CRUD Operations

CRUD Operations

Create

Read

Update

Delete

Creating Documents/ Insertng Records(RDBMS)

- **insertOne()**

insertOne will add an "_id" key to the document (if you do not supply one) and store the document in MongoDB.

Eg : > db.movies.insertOne({"title" : "Stand by Me"})

- **insertMany()**

Inserting multiple documents at a time.

```
➢ db.movies.insertMany([{"title" : "Ghostbusters"},
  ...                  {"title" : "E.T."},
  ...                  {"title" : "Blade Runner"}]);
```

- **Inserting an Array**

```
db.movies.insertOne(
{ name: "Movie2",
  Review: [{comment:"Great",rating:"5*"},{comment:"Average",rating:"3*"},{comment:"Above Average",rating:"3.5*"}]
})
```

Reading/selecting documents

db.collection.find() command is used to read/select the documents from collections

To Find single document

-> db.collection.findOne() // this will result the one document.

Eg :   db.movies.findOne()

To find all documents

-> db.collection.find()

Eg: db.movies.find()

Reading/selecting documents based on a criteria

we can pass the filtering condition as the first argument to the command, as in this example

- db.collection.find({“filter_condition”})
- db.movies.find({title:"E.T."})

Displaying only selected fields

we can pass the key names as the second argument to the find() command, as in this example

- ➢db.users.find({}, {"username" : 1, "email" : 1})
- To avoid a key from the result , we can give 0 instead of 1 , as in this example..

```
> db.users.find({}, {"fatal_weakness" : 0})
```

This can also prevent "_id" from being returned:

```
> db.users.find({}, {"username" : 1, "_id" : 0})
```

Pattern Search using regex

- Selecting only if the data contains only a specific text

  db.movies.find({title:{"$regex":"E.T"}}, {_id:0})
- Selecting only if the values contains a specific text at the beginning
  - db.movies.find({title:{"$regex":"^Stand"}}, {_id:0})
- Selecting only if the values contains a specific text at the end

```
db.movies.find(
{title:{"$regex":"too$"}},
{_id:0}
)
```

Case insensitive search using regex

- using $options

```
db.movies.find(
{title:{"$regex":"Too$", $options:"i"}},
{_id:0}
)
```

- The $options with 'I' parameter (which means case insensitivity)

Sorting and limiting output

- db.movies.find().sort({_id:-1}).limit(2)
  - Here the documents are sorted in the reverse order on _id and result is limited only to 2 documents.

Using less than and greater than

- ("imdb.rating" : {$lt : 2}})
- the second condition says the total number of IMDb votes should be more than 50,000. Add this condition to your query:("imdb.rating" : {$lt : 2}, "imdb.votes" : {$gt : 50000}}

```
db.movies.find(
  {"imdb.rating" : {$lt : 2}, "imdb.votes" : {$gt : 50000}}
)
```

Finding the distinct

- The **distinct()** function is used to get the distinct or unique values of a field with or without query criteria.

```
MongoDB Enterprise > db.movies.distinct("year")
[ 2015, "2015" ]
MongoDB Enterprise >
```

- The **distinct()** function can also be used along with a query condition. The following example finds all the unique ratings the films that were released in 1994 have received:
  - db.movies.distinct("rated", {"year" : 1994})
  - The first argument to the function is the name of the required field, while the second is the query expressed in the document format.

Finding the count

- count() helps in finding the count of documents in a collection

```
MongoDB Enterprise > db.movies.count();
4
```

- Without a query, this function will not physically count the documents. Instead, it will read through the collection's metadata and return the count. The MongoDB specification does not guarantee that the metadata count will always be accurate.
- When the function is provided with a query, the count of documents that match the given query is returned.

```
MongoDB Enterprise > db.movies.count({year:2015})
3
```

Conditional Operators

- $eq
  - to find documents with fields that match a given value.
  - *db.movies.find({num_mflix_comments:{$eq:"2"}})*
- $ne (Not Equal)
  - *db.movies.find({num_mflix_comments:{$ne:"3"}})*
- $gt (Greater than)
  - *db.movies.find({num_mflix_comments:{$gt:"1"}}).count()*
- $lt (Less than)
  - *db.movies.find({num_mflix_comments:{$lt:"3"}}).count()*

