mongo

09 November 2016 19:11

Commands

```
Use <db_name>
```
set current db, crete if not exists

1. **Insert**
```
db.movies.insertOne({"title":"force", year:2016})
```
2. **Find**
```
db.movies.find({})
db.movies.find({}).pretty()
db.movies.find({title:"Jaws"})
//to show specific fields from the response
db.movies.find({title:"Jaws"},{title:true,_id:false})
db.movies.findOne({})

db.movies.find({"year":{$gt:1985}})
db.movies.find({"year":{$lt:1985}})

//to check if a field exisis
db.movies.find({title:{$exists:true}})
db.movies.find({name:{$exists:true}})

//to check if field is of type string
db.movies.find({title:{$type:2}})
db.movies.find({title:{$regex:"^f"}})

//To combine 2 conditions using or
db.movies.find({$or:[{title:"Jaws"},{title:"Jurassic park"}]})

//to match multiple values of a field with value as array
db.movies.find({stars:{$all:["abc","acb"]}})

//to match multiple values of a field with value from array
db.movies.find({title:{$in:["Jaws","Jurassic park"]}})
```
3. **sort**
```
db.movies.find().sort({title:1})
//to skip the first 2 documents
db.movies.find().skip(2)
```
4. **count**
```
db.movies.count({})
db.movies.count({year:{$gt:1985}})
```
5. **Update**
```
//To update a document use the update, 1st arg is analogous to where clause & 2nd arg is the document, so the old one will be replaced with the new one
db.movies.update({title:"force"},{title:"force2", year:2017})
//To update specific field of a doc use $set
db.movies.update({title:"force"},{$set:{year:2013}})

//To increase value of a field having numeric value
db.movies.update({title:"force"},{$inc:{year:1}})

//To remove field from a document
db.movies.update({title:"force"},{$unset:{year:1}})

//To manipulate array elements
//To modify specific element in a array
db.movies.update({title:"force"},{$set:{"stars.1":"John"}})

//To add elements into an array use $push
db.movies.update({title:"force"},{$push:{heroes:"Abraham"}})

//to remove the right most element from the array use $pop
db.movies.update({title:"force"},{$pop:{heroes:1}})
//to remove the left most element from the array use $pop
db.movies.update({title:"force"},{$pop:{heroes:-1}})

//To push multiple elements use $pushAll
db.movies.update({title:"force"},{$pushAll:{heroes:["John","Abraham","Rambo"]}})

//To remove specific element from array
db.movies.update({title:"force"},{$pull:{heroes:"Abraham"}})

db.movies.update({title:"force"},{$pullAll:{heroes:["Abraham","John"]}})

//To add an element into array only if its not present
db.movies.update({title:"force"},{$addToSet:{heroes:"Abraham"}})

//Upsert: update if present or insert
db.movies.update({title:"Rock on"},{$set:{year:2018}},{upsert:true})

//If the query specifies incomplete info then insert will not happen
db.movies.update({year:{$gt:10}},{$set:{year:2018}},{upsert:true})

//TO update multiple docs set property {multi:true}
db.movies.update({year:{$gt:2017}},{$set:{year:2020}},{multi:true})
```
6. **Remove document**
```
//To delete a document use remove
db.movies.remove({year:2020})
//to remove all documents from collection
db.movies.remove({})
```

7. **drop collection**

```
db.movies.drop()

db.movies.find()
```

8. Aggregation

- **project:**

Passes along the documents with only the specified fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

```
db.movies.aggregate([{$project:{title:1,"_id":0}}])
db.movies.aggregate([{$project:{title:1,"star":"$title"}}])
```

- **match:**

Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

```
db.movies.aggregate([{$match:{title:"Force"}}])

db.movies.aggregate([{$match:{title:"Force"}},{$project:{title:1,"_id":0}}])

db.movies.aggregate({$limit:2})
```

- **unwind:**

Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.

```
db.movies.aggregate([{$unwind:"$heroes"}])

db.movies.aggregate([{$unwind:{path:"$heroes",preserveNullAndEmptyArrays: true}}])
```

- **group**

```
db.movies.aggregate( [ { $group : { _id : "$year" } } ] )

db.movies.aggregate( [ { $group : { _id : "$year", name:{$push:"$title"} } } ] )
```

- **lookup:**

Performs a left outer join to an unsharded collection in the same database to filter in documents from the “joined” collection for processing. The $lookup stage does an equality match between a field from the input documents with a field from the documents of the “joined” collection.

9. indexes:

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.
Indexes are special data structures that store a small portion of the collection’s data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index.
MongoDB creates a unique index on the _id field during the creation of a collection. The _id index prevents clients from inserting two documents with the same value for the _id field.

- **Index on a Single Field**

```
db.movies.createIndex({title:1})
db.movies.getIndexes()
```