CRUD Operations-->C

CRUD Operations--->R Operation/Read Operation/Retrieve Operation/Find Operation:

--
We can read documents from the collection by using the following find methods.

1. find({query}) --->Returns all matched documents based on query.
2. findOne({query}) --->Returns one matched document based on query.

The argument,query is a simple javascript object.

These methods are related to collection and hence we have to call these methods on collection object.

db.collection.find()
db.collection.findOne()

These find methods are similar to select query in relational databases.

eg:
read all employees
read all employees where esal > 10000
read all employees where eaddr is Hyderabad
read all employees where eaddr is Hyderabad or esal > 10000
read all employees where eaddr is Hyderabad and esal > 10000
aggregate functions
logical operations
etc

All these things possible in MongoDB.

storedb --->database name
books --->collection name
books.json:

```
-----------
[
   {
     "title": "Linux in simple way",
     "isbn":  6677,
     "downloadable": false,
     "no_of_reviews": 1,
     "tags": ["os","freeware","shell programming"],
  "languages": ["english","hindi","telugu"],
  "author": {
         "name": "Shiva Ramachandran",
          "callname": "Shiv",
          "profile": {
                     "exp":8,
                      "courses":3,
                      "books":2
                      }
          }
 },
 {
  "title": "Java in simple way",
  "isbn": 1122,
  "downloadable": true,
  "no_of_reviews": 2,
  "tags": ["language","freeware","programming"],
  "languages": ["english","hindi","telugu"],
  "author": {
         "name": "Karhik Ramachandran",
          "callname": "Karthik",
          "profile": {
                     "exp":1,
                      "courses":2,
                      "books":3
                      }
          }
},
{
  "title": "Python in simple way",
  "isbn": 1234,
  "downloadable": false,
  "no_of_reviews": 5,
  "tags": ["language","freeware","programming"],
  "languages": ["english","hindi","telugu"],
  "author": {
        "name": "Daniel IA Cohen",
         "callname": "Dan",
         "profile": {
                   "exp":8,
                    "courses":7,
                    "books":6
                    }
        }
},
{
  "title": "Devops in simple way",
  "isbn": 6677,
  "downloadable": false,
  "no_of_reviews": 3,
  "tags": ["jenkins","git","cicd"],
  "languages": ["english","hindi","telugu"],
  "author": {
        "name": "Dhoni Chandra",
         "callname": "Dhoni",
         "profile": {
                   "exp":4,
                    "courses":4,
                    "books":4
                    }
        }
},
{
  "title": "MongoDB in simple way",
  "isbn": 6677,
  "downloadable": true,
  "no_of_reviews": 4,
  "tags": ["database","cloud","nosql"],
  "languages": ["english","hindi","telugu"],
  "author": {
        "name": "Sachin Tendulkar",
         "callname": "Sachin",
         "profile": {
                    "exp":6,
                     "courses":7,
                     "books":8
                     }
         }
},
{
  "title": "Oracle in simple way",
  "isbn": 6677,
  "downloadable": true,
  "no_of_reviews": 3,
  "tags": ["database","sql","relational"],
  "languages": ["english","hindi","telugu"],
  "author": {
        "name": "Virat Kohli",
         "callname": "kohli",
         "profile": {
                    "exp":2,
                     "courses":2,
                     "books":2
                     }
         }
},
{
  "title": "Shell Scripting in simple way",
  "isbn": 9988,
    "downloadable": true,
    "no_of_reviews": 1,
    "tags": ["programming"],
    "languages": ["english","hindi","tamil"],
    "author": {
            "name": "Rama Ramachandran",
             "callname": "Rama",
             "profile": {
                        "exp":8,
                         "courses":3,
                         "books":2
                         }
              }
   }
]
```

```
mongoimport --db storedb --collection books --file books.json --
jsonArray
```

Q1. List out all documents present in books collection?

```
> db.books.find().pretty()
> db.books.find({}).pretty()
```

Q2. Find total number of documents present in books collection?

```
> db.books.find().count()
```

Q3. List out first document present in books collection?

```
> db.books.findOne()
> db.books.findOne({})
```

