WEEK2 : CRUD Opeartions

Create Read Update Delete

Create Commands:

>> db.movieScratch.insertOne({JSON});

O/P

acknowledged: true

insertedId: ObjectID(“some\_obj\_id”)

>> db.movieScratch.insertMany([{},{},{}]);

insertMany does ordered insert meaning the order in which you have specified your doc list

ordered: false will insert doc in unordered manner

Note that ObjectId is created by Mongo but user can override it by passing value to “\_id” parameter while inserting doc/docs

>> db.movieScratch.drop();

\_id FIELD :

unique primary index on each doc

ObjectId: DATE | MAC ADDRESS | PID | COUNTER

\_ \_ \_ \_ | \_ \_ \_ | \_ \_ | \_ \_ \_

4 byte | 3 byte | 2 byte | 3 byte

A BSON ObjectID is a 12-byte value consisting of a 4-byte timestamp (seconds since epoch), a 3-byte machine id, a 2-byte process id, and a 3-byte counter. Note that the timestamp and counter fields must be stored big endian unlike the rest of BSON.

Read Commands:

>> db.movieDetails.find({ rated: “PG-13” }).count()

>> db.movieDetails.find({ rated: “PG-13”, year: 2009 }).count()

>> db.movieDetails.find({ “tomato.meter” : 100 }).pretty() // dot notation require double closed

Equality matches on arrays:

1. On the entire array
2. based on any element
3. based on specific element
4. complex match using operators

1.

>> db.movieDetails.find({ “writers” : [“Ethan Coen”, “Joel Coen”] }).count() // ordered search first Ethan and then Joel

2.

>> db.movieDetails.find({ “actor”: “Jeff Bridges” })

3.

>> db.movieDetails.find({ “actor.0” : “Jeff Bridges” }) // first actor should be matched, ordered preference

4. to be covered in later chapters

Cursors:

cursor be default returns 20 doc if not assigned to a variable

>> it // for iterating through cursor batches

>> c.obsLeftInBatch() // number of batches left in cursor

>> var c = db.movieDetails.find();

>> var doc = function() { return c.hasNext() ? next : null };

PROJECTIONS: what all fields you need in output

\_id is ALWAYS returned in o/p.

pass { \_id: 0 } to hide from output

>> db.movieDetail.find({ rated: “PG” }, { title: 1 }).pretty() // will print \_id and title

>> db.movieDetail.find({ rated: “PG” }, { writers: 0, actors: 0, \_id: 0 }) // will print everything other than \_id, writers, actors

COMPARISON OPERATOR:

$eq $ne

$gt $gte

$lt $lte

$in $nin // in and not in

>> db.movieDetail.find({ runtime: { $gt: 90 } }) // print doc where runtime > 90min

>> db.movieDetail.find({ runtime: { $gt: 90 }}, { title: 1 })

>> db.movieDetail.find({ runtime: { $gte: 90, $lte: 120 }}, { title: 1, runtime: 1, \_id: 0 }).pretty()

>> db.movieDetail.find({ “tomato.meter”: { $gte: 95 }, { runtime: { $gt : 90}}, { title: 1, runtime: 1, \_id: 0 }).pretty()

>> db.movieDetail.find({ rated: { $in : [“G”, “PG”, “PG-13”] } }).pretty()

ELEMENT OPERATORS:

$exists

$type

>> db.movieDetail.find({ “tomato.meter”: { $exists: true } })

>> db.movieDetail.find({ “\_id”: { $type: “string” } })

LOGICAL OPERATORS:

$or $nor

$and $not

>>db.movieDetail.find({ $or: [{ “tomato.meter”: { $gt : 95}} , { “metacritic” : { $gt: 80 }}]}, { \_id: 0 }).pretty()

>>db.movieDetail.find({ $and: [{ “tomato.meter”: { $gt : 95}} , { “metacritic” : { $gt: 80 }}]}, { \_id: 0 }).pretty()

Update Commands: also known as “upserts”

>>db.movieDetail.updateOne({ title: “The Martian” } , { $set: { “field” : “newValue” }} ) //update first doc matched

o/p:

acknowledged: true

matchedCount: 1

modifiedCount: 1

>> db.movieDetail.updateOne({ title: “The Martian” }, { $set: { “awards” : { “wins”: 8, nominations: 14, “text” : <some\_text> } } })

$set : update $unset: removes specified field from doc

$inc

$min

$rename

$setOnInsert :

>> db.movieDetail.updateOne({ title: “The Martian” }, { $inc: { “tomato.meter”: 3, “tomato.userReviews”: 25 } })

ARRAY METHODS

$push : adds item to array

$pull : removes all matched elements from array

$pullAll

$addToSet : add element only when it doesn’t exist in doc

$pop : removes first or last

>>db.movieDetail.updateOne({ “title: “The Martian” } , { $push:

{ reviews :

{ $each :

[

{

“rating: 0.5,

date: <new\_Date>,

}

]

}

}

});

if we don’t use $each : array element used inside $each will be added as a single element

>>db.movieDetail.updateOne({ “title: “The Martian” } , { $push:

{ reviews :

{ $each :

[

{

“rating: 0.5,

date: <new\_Date>,

}

],

$position: 0,

$slice: 5 // keep only 5 reviews after pushing

}

}

});

remove null value field

>>db.movieDetail.updateMany({ rated: null },{ $unset: { rated: “” }} )

UPSERT: if no matching value is found insert it

>>db.movieDetail.udpateOne({ “imdb.id”: detail.imdb.id }, { $set: detail }, { upsert: true } )

replaceOne