

MongoDB : Query Operators

Operators that can be used along with find() query



mongoDB®

Let's have some data



```
db.trainee.drop()
```

```
db.createCollection("trainee")

db.trainee.insertMany([
  {
    "admission_no": 3354,
    "first_name": "Spider",
    "last_name": "Man",
    "age": 13,
    "city": "Texas",
    "subjects" : [
      "android",
      "java"
    ],
    "semester": 2,
    "fee": 1000
  },
  {
    "admission_no": 3355,
    "first_name": "James",
    "last_name": "Bond",
    "age": 15,
    "city": "Alaska",
    "subjects" : [
      "dotnet",
      "mssql"
    ],
    "semester": 2,
    "fee": 2000
  },
  {
    "admission_no": 3356,
    "first_name": "Jack",
    "last_name": "Sparrow",
    "age": 14,
    "city": "California",
    "subjects" : [
      "Python",
      "machine learning"
    ],
    "semester": 2,
    "fee": 3000
  }
],
```


Let's have some data



```
{
  "admission_no": 3357,
  "first_name": "John",
  "last_name": "McClane",
  "age": 17,
  "city": "New York",
  "subjects" : [
    "Blockchain",
    "Solidity",
    "Ethereum"
  ],
  "semester": 3,
  "fee": 4000
},

{
  "admission_no": 3358,
  "first_name": "Optimus",
  "last_name": "Prime",
  "age": 18,
  "city": "Florida",
  "subjects" : [
    "android",
    "java"
  ],
  "semester": 2,
  "fee": 1000
},

{
  "admission_no": 3359,
  "first_name": "Captain",
  "last_name": "Kirk",
  "age": 15,
  "city": "Arizona",
  "subjects" : [
    "dotnet",
    "mssql"
  ],
  "semester": 2,
  "fee": 2000
},
```

Let's have some data



```
{
  {
    "admission_no": 3360,
    "first_name": "Harry",
    "last_name": "Potter",
    "age":14,
    "city": "New York",
    "subjects" : [
      "Magic",
      "Potions"
    ],
    "semester":1,
    "fee": 10000
  },
  {
    "admission_no": 3361,
    "first_name": "Rose",
    "last_name": "Dawson",
    "age":13,
    "city": "California",
    "subjects" : [
      "deep learning",
      "computer vision"
    ],
    "semester":4,
    "fee": 5000
  }
});
```


Equality Operator



Similar to the WHERE name = 'Abhi' in RDBMS

The syntax is : {<key>:{<value>}}

Example:

```
db.trainee.find({"subjects":"android"}).pretty()
```

-- The pretty() Method is used to display the results in a formatted way

Not Equals Operator



Similar to the WHERE fee != 5000 in RDBMS

The syntax is :

```
{<key>:{$ne:<value>}}
```

Example:

```
db.trainee.find({"fee":{$ne:5000}})
```


Less than & Less than or Equals Operator



Similar to the WHERE fee < 5000 or fee <= 5000 in RDBMS

The syntax is :

```
{<key>:{$lt:<value>}}
```

```
{<key>:{$lte:<value>}}
```

Example:

```
db.trainee.find({"fee":{$lt:5000}})
```

```
db.trainee.find({"fee":{$lte:5000}})
```

Greater than & greater than or Equals Operator



Similar to the WHERE fee > 5000 or fee >= 5000 in RDBMS

The syntax is :

```
{<key>:{$gt:<value>}}
```

```
{<key>:{$gte:<value>}}
```

Example:

```
db.trainee.find({"fee":{$gt:5000}})
```

```
db.trainee.find({"fee":{$gte:5000}})
```


Not Operator



Similar to the WHERE fee > 5000 or fee <= 5000 in RDBMS

The syntax is :

```
{<key>:{$not:{$gt:<value>}}}
```

```
{<key>:{$not:{$gte:<value>}}}
```

Example:

```
db.trainee.find({"fee":{$not:{$gt:5000}}})
```

```
db.trainee.find({"fee":{$not:{$gte:5000}}})
```

Values In Array Operator



The syntax is :

```
{<key>:{$in:[<value1>, <value2>,.....<valueN>]}}
```

Example:

```
db.trainee.find({"first_name":{$in:  
["Spider","James"]}})
```


Values Not In Array Operator



The syntax is :

```
{<key>:{$nin:[<value1>, <value2>,.....<valueN>]}}
```

Example:

```
db.trainee.find({"first_name":{$nin:
["Spider","James"]}))
```

AND Operator



The syntax is :

```
{ $and: [{ <key1>: <value1> }, { <key2>: <value2> } ] }
```

Example:

```
db.trainee.find({ $and: [ { "first_name": "Spider" },  
  { "last_name": "Man" } ] })
```


OR Operator



The syntax is :

```
{ $or: [{<key1>:<value1>}, {<key2>:<value2>} ] }
```

Example:

```
db.trainee.find({ $or: [ {"first_name": "Spider"},  
{"first_name": "James"} ] })
```

NOR Operator



The syntax is :

```
{ $nor: [{<key1>:<value1>}, {<key2>:<value2>} ] }
```

Example:

```
db.trainee.find({ $nor: [ {"first_name": "Spider"},  
{"first_name": "James"} ] })
```


NOR Operator



The syntax is :

```
{ $nor: [{<key1>:<value1>}, {<key2>:<value2>} ] }
```

Example:

```
db.trainee.find({ $nor: [ {"first_name": "Spider"},  
{"first_name": "James"} ] })
```