Note: pretty() and count() methods can be used on find() result but not on findOne() result.

```
> db.books.findOne().pretty()
uncaught exception: TypeError: db.books.findOne(...).pretty is not a
function :
@(shell):1:1
> db.books.findOne().count()
uncaught exception: TypeError: db.books.findOne(...).count is not a
function :
```

Q4. List out all documents from books collection where "downloadable" is false?

```
> db.books.find({downloadable: false}).pretty()
```

Q5. List out all documents from books collection where no_of_reviews is 3.

```
> db.books.find({no_of_reviews: 3}).pretty()
```

Querying Nested Documents:

If the value of a field is again a document, then that nested property value can be accessed by using dot operator. In this case, key must be enclosed within quotes.

Q1. List out all documents from books collection where author's call name is kohli?

```
> db.books.find({author.callname: "kohli"}).pretty() ==>invalid
> db.books.find({"author.callname": "kohli"}).pretty() ==>valid
```

Q2. List out all documents from books collection where author's profile contains exactly 2 courses?

```
> db.books.find({"author.profile.courses": 2}).pretty()
```

Querying Array elements:

It is exactly same as exact match

Q1. List out all documents where 'tags' array contains 'programming' element?

```
> db.books.find({tags: "programming"}).pretty() ===>Query array elements
```

Q2. List out all documents where 'languages' array contains 'telugu' element?

```
> db.books.find({languages: "telugu"}).pretty() ===>Query array elements
```

Querying Array itself:

Q1. List out all documents where 'tags' array contains only one element 'programming'?

```
> db.books.find({tags: ["programming"]}).pretty() ===>Query array itself
```

Q2. List out all documents where 'tags' array is:
["language","freeware","programming"]

```
> db.books.find({tags: ["language","freeware","programming"]}).pretty()
```

Here order of elements inside array is important and even case also.

db.books.find({tags: ["language","programming","freeware"]}).pretty()
It won't return any document because the given order not matched with any document.

Note:
> db.books.find({tags: "programming"}).pretty() ===>Query array elements
> db.books.find({tags: ["programming"]}).pretty() ===>Query array itself

It is somthing like we are finding fruit in a basket not basket but in the second case we are trying to find basket itself.

Query Operators:

We can use operators in our queries.
Every operator prefixed with $ symbol to indicate that it is operator butnot field or value. By seeing $ symbol, MongoDB server will execute the corresponding operator functionality.

1. Comparison Query Operators:

$eq, $ne, $gt, $gte, $lt, $lte, $in, $nin

$eq ---> Equality:

The $eq operator matches documents where the value of the field is equal to specified value.

Syntax: db.collection.find({ field: {$eq: value} })

It is exactly same as
db.collection.find({field: value}) ==>It is the short cut way

Case-1: Equals to Specific Value:

Q1. Select all documents from books collection, where no_of_reviews is 3.

```
> db.books.find({ no_of_reviews: { $eq: 3}}).pretty()
> db.books.find({ no_of_reviews: 3}).pretty()
```

Case-2: Field in Nested Document equals a value:
--
Q1. Select all documents from the books collection where author profile contains 2 courses?

```
> db.books.find({"author.profile.courses": {$eq: 2}}).pretty()
> db.books.find({"author.profile.courses": 2}).pretty()
```

Case-3: Array element equals a value:

Q1. Read all documents from the books collection where 'tags' array contains 'database' element?

```
> db.books.find({tags: {$eq: "database"}}).pretty()
> db.books.find({tags: "database"}).pretty()
```

Case-4: Equals Array Value directly:

Q1. Select all documents from books collection where tags array is exactly equal to ["language","freeware","programming"].

```
> db.books.find({tags: {$eq:
["language","freeware","programming"]}}).pretty()
> db.books.find({tags: ["language","freeware","programming"]}).pretty()
```

$ne operator:

----

ne ---> means not equal

We can use $ne operator to select all the documents where the value of the field is not equal to specified value.

Syntax: db.collection.find({filed: {$ne: value}})

Q. To select all documents from books collection where no_of_reviews is not equal to 3.

```
> db.books.find({no_of_reviews: {$ne: 3}}).pretty()
```

Note: If the specified field not available, such documents also will be included in the result.

