Comparision Operators

MongoDB comparison operators can be used to compare values in a document. The following table contains the common comparison operators.

Operator	Description
$eq	Matches values that are equal to the given value. **Syntax: { field: { $eq: value } }** db.people.find({age:{$eq:20}})
$gt	Matches if values are greater than the given value. **Syntax: { field: { $gt: value } }** db.people.find({age:{$gt:20}})
$lt	Matches if values are less than the given value. **Syntax: { field: { $lt: value } }** db.people.find({age:{$lt:20}})
$gte	Matches if values are greater or equal to the given value. **Syntax: { field: { $gte: value } }** db.people.find({age:{$gte:20}})
$lte	Matches if values are less or equal to the given value. **Syntax: { field: { $lte: value } }** db.people.find({age:{$lte:20}})
$in	Matches any of the values in an array. **Syntax: { field: { $nin: [<value1>, <value2>, ... <valueN>] } }** db.inventory.find({ qty: { $in: [5, 15] } })
$ne	Matches values that are not equal to the given value. **Syntax: { field: { $ne: value } }** db.people.find({age:{$ne:20}})
$nin	Matches none of the values specified in an array. **Syntax: { field: { $nin: [<value1>, <value2>, ... <valueN>] } }** db.inventory.find({ qty: { $nin: [5, 15] } })

- ✓ **Update only one document with branch “CSE” and age “21” where age is equal to 5.**
  - db.people.updateOne({age:{$eq:5}},{$set:{branch:"CSE",age: 21}})

- ✓ To display all documents where age is greater than 25
  - db.people.find({age:{$gt:25}})
- ✓ To display all documents where age is greater than 25 and less than 50
  - db.people.find({age:{$gt:25,$lt:50}})
  - db.people.find({$and:[{age:{$gt:25}},{age:{$lt:50}}]})
- ✓ To display exact match of the documents.
  - db.people.find({age:{$eq:70}})
- ✓ To display all documents other than age is 70
  - db.people.find({age:{$ne:70}})
- ✓ To display Matches any of the values in an array.
  - db.people.find({age:{$in:[45,70]}})
- ✓ To display other than Matches any of the values in an array.
  - db.people.find({age:{$nin:[45,70]}})

Logical operators

MongoDB supports logical query operators. These operators are used for filtering the data and getting precise results based on the given conditions. The following table contains the comparison query operators:

Operator	Description
$and	It is used to join query clauses with a logical AND and return all documents that match the given conditions of both clauses.
$or	It is used to join query clauses with a logical OR and return all documents that match the given conditions of either clause.
$not	It is used to invert the effect of the query expressions and return documents that does not match the query expression.
$nor	It is used to join query clauses with a logical NOR and return all documents that fail to match both clauses.

- ✓ **$and**
  - **To fetch documents with more than one conditions and all the conditions must be satisfied.**
    - db.student.find({$and:[{name:"N1"},{age:28}]})
      or
      db.student.find({name:"N1",age:28})

This fetches documents which satisfies both the conditions.

- ✓ **$or**
  - **To fetch documents with more than one conditions and one of the conditions must be satisfied.**
    - db.student.find({$or:[{name:"N1"},{age:28}]})

This fetches documents which satisfies one of the conditions.

- ✓ **$not**
  - It performs a logical NOT operation on the specified <operator-expression> and selects the documents that do not match the <operator-expression>. This includes documents that do not contain the field.

Syntax: { field: { $not: { <operator-expression> } } }

db.people.find({ age: { $not: { $lt: 20 } } })

- ✓ **$nor**
  - The $nor is a logical query operator that allows the user to perform a logical NOR operation on an array of one or more query expressions. This operator is also used to select or retrieve documents that do not match all of the given expressions in the array. The user can use this operator in methods like find(), update(), etc., as per their requirements.
    - **db.people.find({$nor: [{name: "N1"}]})**
    - **db.people.find({$nor: [{name: "N1"},{age:20}]})**
      - **It will show documents which do not have name "N1" or age "20"**
      - **To display only name field.**
        - **db.people.find({ age: { $in: [20, 21] } },{name:1,_id:0})**

Field Update operators

Name	Description
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp.
$inc	Increments the value of the field by the specified amount.
$mul	Multiplies the value of the field by the specified amount.
$rename	Renames a field.
$set	Sets the value of a field in a document.
$unset	Removes the specified field from a document.

- ✓ **$inc**
  - **{ $inc: { <field1>: <amount1>, <field2>: <amount2>, ... } }**

    Examples:
  - db.table1.updateMany({},{$inc:{age:10}});
    - Increase age field values by 10 for all documents
  - db.table1.updateOne({},{$inc:{age:15}});
    - Increase age field values by 15 of 1st document
  - db.table1.updateOne({},{$inc:{age:-15}});
    - increase age field values by (-15) of 1st document

- ✓ **$mul**

  Multiply the value of a field by a number. To specify a $mul expression, use the following prototype:

  { $mul: { <field1>: <number1>, ... } }

  The field to update must contain a numeric value.