Conditional Operators

- $lte (less than or equal)
  - *db.movies.find({num_mflix_comments:{$lte:"3"}}).count()*
- $gte (Greater than or equal)
  - *db.movies.find({num_mflix_comments:{$gte:"1"}}).count()*
- $in (in) // used to find the document matches at least on of the value
  - *db.movies.find({rated:{$in:["G", "PG", "PG-13"] }})*
- $nin (Not in) //used to find the document which is not the values given
  - *db.movies.find({rated:{$nin:["G", "PG", "PG-13"] }})*

Logical Operators

- $and
- $or
- $nor
- $not

Logical Operators

- $and

Using the **$and** operator, you can have any number of conditions wrapped in an array and the operator will return only the documents that satisfy all the conditions. When a document fails a condition check, the next conditions are skipped.

```
db.movies.countDocuments (
   {$and :
      [{"rated" : "UNRATED"}, {"year" : 2008}]
   }
)
```

Logical Operators

- In MongoDB queries, the **$and** operator is implicit and included by default if a query document has more than one condition.

```
db.movies.countDocuments (
   {"rated": "UNRATED", "year" : 2008}
)
```

$or operator

- With the **$or** operator, you can pass multiple conditions wrapped in an array and the documents satisfying either of the conditions will be returned.

```
db.movies.find(
    { $or : [
        {"rated" : "G"},
        {"rated" : "PG"},
        {"rated" : "PG-13"}
    ]}
)
```

$or operator

- The **$in** operator is used to determine whether a given field has at least one of the values provided in an array, whereas the **$or** operator is not bound to any specific fields and accepts multiple expressions.

```
db.movies.find(
  {$or:[
     {"rated" : "G"},
     {"year" : 2005},
     {"num_mflix_comments" : {$gte : 5}}
  ]}
)
```

$nor Operator

- The **$nor** operator accepts multiple conditional expressions in the form of an array and returns the documents that do not satisfy any of the given conditions.

```
db.movies.find(
    {$nor:[
        {"rated" : "G"},
        {"year" : 2005},
        {"num_mflix_comments" : {$gte : 5}}
    ]}
)
```

$not Operator

- The **$not** operator represents the logical NOT operation that negates the given condition. Simply put, the **$not** operator accepts a conditional expression and matches all the documents that do not satisfy it.

```
db.movies.find(
    {"num_mflix_comments" :
        {$not : {$gte : 5} }
    }
)
```

Replacing Documents

- replaceOne

  accepts a query filter and a replacement document. The function finds the document that matches the criteria and replaces it with the provided document.

```
 db.users.replaceOne(
...
...   {"_id" : 5},
...
...   {"name": "Margaery Baratheon", "email": "Margaery.Baratheon@got.es"}
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
MongoDB Enterprise >
```

Replacing Documents

- replaceOne

  accepts a query filter and a replacement document. The function finds the document that matches the criteria and replaces it with the provided document.

```
 db.users.replaceOne(
...
...   {"_id" : 5},
...
...   {"name": "Margaery Baratheon", "email": "Margaery.Baratheon@got.es"}
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
MongoDB Enterprise >
```

upsert

- replace an existing document with a new one and, if the document does not already exist, insert the new document.

```
MongoDB Enterprise > db.users.replaceOne(
...
...   {"name" : "Margaery Baratheon"},
...
...   {"name": "Margaery Tyrell", "email": "Margaery.Tyrell@got.es"},
...
...   { upsert: true }
...
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

upsert

```
MongoDB Enterprise > db.users.replaceOne(
...
...   {"name" : "Tommen Baratheon"},
...
...   {"name": "Tommen Baratheon", "email": "Tommen.Baratheon@got.es"},
...
...   { upsert: true }
...
... )
{
        "acknowledged" : true,
        "matchedCount" : 0,
        "modifiedCount" : 0,
        "upsertedId" : ObjectId("6068907d421464e2522eca04")
```

Updating Documents

- updateOne()
- updateMany()
- Update commands are used to change the values of a field
- The first argument is the matching criteria
- The second argument is the value updation

```
db.movies.updateOne({"title":"Macbeth"},{$set:{year:"2015"}})
```

In this example the releasing year of Macbeth is updated to 2015

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Modifing more than one field

- To modify more than one field, the update expression ($set expression) can contain more than one field and value pair.