MongoDB : Project, Limit and Sort

Project, Limit and Sort the Documents to Display



mongoDB®

Projection



- Projection is the process of selecting only the necessary data from whole of the data of a particular document.
- If a document has 10 fields and you need to show only 5, then select only 5 fields from them
- Projection is done by passing fields as argument into the find()

Projection Example



The syntax is :

```
db.COLLECTION_NAME.find({}, {KEY:1})
```

`1` for showing the field

`0` for not showing the field

Example:

```
db.trainee.find({}, {"first_name":1, "fee":1})
```

To avoid id:

```
db.trainee.find({}, {"first_name":1, "fee":1,  
"_id":0})
```


Limiting query results



We can limit the number of documents returned back by the query using `limit()` method

The syntax is :

```
db.COLLECTION_NAME.find().limit(NUMBER)
```

Example:

```
db.trainee.find().limit(3)
```

Skipping query results



We can skip a particular document index returned by using the query using skip() method

The syntax is :

```
db.COLLECTION_NAME.find().skip(NUMBER)
```

Example:

```
db.trainee.find().limit(3).skip(1)
```


Sorting query results based on key



We can sort the returned set of documents based on a key using sort method. The parameter 1 to sort in Ascending order and -1 in Descending order.

The syntax is :

```
db.COLLECTION_NAME.find().sort({key:1/-1})
```

Example:

```
db.trainee.find().skip(1).sort({"fee":-1})
```

```
db.trainee.find().sort({"fee":-1,"first_name":1})
```

MongoDB : Indexing

Manually Index fields to make search easier



mongoDB®

Indexing in MongoDB



- Indexes are special data structures, which store a small amount of the data set in an easy-to-navigate form.
- We can create index of a specific field or set of fields ordered by the value of the field as specified in the index.
- Mongodb provides a default index named `_id` which acts as a primary key to access any document in a collection.
- We can manually create index for other fields so that during search, the db engine have to go through this index rather than the whole db.

Creating an Index



To create an index in MongoDB:

The syntax is :

```
db.collection_name.createIndex({field : value })
```

Example:

```
db.trainee.createIndex({"first_name":-1})
```

1 is for ascending order and -1 for descending.

View the Created Indexes using getIndexes()



To view the indexes:

The syntax is :

```
db.collection_name.getIndexes()
```

Example:

```
db.trainee.getIndexes()
```

View the Created Indexes



training.trainee

DOCUMENTS 8 TOTAL SIZE 1.5KB AVG. SIZE 183B INDEXES 3 TOTAL SIZE 61.4KB AVG. SIZE 20.5KB

Documents Aggregations Schema Explain Plan **Indexes** Validation

CREATE INDEX

Name and Definition ^	Type	Size	Usage	Properties
<div>_id_</div> <div>_id_ ↕</div>	REGULAR ⓘ	20.5 KB	2 since Wed Dec 22 2021	UNIQUE ⓘ
<div>first_name_-1</div> <div>first_name ↕</div>	REGULAR ⓘ	20.5 KB	1 since Wed Dec 22 2021	
<div>first_name_-1_last_name_-1</div> <div>first_name ↕ last_name ↕</div>	REGULAR ⓘ	20.5 KB	0 since Wed Dec 22 2021	COMPOUND ⓘ

Verify and Use a Created Index Example



Using the `explain()` , we are using index to explain the field

The syntax is :

```
db.collection_name.find({field:value}).explain()
```

Examples:

```
db.trainee.find({"first_name":"James"}).explain()
```

Drop Created Index



Using the dropIndex() method

The syntax is :

```
db.collection_name.dropIndex("index name")
```

Example:

```
db.trainee.dropIndex("first_name_-1")
```


Creating a Text Index



The text indexes to search inside string content. This will be very useful in search when the field contains paragraphs of text

The syntax is :

```
db.collection_name.createIndex({field : "text" })
```

Example:

```
db.trainee.createIndex({"first_name":"text"})
```

View the Created Indexes using getIndexes()



To view the indexes:

The syntax is :

```
db.collection_name.getIndexes()
```

Example:

```
db.trainee.getIndexes()
```


Search Using Text Index



The syntax is :

```
db.collection_name.find({$text:{$search:""}})
```

Example:

```
db.trainee.find({$text:{$search:"James"}})
```

Maintaining Atomicity in the Database



- We should maintain atomicity in MongoDB by compiling all the related information in a single document, which will update consistently.
- We can create such type of consistency via embedded documents.
- The embedded is for ensuring that every single update that takes place in the document is atomic.



Atomicity Example using a sample collection.



```
db.products.drop()
db.createCollection("products")
db.products.insert(
    {
        "product_name": "iphone 12",
        "category": "mobiles",
        "product_total": 10,
        "product_available": 7,
        "product_bought_by": [
            {
                "customer": "tom",
                "date": "17-Dec-2021"
            },
            {
                "customer": "jerry",
                "date": "18-Dec-2021"
            }
        ]
    })
```

Querying by maintaining atomicity of the db



- When a new customer buys the product:
- Step1 : check if the product is still available in product_available field.

- Step 2: If available, decrement the value of product_available field.

- Step 3: Insert the new customer details in the product_bought_by field.

Querying by maintaining atomicity of the db



```
db.products.findAndModify({  
  query:{product_available:{$gt:0}},  
  update:{  
    $inc:{product_available:-1},  
    $push:{product_bought_by:{customer:"Tim",  
      date:"19-Dec-2021"}}  
  }  
})
```

\$inc operator is used for incrementing value
\$push operator appends value to an array.