**READ OPERATIONS**

In mongodb we have mainly two types of read operations:

* **findOne(query, projection):** Used to find a single document.
* **find(query, projection):**  Used to find multiple documents. (by default returns cursor)

NOTE: Array element can be matched using simple match query in includes in manner.

e.g. a document like this:

{

\_id:objectId(“somerandomobjectid”),

hobbies: [‘cricket’, ‘football’, ‘basketball’]

}

can be matched by the queries:

{hobbies: ‘basketball’} (includes in manner)

{hobbies: [‘cricket’, ‘football’, ‘basketball’] } (exact manner)

OFFICIAL QUERY & PROJECTION OPERATORS DOCS : [Query and Projection Operators - MongoDB Manual v8.0](https://www.mongodb.com/docs/manual/reference/operator/query/)

**Query Operators:**

1. **Comparision Operators ($eq, $ne, $gt, $lt, $gte, $lte, $in, $nin)**
2. **Logical Operators ($or, $nor, $and, $not)**
3. **Element Operators ($exists, $type)**
4. **Evaluation Operators ($regex, $expr)**
5. **Array Operators ($size, $all, $elemMatch)**

* **$eq:** used for exact matching. Similar to {a: “match string”}.
* **$ne:** used for “not equals to” matching.

{name: {$ne: “Harshit”}}

[returns document(s) where name is not equals to ‘Harshit’]

* **$gt/$gte:** used for “greater than/greater than equals to” matching.

{age: {$gt: 17}}

[return document(s)where age is greater than 17]

* **$lt/$lte:** used for “less than/less than equals to” matching.

{age: {$lt: 17}}

[return document(s)where age is less than 17]

* **$in/$nin:** used to match documents based on an array of elements.

{age: {$in: [17,18,22,45]}}

[return document(s)where age either of 17,18,22,45]

{age: {$nin: [17,18,22,45]}}

[return document(s)where age niether of 17,18,22,45]

* **$or:** By default the conditions works in ‘AND (all must be true)’ pattern to make conditions works in ‘OR (at least one must be true)’ pattern we utilize this operator.

(default AND matching)

{profession: “sports”, age: 18}}

[return all the document(s) where the profession is “sports” and the age is 18]

(OR Matching)

{$or: [{profession: “sports”}, { age: 18}]}}

[return all the document(s) where the either the profession is “sports” or the age is 18 or both are true]

* **$nor:** This operator is simply the vice versa of the $or operator it will match all the documents where neither of the condition is true.

{$nor: [{profession: “sports”}, { age: 18}]}}

[return all the document(s) where the neither the profession is “sports” nor the age is 18]

* **$and:** By default all the matching occurs in “AND” manner, but the $and operator is required when we have to use two or more conditions on the same field (and we know that object can have two fields of same name).

(without $and)

{age: {$gt: 17}, age: {$lt: 19}}

Above query will be equivalent to:

{age: {$lt: 19}})

(with $and)

{$and: [{age: {$gt: 17}}, { age: {$lt: 19}}]}

* **$exists:**  This operator is used to query the documents based upon the field exists of not.

{nationality: {$exists: false/true}}

[return all the documents where the nationality field exists if true, and documents where nationality field do not exists if false.]

* **$type:** This operator is used to query the documents based upon the type of certain field.

{ nationality: {$type : ‘string’}} // $type value must be bsonType.

[returns all the documents where the type of nationality value is string.]

* **$regex :** This operator is used to query document based regex expression on the field.

{ description: { $regex : /I m fine/ } }

[return all the documents whose description contains ‘I m fine’ text.]

* **$expr:** This operator is used to query documents based upon some conditions on the fields inside the documents. (it is generally used for writing more complex expressions in conjunction with other operatores.)

({$expr : {$gt: ['$visitors', '$expectedVisitors']}}) [returns all the documents where the visitors is greater than expectedVisitors]

* **$size:** This operator is used to query documents depending upon the size of the array.

{cars: {$size: 4}}

[return all the documents where the size the cars array is 4]

* **$all:**  This operator is used to query documents based on the array, no matter what the position of elements in the array.

{cars: {$all : [‘porche’, ‘bmw’, ‘audi’]}}

[return all the documents that contains all of the array elements irrespective of its order.]

* **$elemMatch:**  This operator is used to query documents based on a particular embedded document in the array.

{‘hobbies’: {$elemMatch: {title: ‘cricket’, frequency: 6}}}

[return all the document where the hobbies array have at-least one document which have title: ‘cricket’ and frequency: 6, the document may have additional fields]

**Cursor:**

The cursor is a mapping pointer retuned by the .find() query to manage the resources by not fetching millions of the documents at once, but loading all the documents efficiently in the memory from the database and showing only 20 documents (by default) as cursor at once. It is efficient when querying a larger dataset/ collection.