```
db.movies.updateOne(
  {"title" : "Macbeth"},
  {$set : {"type" : "movie", "num_mflix_comments" : 1}}
)
```

- If any of the field in the update statement doesn’t exist, it will be created.

Modifying more than one document

- updateOne, will be modifying only one document ,even if the matching condition results in more than one document
- To update all matching documents , we must use **updateMany()** command
  - *db.movies.updateMany({type:"movie"},{$set:{num_mflix_comments:"2"}})*

Update Operators

- $set
- $inc // increment operator
- $mul   //multiply
- $rename // rename
- $currentDate // current date
- $unset // removing fields from a document

$inc //increment operator

- The increment operator **($inc**) is used to increment the value of a numeric field by a specific number. The operator accepts a document containing pairs of a field name and a number. Given a positive number, the value of the field will be incremented and if a negative number is provided, the value will be decremented. It is obvious but worth mentioning that the **$inc** operator can only be used with numeric fields;

```
db.movies.findOneAndUpdate(
  {"title" : "Macbeth"},
  {$inc : {"num_mflix_comments" : 3, "rating" : 1.5}},
  {returnNewDocument : true}
)
```

$mul //multiply operator

- The multiplication **($mul**) operator is used to multiply the value of a numeric field by the given number. The operator accepts a document containing pairs of field names and numbers and can only be used on numeric fields.

```
db.movies.findOneAndUpdate(
  {"title" : "Macbeth"},
  {$mul : {"box_office_collection" : 16.3}},
  {returnNewDocument : true}
)
```

$mul //multiply operator

- The multiplication **($mul**) operator is used to multiply the value of a numeric field by the given number. The operator accepts a document containing pairs of field names and numbers and can only be used on numeric fields.

```
db.movies.findOneAndUpdate(
  {"title" : "Macbeth"},
  {$mul : {"box_office_collection" : 16.3}},
  {returnNewDocument : true}
)
```

$rename //rename operator

- the **$rename** operator is used to rename fields. The operator accepts a document containing pairs of field names and their new names.

```
db.movies.findOneAndUpdate(
  {"title" : "Macbeth"},
  {$rename : {"num_mflix_comments" : "comments", "imdb_rating" : "rating"}},
  {returnNewDocument : true}
)
```

- Rename the field in all documents
  - db.movies.updateMany({},{$rename:{"title":"name"}})

$unset //removing fields

- The **$unset** operator removes given fields from a document. The operator accepts a document containing pairs of field names and values and removes all the given fields from the matched document.

```
db.movies.findOneAndUpdate(
  {"title" : "Macbeth"},
  {$unset : {
    "created_date" : "",
    "last_updated" : "dummy_value",
    "box_office_collection": 142.2,
    "imdb" : null,
    "flag" : ""
  }},
  {returnNewDocument : true}
)
```

Deleting Documents

- deleteOne()
- deleteMany()

Both of these methods take a filter document as their first parameter. The filter specifies a set of criteria to match against in removing documents.

To delete the document with the "_id" value of 4, we use deleteOne in the *mongo* shell as illustrated here:

```
> db.movies.deleteOne({"_id" : 4})
```

Deleting Documents

For deleting more than one document , deleteMany() can be used

Eg :

```
-> db.movies.deleteMany({"title":"Stand by Me"})
{ "acknowledged" : true, "deletedCount" : 2 }
```

The above command deleted all documents which contains Title as “ Stand by Me”

Deleting Using findOneAndDelete()

findOneAndDelete(), which, as the name indicates, finds and deletes one document from the collection.

- It finds one document and deletes it.
- If more than one document is found, only the first one will be deleted.
- Once deleted, it returns the deleted document as a response.
- In the case of multiple document matches, the **sort** option can be used to influence which document gets deleted.

- Eg :

```
db.companyName.findOneAndDelete({name:"Yahya A"})
{
        "_id" : ObjectId("60654f32de49f71db773bb34"),
        "name" : "Yahya A",
        "company" : "Sony"
}
MongoDB Enterprise >
```