C.R.U.D. in MongoDB

Dr. Chetan Jaiswal,
Department of Computer Science,
Truman State University
cjaiswal@truman.edu

C.R.U.D.

- Create – INSERT (Mongo) - INSERT (SQL)
- Read – FIND (Mongo) – SELECT (SQL)
- Update – UPDATE (Mongo) – UPDATE (SQL)

One variation in Mongo is UPSERT

- Delete – REMOVE (Mongo) – DELETE (SQL)

C.R.U.D.

- Mongo Shell is simple javascript shell:

```
for(i=0;i<5;i++) print("Hello" + i);
```

- Type “help” for help tutorials
- Using built-in functions: BSON (http://bsonspec.org/)

```
obj={a:1, b:2, c:"Hello", d:["Apples", "Oranges"],e:new Date()}
```

INSERT

- Let document be:
- Use “insert” command
- Use mongoimport

```
{ author: 'joe',
title : 'Yet another blog post',
text : 'Here is the text...',
tags : [ 'example', 'joe' ],
comments : [
{ author: 'jim',
comment: 'I disagree'
},
{ author: 'nancy',
comment: 'Good post'
}
]
}
```

- Do it now:

Insert a document into the *fruit* collection with the attributes of "name" being "apple", "color" being "red", and "shape" being "round". Use the "insert" method.

findOne

- It is used to get one document out of the collection
- db.test.findOne(<Conditions>, <Projections>)
- db.test.findOne({name:”James”}, <Projections>)
- db.test.findOne({name:”James”}, {name:true, _id:false})// by default _id will always be present until explicitly specified “false”
- db.zips.distinct("state")

find

- It is used to get documents out of the collection
- db.test.find(<Conditions>, <Projections>)
- db.test.find({name:”James”}, <Projections>)
- db.test.find({name:”James”}, {name:true, _id:false})// by default _id will always be present until explicitly specified “false”
- db.test.find({name:”James”, zip:64110}, {name:true, _id:false})
- Assume documents to be like:
- how would you find all documents with type: essay and score: 50 and only retrieve the *student* field?

```
{
        "_id" : ObjectId("50906d7fa3c412bb040eb577"),
        "student_id" : 0,
        "type" : "exam",
        "score" : 54.6535436362647
}
```

find with $gt/$gte/$lt/$lte

- It is used to get documents based on range of values
- db.test.find(<Conditions>, <Projections>)
- db.test.find({score:{$gt:50}}, <Projections>)
- db.test.find(({score:{$gt:50, $lte:100}}, {name:true, _id:false})// by default _id will always be present until explicitly specified “false”
- Can also be used with strings
- db.test.find({name:{$gt:”D”}})//lexicographically using UTF8
- db.test.find({name:{$gt:”D”,$lte:”E”}})// will this document be part of the result?:

{_id:87, name: “DOG”, zip: 64110}

Unicode Transformation Format - 8 unicode characters

- Can name field have a number instead of string? Like: {_id:87, name: 60, zip: 64110}
- How will this query work now?: db.test.find({name:{$gt:”D”,$lte:”E”}})
- The MongoDB comparisons do not span between data types

regex, $type, $exists

- $exists: allows you to query documents which contains a particular attribute

db.people.find({profession: {$exists:true}})// or false

- $type: allows you to query documents where a particular field is of certain data type.

db.people.find({name:{$type:2}})//2 corresponds to string type

Refer: https://docs.mongodb.org/v3.0/reference/operator/query/type/

>db.posts.find({post_text:{$regex:“icecream"}})

>db.posts.find({post_text:/icecream/})

>db.posts.find({post_text:{$regex:“icecream",$options:"$i"}})//case insensitive

>db.products.find({ sku: { $regex: /789$/ } })//ends with 789

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

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

$or/$and

- db.people.find({ $or: [{name: { $regex: “e$” } }, { age: { $exists:true } }] })
- $and can be used similarly
- How would you find all documents in the *scores* collection where the *score* is less than 50 or greater than 90?
- How about this: db.scores.find({ score : { $gt : 50 }, score : { $lt : 60 } })

Querying inside arrays

- Same as normal documents
- Can use $in and $all
- db.accounts.find ({ favorites: { $all: [“pretzels”, beer”] } }) // finds all the docs where the array favorites contain pretzels and beer
- db.accounts.find ({ name: { $in: [“John”, “Howard”] } })// find all the docs where name contains John or Howard
- db.accounts.find ({ favorites: { $in: [“beer”, “icecream”] } })???