$gt operator:

$gt ---> means greater than

We can use $gt operator to select all documents where the value of field is greater than specified value.

Syntax: db.collection.find({field: {$gt: value}})

Q1. Select all documents from books collection where the no_of_reviews is greater than 3.

```
> db.books.find({no_of_reviews: {$gt: 3}}).pretty()
```

$gte operator:

$gte ----> means greater than or equal to

We can use $gte operator to select all documents where the value of field is greater than or equal to specified value.

Syntax: db.collection.find({field: {$gte: value}})

Q1. Select all documents from books collection where the no_of_reviews is greater than or equal to 3.

```
> db.books.find({no_of_reviews: {$gte: 3}}).pretty()
```

$lt operator:

$lt ---> means less than

We can use $lt operator to select all documents where the value of field is less than specified value.

Syntax: db.collection.find({field: {$lt: value}})

Q1. Select all documents from books collection where the no_of_reviews is less than 3.

> db.books.find({no_of_reviews: {$lt: 3}}).pretty()

$lte operator:

$lte ---> means less than or equal to

We can use $lte operator to select all documents where the value of field is less than or equal to specified value.

Syntax: db.collection.find({field: {$lte: value}})

Q1. Select all documents from books collection where the no_of_reviews is less than or equal to 3.

> db.books.find({no_of_reviews: {$lte: 3}}).pretty()

$in operator:

We can use $in operator to select all documents where the value of a field equals any value in the specified array.
It is something like python membership operator.

Syntax: db.collection.find({field: {$in: [value1,value2,...,valueN]}})

Q1. Select all documents from the books collection where the no_of_reviews is 1 or 4 or 5?

> db.books.find({no_of_reviews: {$in: [1,4,5]}}).pretty()

Q2. Select all documents from the books collection where the tags array contains either database or programming.

```
> db.books.find({tags: {$in: ["database", "programming"]}}).pretty()
```

$nin operator:

$nin means not in
It is inverse of in operator

Syntax: db.collection.find({field: {$nin: [value1,value2,..,valueN]}})

We can use $nin operator to select all documents where:

1. The field value not present in the specified array.
2. The field does not exist.

Q1. Select all documents from books collection where the no_of_reviews is not 1 or 4 or 5?

```
> db.books.find({no_of_reviews: {$nin: [1, 4, 5]}}).pretty()
```

Q2. consider the query

```
> db.books.find({exams: {$nin: [1, 4, 5]}}).pretty()
```

We will get all documents, because exams field is not available in any document.

Note:
$in result + $nin result = total no of documents

```
> db.books.find({no_of_reviews: {$nin: [1, 4, 5]}}).count()
4
```

```
> db.books.find({no_of_reviews: {$in: [1, 4, 5]}}).count()
4
> db.books.find().count()
8
```

Logical Query Operators:

$or, $nor, $and, $not

oldtamirindrecepie

olden days tammarend pickle

sir take session on procastination or time manegment

If you are not having goal in your life, you will part of some other's goal.

One Movie Dialogue sir ,If we not working for our goals then others will use to full fill their Dreamzs.

Logical Query Operators:

$or, $nor, $and, $not

$or operator:

$or performs logical OR operation on an array of two or more expressions(conditions) and selects the documents that satisfy atleast one of the expression(condition)

Syntax: {$or: [{expression1},{expression1},..{expressionN}]}

Q1. Select all documents where either no_of_reviews >3 or tags array contains programming element?

c1: {no_of_reviews: {$gt: 3}}
c2: {tags: "programming"}

```
> db.books.find({$or: [{no_of_reviews: {$gt: 3}}, {tags: "programming"}
]}).pretty()
```

Q2. Select all documents where either no_of_reviews is less than 3 or downloadable is true or author profile contains atleast 2 books?

c1: {no_of_reviews: {$lt: 3}}
c2: {downloadable: true}
c3: {"author.profile.books": {$gte: 2}}

```
> db.books.find({$or: [{no_of_reviews: {$lt: 3}}, {downloadable: true},
{"author.profile.books": {$gte: 2}}]}).pretty()
```

$nor operator:

It is inverse of $or operator.

