MongoDB

Paolo Bettelini

Contents

1 Introduction

1.1 MondoDB

MongoDB is a NoSQL document database (no relations)

A document is a way to store data in keys-values.
A collection contains multiple documents
A database contains multiple collections

BSON (Binary JSON)
BSON also supports Dates, Raw Binary types

1.2 Documents

Each document must have a unique "_id" value The type ObjectId is used by default to ensure unique values, however it is not mandatory. If no "_id" is provided, it will be automatically popoulated with an ObjectId

```
    "_id": ObjectId("5ec5f1b710ca9222e6a46c42")
    "_id": "10"
```

1.3 Data types

- Array
- Binary
- Boolean
- Code
- Date
- Decimal128
- Double
- Int32
- Int64
- MaxKey
- MinKey
- Null
- Object
- ObjectID
- BSONRegexp
- String
- BSONSymbol
- BSONMap
- Timestamp
- Undefined

1.4 Commands

1.5 Load from JSON

```
mongoimport --uri "<Cluster URI>"
            --drop <filename>.json
```

1.6 Export in JSON

```
mongoexport --uri "<Cluster URI>"
            --collection=<collection name>
            --out=<filename>.json
```

1.7 Load from BSON

```
mongorestore --uri "<Cluster URI>"
             --drop <dump>
```

1.8 Export in BSON

```
mongodump --uri "<Cluster URI>"
```

1.9 Show databases

```
show dbs
```

1.10 Use databases

```
use <database>
```

1.11 Show collections

```
show collections
```

2 Basic operations

2.1 Search for filter

returns a cursor

```
db.<collection>.find( {"state":"NY"} )
db.<collection>.find( {"state":"NY", "city": "ALBANY"} )
```

2.2 Search one

```
db.<collection>.findOne( {"state":"NY"} )
```

2.3 Iterate through the cursor

```
it
```

2.4 Count cursor length

```
db.<collection>.find( <query> ).count()
```

2.5 Insert

```
db.<collection>.insert( <document> )
db.<collection>.insert( [ <document>, <document>, ... ] )
```

2.6 Insert without order

When ordered=true as default, the insertions will stop at the first error

```
db.<collection>.insert( [ <document>, <document>, ... ], { "ordered": false } )
```

2.7 Update all

```
// Increment by 10 every "pop" field of every document matching the filter
db.<collection>.updateMany( {"city": "HUDSON"}, {"$inc": {"popoulation": 10}} )
```

2.8 Update one

```
db.<collection>.updateOne( <query>, <updates> )
```

2.9 Delete all documents

```
db.<collection>.deleteMany( <query> )
```

2.10 Delete one document

should only be used when filtering by "_id".

```
db.<collection>.deleteOne( <query> )
```

2.11 Drop collection

```
db.<collection>.drop()
// Deleting all collections will also result in the deletion of the container database
```

3 Syntax

3.1 Dollar operator

The $ → operator represents the value of a field rather than the field itself

3.2 Updates

```
{"$set": {"field1": value, "field2": value, ...}} // Sets the field value
{"$inc": {"field1": value, "field2": value, ...}} // Increment the field value
{"$push": {"field1": value, "field2": value, ...}} // Adds an element to an array
```

3.3 Queries

```
{"field": {"$eq": 60}, ...} // Equals
{"field": {"$ne": 60}, ...} // Not equals
{"field": {"$gt": 60}, ...} // Greater than
{"field": {"$lt": 60}, ...} // Less than
{"field": {"$gte": 60}, ...} // Greater than or equal
{"field": {"$lte": 60}, ...} // Less than or equal
```

3.4 Logic Operators

```
{"$not": {statmement}}
{"$and": [{statmement}, {statement}, ...]} // Default operator if not specified
{"$or": [{statmement}, {statement}, ...]}
{"$nor": [{statmement}, {statement}, ...]}
```

3.5 Expressions

```
{"$expr": {<expression>}}
```

3.6 Queries for arrays

```
{"array field": {"$size": 20}}
{"array field": {"$all": <array>}} // field must contains all the elements in the <array>
```

4 Operators

4.1 Projections

Projections are used to see only certain fields in the resulting output.

```
db.<collection>.find({<query>}, {<projection>})
```

```
{"field1": 1, "field2": 1}
0 -> exclude
1 -> include
```

4.2 Array operator

Maches array with at least one element that match the criteria.

```
"$elemMatch": {<field>:<value>}
```

```
db.companies.find(
  {"offices": {"$elemMatch": {"city": "Seattle"}} }
).count()
```

4.3 Querying sub-documents

```
{"doc.field1.subfield1": "value"}
{"array.0": "value"} // first element of array

db.inspections.find({"address.city": "NEW YORK"})

db.trips.find({"start station location.coordinates.0":
{"$lt": -74} })
```

4.4 Regular Expressions

```
{"field": {"$regex": "<regex>"}}
```

4.5 Aggregations

```
db.<collection>.aggregate([{...},{...},{...}])
```

The output of each stage is the output to the next stage.

Stages:

- $project
- $match
- $group

```
db.listingsAndReviews.aggregate([
    { "$project": { "address": 1, "_id": 0 }},
    { "$group": {
        "_id": "$address.country",
        "total ": {"$sum": "$price"}
    }}
])

db.listingsAndReviews.aggregate([
    { "$project": { "room_type": 1, "_id": 0 }},
    { "$group": {
        "_id": "$room_type",
        "count": {"$sum": 1}
    }}
])
```

4.6 Limit

```
db.<collection>.find(<query>).limit(1)
```

4.7 Sort

```
db.<collection>.find(<query>).sort(<sorting>)
```

Sorting:

```
{"value": 1} // Increasing
{"value": -1} // Decreascing
```

4.8 Indexes

```
db.<collection>.createIndex({"field": 1})
```

$$\begin{cases} +1, & \rightarrow \text{Increasing} \\ -1, & \rightarrow \text{Decreasing} \end{cases}$$

4.9 Compound Index

```
db.<collection>.createIndex({"field1":1, "field2":1})
```

4.10 Upsert

if there is a match (upsert == true), update. Otherwise, insert.

By default upsert is false

```
db.iot.updateOne({ "sensor": r.sensor, "date": r.date,
"valcount": { "$lt": 48 } },
      { "$push": { "readings": { "v": r.value, "t": r.time } },
     "$inc": { "valcount": 1, "total": r.value } },
{ "upsert": true })
```