$set/$unset

- Assume a doc like: {_id:…, name: “Helen”}
- We would like to give age to Helen. How can we do that?
- a. db.people.update({ name: “Helen” }, { name: “Helen”, age: 17.5 })

Is this way good? Wouldn’t it delete everything else from the previous doc?

- b. db.people.update ({ name: “Helen” }, { $set: { age: 17.5 } })

This will just add one more field in the doc without disturbing the previous fields.

- $unset is just the opposite

More $ operators

- Assume a doc: { _id: 0, a: [1, 2, 3, 4] }
- db.collection.update ({ _id:0 }, { $set: { “a.2”: 5 } })
- db.collection.update ({ _id:0 }, { $push: { a: 6 } })
- db.collection.update ({ _id:0 }, { $pop: { a: 1 } })//right most
- db.collection.update ({ _id:0 }, { $pop: { a: -1 } })//left most
- db.collection.update ({ _id:0 }, { $pushAll: { a: [5, 6, 7] } })
- db.collection.update ({ _id:0 }, { $pull: { a: 5 } })//removes the value 5
- Similarly $pullAll
- Array can be treated as SET
- db.collection.update ({ _id:0 }, { $addToSet: { a: 8 } })//a will be now treated as a SET, its idempotent, meaning if you run above query multiple times it will only add 8 once

https://docs.mongodb.com/manual/reference/operator/update/set/

remove

- remove: it deletes the document specified
- db.people.remove({ name: “Alice” })
- db.people.remove({ name: { $gt:”M” } })
- How about: db.people.remove()???
- Instead give: db.people.remove({ })
- Or you can do: db.people.drop()
- Which is better?

Mmapv1 engine

- Mmap is a system call (refer mmap MAN page)
- File is mapped to exact sized virtual memory
- Provides collection level locking
- In place update

WiredTiger engine

- It is faster for a lot of workloads
- Document level concurrency/ no locking engine
- If two writes come for same document, one is rolled back and restarted again later
- Offers compression and indexes
- No inplace update
- Default engine is mmapv1: mongod –dbpath <some-new-directory> -storageEngine wiredTiger

indexes

- Faster reads and slower writes
- You can create composite/compound index
- db.test.ensureIndex ({ a: 1 }) // 1 represents ascending
- Same as using db.test.createIndex
- db.students.createIndex({"class":1,"student_name":1})
- db.collection.getIndexes()
- db.collection.dropIndex({ a:1 })
- db.students.createIndex({student_id:1,class_id:1},{unique:true})
- db.students.createIndex({student_id:1,class_id:1},{unique:true, sparse:true})//some docs may not have the field on which you are trying to create index on

Using explain

- db.collection.explain().find()// explains how the query will be processed
- db.collection.explain().help()
- db.example.example.find({ a:5, b:23 }), to know how mongodb is processing the query run with explain.
- db.example.explain().find({ a:5, b:23 })
- db.example.createIndex({ a:1 })// run explain after this and see if it is getting used
- Explain can used in 3 different modes: queryPlanner, executionStats, allPlansExecution
- Latter ones include the former ones!
- db.example.explain("executionStats").find({ a:5, b:23 })
- allPlansExecution: this mode shows what query optimizer has done, that is to explore all possible options
- Covered Queries: Mongodb programmers strive to satisfy all the front end queries using indexes. This makes the queries a lot faster. Use explain to find if the query is covered or not.

for(i=0;i<500000;i++) db.test.insert({a:i,b:500000-i,c:i%46})

Aggregation – Self Study

- Refer: https://docs.mongodb.org/manual/core/aggregation-introduction/