$or --->Atleast one condition satisfied
$nor --->neither condition satisfied i.e all conditions fails

Syntax: {$nor: [{expression1},{expression1},..{expressionN}]}

$nor performs a logical NOR operation on an array of one or more expressions(conditions) and selects the documents that fail all query expressions in the array.

eg:
```
c1: { no_of_reviews: {$gt: 3}}
c2: { downloadable: true}
```

```
> db.books.find({$nor: [{ no_of_reviews: {$gt: 3}}, { downloadable:
true}]}).pretty()
```

It will select all documents where
1. The no_of_reviews is less than or equal to 3 (i.e not greater than 3)
AND
2. downloadable is false
3. documents does not conain no_of_reviews and downloadable fields

Note: $or results + $nor results = total no of documents

```
> db.books.find({$or: [{ no_of_reviews: {$gt: 3}}, { downloadable:
true}]}).count()
5
> db.books.find({$nor: [{ no_of_reviews: {$gt: 3}}, { downloadable:
true}]}).count()
3
> db.books.find().count()
8
```

$and operator:

$and performs logical AND operation on an array of one or more expressions and selects the documents that satisfy all expressions in the array. i.e all conditions should be satisfied

Syntax: {$and: [{expression1},{expression1},..{expressionN}]}

Q. Select all documents where the no_of_reviews is greater than or equals to 3 and downloadable is true?

```
c1: {no_of_reviews: {$gte: 3}}
c2: {downloadable: true}
```

```
> db.books.find({$and: [{no_of_reviews: {$gte: 3}}, {downloadable:
true}]}).pretty()
```

Assignment:

students.json:

```
[
 {
   "name": "A",
   "marks": 10
 },
 {
   "name": "B",
   "marks": 20
 },
 {
   "name": "C",
   "marks": 30
},
{
  "name": "D",
  "marks": 40
},
{
  "name": "E",
  "marks": 50
},
{
  "name": "F",
  "marks": 60
},
{
  "name": "G",
  "marks": 70
},
{
  "name": "H",
  "marks": 80
},
{
  "name": "I",
  "marks": 90
},
{
  "name": "J",
  "marks": 100
}
]
mongoimport --db storedb --collection students --file students.json --
jsonArray
```

Q1. Select all students where marks are less than 85 and greater than 45?

```
c1: {marks: {$lt: 85}}
c2: {marks: {$gt: 45}}
> db.students.find({$and: [{marks: {$lt: 85}}, {marks: {$gt: 45}}]}).pretty()
```

Q2. Select all students where marks are less than 50 and greater than or equal to 35?

```
c1: {marks: {$lt: 50}}
c2: {marks: {$gte: 35}}
> db.students.find({$and: [{marks: {$lt: 50}}, {marks: {$gte:
35}}]}).pretty()
```

Shortcut for AND operation:

MongoDB provides an implicit AND operation when specifying a comma separated list of expressions.

```
Normal way: {$and: [{expression1},{expression1},...]}
Shortcut way: {expression1, expression2,... }
```

Q. Select all documents where the no_of_reviews is greater than or equals to 3 and downloadable is true?

```
c1: {no_of_reviews: {$gte: 3}}
c2: {downloadable: true}

> db.books.find({$and: [{no_of_reviews: {$gte: 3}}, {downloadable:
true}]}).pretty()
```

shortcut way:

```
> db.books.find({no_of_reviews: {$gte: 3}, downloadable: true}).pretty()
```

Limitation of this shortcut:

If the conditions are on the same field then this short cut won't work.