**Cursor Methods:**

* **next():**  The next method is use to get the next document present in the cursor.
* const cursor = db.<collection>.find()

[cursor holding first 20 documents]

* cursor.next()

[returns the next document i.e. 21’st document]

* **hasNext():** The hasNext method is use to identify whether the next document exists or not.
* const cursor = db.<collection>.find()

[cursor holding first 20 documents]

* cursor.hasNext()

[return true/false if 21’st document exists or not.]

* **count():**  The count method is used to get the count of all the matched documents that are loaded into the memory from the database for the operation.

**e.g:** Imagine if a collection have 10000 documents and we executed .find()

* const cursor = db.<collection>.find()

[return cursor to 20 documents]

* cursor.count()

[return 10000, the number of documents matching the query.]

* **toArray():**  The toArray method is used to force the cursor to return all the matching documents instead of returning only 20.
* const cursor = db.<collection>.find()
* cursor.toArray()

[returns all the documents that matches the query.]

* **forEach(()=>{}):** The forEach is similer to toArray method and also used to force the cursor to return all the matching documents instead of returning only 20. In addition with that we can mutate the resulting documents in for Each.
* const cursor = db.<collection>.find()
* cursor.forEach(doc => { delete doc.\_id })

[returns all the documents without the \_id]

* **sort(criteria):**  This method is used to sort the document based upon the mentioned criteria 1 => ascending , -1 decending.
* const cursor = db.<collection>.find()
* cursor.sort({‘score’: 1})

[returns a cursor to 20 documents with lowest score document first]

* cursor.sort({score: 1, matchesPlayed: -1})

[sorting can be done on more than one fields]

* **skip(count):** The skip method is used to skip a certain number of document in the cursor before returning the value.
* const cursor = db.<collection>.find()
* cursor.skip(10)

[Skip the first 10 documents and return the cursor to next 20 documents]

* **limit(count):**  This method is used to limit the number of documents retuned by the cursor.
* const cursor = db.<collection>.find()
* cursor.limit(10)

[return cursor to only 10 documents]

Note: These cursor methods can be used in conjunction with each other for example.

* db.<collection>.find().sort({\_id: 1}).limit(10)

[return the 10 documents with the oldest \_id field]

* db.<collection>.find().skip(100).limit(10)

[returns 10 documents after skipping first 100 documents e.g. applying pagination]

**Projection:**

Projection in MongoDB read queries is a mechanism used to retrieve only the required fields from matching documents. This approach results in faster query performance, reduced data transfer, and ensures that only the necessary data is fetched. \_id is always the part of the fetched data, unless excluded explicitly.

**Projection Values :**

* **1 (include):**  This indicates to include this field in the data retrieval.
* db.<collection>.find({}, {‘name’: 1})

[returns the documents in below format:]

{

\_id: ObjectId(‘randomobjectid’),

name: “Harshit Bhawsar”

}

* **0 (exclude):**  This indicates to exclude the field in the data retrieval.
* db.<collection>.find({}, {‘name’: 0})

[returns the documents in below format name will not be fetched:]

{

\_id: ObjectId(‘randomobjectid’),

“address” : “somerandomaddress”,

“otherfield1” : val1,

...

}

**Projection Operators (specifically for array) :**

1. **$** :  This operators is used to return only the element of the array matched by our query. Basically this projection depends upon our query.

* db.<collection>.find({ hobbies: ’drama’ }, { “hobbies.$”: 1 })

[return the hobbies field in the document as [‘drama’] ]

1. **$elemMatch :**  This is utilized to counter the **$** operator return all the other elements of the array except the matching element.

* db.<collection>.find({ hobbies: ’drama’ }, { “hobbies”: {$elemMatch: { $ne: ‘drama’}} })

[return the hobbies field in the document as [‘action’, ‘adventure’,...] but never include drama]

1. **$slice:**  As per its name, it is a simple operator that is used to trim the array before returning the document. It can take two types of values
   1. **Single Value (length):**  It represent how many elements we want to return.

* db.<collection>.find({}, {‘hobbies’: {$slice: 2}})

[returns the documents with hobbies field have exactly 2 or less elements.]

* 1. **Array ([skip, length]):**  It represent how many elements to skip and how many elements to return after it.
* db.<collection>.find({}, {‘hobbies’: {$slice: [1,2]}})

[returns the documents with hobbies field have exactly 2 or less elements by skipping the first hobbies element.]