  - db.table1.updateOne({},{$mul:{age:15}});
    - Multiply age field by 15 of 1st document
  - db.table1.updateMany({name:'N1'},{$mul:{age:0.5}});
    - Multiply age field by 0.5 for all documents where name is "N1"
  - db.table1.updateOne({},{$mul:{age:2}});
    - Multiply age field by 0.5 for all documents

- ✓ **$unset**

  The $unset operator deletes a particular field. Consider the following.

  syntax:{ $unset: { <field1>: "", ... } }

  Example:

  db.people.updateOne({age:{$eq:21}},{$unset:{branch:"CSE",age: 21}})

- ✓ **$rename**

  The $rename operator updates the name of a field and has the following form:

  Syntax: {$rename: { <field1>: <newName1>, <field2>: <newName2>, ... } }

  db.people.updateMany({},{$rename:{'name':'uname'}})
  db.people.updateMany({},{$rename:{'name':'Uname','branch':'Branch'}})

MongoDB Cursor

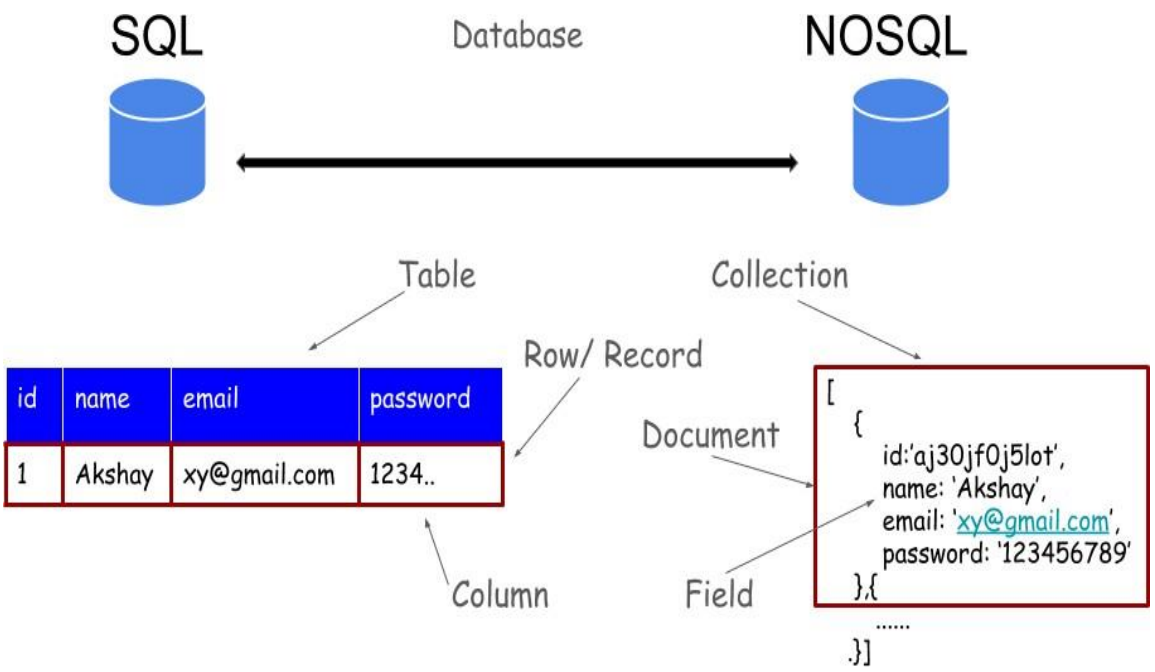
When find() method is used to find a document present in a given collection, then this method returns a pointer which will point to a document of a collection, this pointer is known as cursor.

Using this cursor(pointer) also, we can access the document. By default, cursor iterate automatically, but we can also iterate it manually.

For Example:

```
> let rec=db.lju.find({age:30})
> rec
```

SQL to MongoDB Mapping



SQL	MongoDB
CREATE TABLE lju (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(20),age Number, status char(1), PRIMARY KEY (id))	**db.createCollection ("lju")** creates collection no need to define schema
ALTER TABLE lju ADD join_dateDATETIME	db.lju.updateMany({ }, { $set: { join_date: new Date() } }) **Note:** **new Date().toLocaleDateString()-The toLocaleDateString() method returns the date (not the time) of a date object as a string, using locale conventions.**
ALTER TABLE lju DROP COLUMN join_date	db.lju.updateMany({ }, { $unset: { "join_date": "" } })