Q. List out all students whose marks are >=50 and <= 90?

```
> db.students.find({marks: {$gte: 50}, marks: {$lte: 90}}).pretty()
{ "_id" : ObjectId("5fed4de50df1d8f23c0f678c"), "name" : "B", "marks" :
20 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f678d"), "name" : "I", "marks" :
90 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f678e"), "name" : "C", "marks" :
30 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f678f"), "name" : "A", "marks" :
10 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6791"), "name" : "E", "marks" :
50 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6792"), "name" : "G", "marks" :
70 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6793"), "name" : "D", "marks" :
40 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6794"), "name" : "F", "marks" :
60 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6795"), "name" : "H", "marks" :
80 }
```

Reason: In Javascript object, duplicate keys are not allowed. If we are trying to add duplicate keys then old value will be replaced with new value.

```
{marks: {$gte: 50}, marks: {$lte: 90}}
```

It will become

{marks: {$lte: 90}}

Solution: we should use $and operator

```
> db.students.find({ $and: [ {marks: {$gte: 50}},{marks: {$lte:
90}}]}).pretty()
{ "_id" : ObjectId("5fed4de50df1d8f23c0f678d"), "name" : "I", "marks" :
90 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6791"), "name" : "E", "marks" :
50 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6792"), "name" : "G", "marks" :
70 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6794"), "name" : "F", "marks" :
60 }
{ "_id" : ObjectId("5fed4de50df1d8f23c0f6795"), "name" : "H", "marks" :
80 }
```

Q. What is the difference between these two quires?

1. db.students.find({ $and: [{marks: {$gte: 50}},{marks: {$lte: 90}}]}).pretty()
2. db.students.find({marks: {$gte: 50, $lte: 90}}).pretty()

Both will provide same results.
In this we are using field only once and shortcut also will work.

$not operator:

It is just to perform inverse operation.

Syntax: { field: {$not: {operator expression}}}
eg: { marks: {$not: {$gte: 10}}}

$not operation performs logical NOT operation on the specified operator expression and selects the documents that do not match operator expression. This includes the documents that do not contain the field.

eg:
c1: { no_of_reviews: {$gt: 3}}

db.books.find({ no_of_reviews: {$not: {$gt: 3}}}).pretty()
It returns all documents where

1. The no_of_reviews not greater than 3(i.e less than or equal to 3)
2. no_of_reviews field does not exist.

Element Query Operators:
----
1. $exists   2. $type

1. $exists:

Syntax: {field: {$exists: <boolean>}}

If <boolean> is true, then it selects all documents that contain specified field even the value of the field is null.

If <boolean> is false, then it selects all documents that do not contain specified field.

Q1. Select all documents which contains no_of_reviews field.

> db.books.find({no_of_reviews: {$exists: true}}).pretty()

Q2. Select all documents which does not contain no_of_reviews field.

```
> db.books.find({no_of_reviews: {$exists: false}}).pretty()
```

case study:

```
db.students.insertOne({name: "Durga", marks: 100, gf: "Sunny"})
db.students.insertOne({name: "Ravi", marks: 20, gf: "Mallika"})
```

Q1. Select all students who are having gf field?

```
> db.students.find({ gf: {$exists: true}}).pretty()
```

Q2. Select all students who are not having gf field?

```
> db.students.find({ gf: {$exists: false}}).pretty()
```

Q3. Select all students who are having the gf, but still marks are greater than 70?

```
c1: {gf: {$exists: true}}
c2: { marks: {$gt: 70}}

> db.students.find({ gf: {$exists: true}, marks: {$gt: 70} }).pretty()
> db.students.find({ $and: [{gf: {$exists: true}}, { marks: {$gt: 70}}
]}).pretty()
+
```

----
2. $type operator:

$type operator selects the documents where the value of the field is of a particular type.
We have to specify the type as BSON type.

This operator is very helpful, whenever we are dealing with large volumes of unstructured data where types are unpredictable.

Syntax-1: Querying for a single type

{field: {$type: <BSONType>}}

We can specifiy either number or alias for the BSON Type.
eg:

{field: {$type: "int"}}
{field: {$type: "string"}}

Syntax-2: Querying for multiple types

{field: {$type: [<BSONType1>, <BSONType2>, <BSONType3>,...]}}

DATATYPES:

Table:
BSON Type-------->Number ---------- >alias
==
Double-------->1----------->"double"
String-------->2----------- >"string"
Object-------->3----------- >"object"
Array-------- >4----------- >"array"
BinaryData---->5 ----------->"binData"
ObjectId------>7 ---------- >"objectId"
Boolean------->8 ----------->"bool"