, 5] } })

removes all the occurences of 0 and5 from the array scores ,resulting

{ "_id" : 1, "scores" : [2, 1] }

Array modifier- $

Identifies an element in an array to update without explicitly specifying the position of the element in the array.

{ "_id" : 1, "grades" : [80, 85, 90] }
{ "_id" : 2, "grades" : [88, 85, 85] }
{ "_id" : 3, "grades" : [85, 100, 90] }

```
> db.students.updateMany({grades:85},{$set:{"grades.$":87 }}  )
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.students.find()
{ "_id" : 1, "grades" : [ 80, 87, 90 ] }
{ "_id" : 2, "grades" : [ 88, 87, 85 ] }
{ "_id" : 3, "grades" : [ 87, 100, 90 ] }
```

```
> db.students.updateMany({grades:85},{$set:{"grades":87 }}  )
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.students.find()
{ "_id" : 1, "grades" : 87 }
{ "_id" : 2, "grades" : 87 }
{ "_id" : 3, "grades" : 87 }
```

Atomic findAndModify

Modifies the document and returns it.

```
db.media.findAndModify( {
query: { "ISBN" : "978-1-4842-1183-0" },  ← to find target document
sort:{"Title":-1},  ← to sort the matching documents
update: {$set: {"Title" : " Different Title"} },  ← operation to be done
new:true  ← optional. To see the updated result
}
)
```

CRU**D**: Delete

Deleting Documents

- To delete full documents and collections (as compared to deleting data from a specific document e.g. using $pop)

```
> db.media.deleteOne( { "Title" : "Different Title" } )
> db.media.deleteMany({ "Title" : "Different Title" }  )
> db.media.deleteMany({}) deletes all documents from media.
```

Deleting the entire collection and the database

> db.newname.drop()

True

> db.dropDatabase() drops the current database you are working on.

{ "dropped" : "library", "ok" : 1 }