DROP TABLE lju	db.lju.drop()
INSERT INTO lju (name, age, status) VALUES (“Test”, 45, "A")	db.lju.insertOne({ name: “Test”, age: 18, status: "A" })
SELECT *FROM lju	db.lju.find()
SELECT id, name FROM lju	db.lju.find({ }, { name:1 })
SELECT name, status FROM lju	db.lju.find({ }, { name:1,status:1,_id: 0 })
SELECT * FROM lju WHERE status ="A"	db.lju.find({ status: "A" })
SELECT name FROM lju WHERE status = "A"	db.lju.find({ status: "A" }, { name: 1,_id: 0 })
SELECT * FROM lju WHERE status != "A"	db.lju.find({ status: { $ne: "A" } })
SELECT * FROM lju WHERE status = "A" AND age = 50	db.lju.find({status:”A”,age:50}) **or** db.lju.find({ $and: [{ status: "A" } , { age: 50 }] })
SELECT * FROM lju WHERE status = "A" OR age = 30	db.lju.find({ $or: [{ status: "A" } , { age: 30 }] })
SELECT * FROM lju WHERE age > 25	db.lju.find({ age: { $gt: 25 } })
SELECT * FROM lju WHERE age <= 25	db.lju.find({ age: { $lte: 25 } })
SELECT * FROM lju WHERE age > 25 AND age<= 50	db.lju.find({ age: { $gt: 25, $lte: 50 } })
SELECT * FROM lju WHERE name like "%bc%"	db.lju.find({ name: {$regex: /bc/ }}) **or** db.lju.find({ name: { $regex: /bc/ } })
SELECT * FROM lju WHERE name like "bc%"	db.lju.find({ name: /^bc/ }) **or** db.lju.find({ name: { $regex: /^bc/ } })
SELECT * from lju wherename=”abc” and age=20	db.lju.find({name:"abc",age:20})
SELECT * FROM lju WHERE status = "A" ORDER BY age ASC	db.lju.find({ status: "A" }).sort({ age: 1 })

SELECT * FROM lju WHERE status = "A" ORDER BY age DESC	db.lju.find({ status: "A" }). sort({ age: -1 })
SELECT COUNT(*) FROM lju	db.lju. count() or db. lju. find(). count()
SELECT COUNT(*)FROM lju WHERE age > 30	db.lju.count({ age: { $gt: 30 } }) or db.lju.find({ age: { $gt: 30 } }).count()
SELECT * FROM lju LIMIT 1	db.lju.findOne() or db.lju.find().limit(1)
SELECT * FROM lju LIMIT 2 SKIP 3	db.lju.find().limit(2).skip(3)
EXPLAIN SELECT * FROM lju WHERE status = "A"	db.lju. find({ status: "A" }).explain()
UPDATE lju SET status = "C" WHERE age > 25	db.lju.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" } })
UPDATE lju SET age = age + 3 WHERE status = "A"	db.lju.updateMany({ status: "A" } , { $inc: { age: 3 }
DELETE FROM lju WHERE status = "D"	db.lju.deleteMany({ status: "D" })
DELETE FROM lju	db.lju.deleteMany({ })
SELECT * FROM lju WHERE name NOT IN ('aaa','abc','bbb');	db.lju.find({ name: { $nin: [‘aaa’,’abc’,’bbb’] } })
SELECT name FROM lju WHERE age IN (20,23,33);	db.lju.find({ age: { $in: [20, 23, 33] } },{name:1,_id:0})

Regex

$regex : Provides regular expression capabilities for pattern matching strings in queries.

To use $regex, use one of the following syntax:

{ <field>: { $regex: /pattern/} }

case 1: Find all names where "test" occurs as a substring or as a separate word.

```
db.lju.find({name:{$regex:/test/}})
```

or

```
db.lju.find({name:{$regex:"test"}
})
```

case 2: To have a case-insensitive matching we can append /i indentifier after RE

```
db.lju.find({name:{$regex:/test/i}})
```

This will return documents if name is stored in collection as "Test","tEST", "teST","Testing" etc.

case 3: To match a string beginning with "test" only.

```
db.lju.find({name:{$regex:/^test/}})
```

case 4: To end with "test" only.

```
db.lju.find({name:{$regex:/test$/}})
```

case 5: To end with digit only.

```
db.lju.find({name:{$regex:/[0-9]$/}})
```

case 6: To start with digit only.

```
db.lju.find({name:{$regex:/^[0-9]/}})
```

or

```
db.lju.find({name:{$regex:/^\d
/}})
```

case 7: To accept only digits, nothing else. Not even blank.

```
db.lju.find({name:{$regex:/^[0-9]+$/}})
```

case 8: To accept only with empty string also.

```
db.lju.find({name:{$regex:/^[0-9]*$/}})
```

case 9: To match having a name of 3-10 letters only.

db.lju.find({name:{$regex:/^[A-Za-z]{3,10}$/}})

Letters only is mentioned in question so we have added ^ and $.

Note:

If we include \w instead of this [A-Za-z], then it may allow digits & underscore also.

If it is asked in question that it **allowed only letters** then **use [A-Za-z]**

Note: all patterns can be matched with string only. If there is one field age and we haveinserted all int values then RegEx can not be compared with it.

Examples

Example1

Create a collection named student having fields name, age, standard, percentage. Insert 5 to 10 random documnets in collection.

1) **Find name of all students having age>5**
2) **Increase the standard for all students by 1.**
3) **Arrange all the records in descending order of age.**
4) **Show the name of student who is the oldest student among all students.**
5) **Delete the record of the student if standard is 12.**

```
db.student.insertMany([{name:"abc",age:13,standard:6,perc:80},{name:"def",age:15,standard:8,perc:90},{name:"ghi",age:10,standard:3,perc:75},{name:"pqr",age:5,standard:1,perc:89},{name:"xyz",age:17,standard:12,perc:97}])
```

```
1) db.student.find({age:{$gt:5}},{name:1,_id:0})
2) db.student.updateMany({},{$inc:{standard:1}})
3) db.student.find().sort({age:-1})
4) db.student.find({},{name:1,_id:0}).sort({age:-1}).limit(1)
5) db.student.deleteOne({standard:12})
```

Example2

Perform the tasks as asked below.

Create Collection “employees” with following data

```
[{_id: 1,name: "Eric",age: 30,position: "Full Stack Developer",salary: 60000},

{_id: 2,name: "Erica",age: 35,position: "Intern",salary: 8000},
```

{_id: 3,name: "Erical",age: 40,position: "UX/UI Designer",salary: 56000},

{_id: 4,name: "treric7",age: 37,position: "Team Leader",salary: 85000},

{_id: 5,name: "Eliza",age: 25,position: "Software Developer",salary: 45000},

{_id: 6,name: "Trian",age: 29,position: "Data Scientist",salary: 75000},

{_id: 7,name: "Elizan",age: 25,position: "Full Stack Developer",salary: 49000}]

1) Find All Documents:
2) Find Documents by Position “Full Stack Developer”:
3) Retrieve name of employees whose age is greater than or equal to 25 and less than or equal to 40.
4) Retrieve name of the employee with the highest salary.
5) Retrieve employees with a salary greater than 50000.
6) Retrieve employees' names and positions, excluding the "_id" field.
7) Count the number of employees who have salary greater than 50000
8) Retrieve employees who are either " **Software Developer**" or "**Full Stack Developer**" and are below 30 years.
9) Increase the salary of an employee who has salary less than 50000 by 10%.
10) Delete all employees who are older than 50.
11) Give a 5% salary raise to all "**Data Scientist**"
12) Find documents where name like “%an”
13) Find documents where name like “Eri--” (Case Insensitive)
14) Find documents where name like “%ric%”
15) Find documents where name contains only 4 or 5 letters.
16) Find documents where name must end with digit

Answers:

1) db.employees.insertMany([{_id: 1,name: "Eric",age: 30,position: "Full Stack Developer",salary: 60000},
   {_id: 2,name: "Erica",age: 35,position: "Intern",salary: 8000},
   {_id: 3,name: "Erical",age: 40,position: "UX/UI Designer",salary: 56000},
   {_id: 4,name: "treric7",age: 37,position: "Team Leader",salary: 85000},
   {_id: 5,name: "Eliza",age: 25,position: "Software Developer",salary: 45000},
   {_id: 6,name: "Trian",age: 29,position: "Data Scientist",salary: 75000},
   {_id: 7,name: "Elizan",age: 25,position: "Full Stack Developer",salary: 49000}])
2) db.employees.find()
3) db.employees.find({position:"Full Stack Developer"})
4) db.employees.find({ age: { $gte: 30, $lte: 40 } },{name:1,_id:0})
5) db.employees.find({},{name:1,_id:0}).sort({ salary: -1 }).limit(1)

6) db.employees.find({ salary: { $gt: 50000 } })
7) db.employees.find({salary:{$gt:50000}}).count()
8) db.employees.find({ $and: [{ $or: [{ position: "Software Developer" }, { position: "Full Stack Developer" }] }, { age: { $lt: 30 } }] })
9) db.employees.updateOne({salary:{$lt:50000}},{ $mul: { salary: 1.1 } })
10) db.employees.deleteMany({ age: { $gt: 50 } })
11) db.employees.updateMany({ position: "Data Scientist" }, { $mul: { salary: 1.05 } })
12) db.employees.find({name:{$regex:/an$/}})
13) db.employees.find({name:{$regex:/^eri[A-z]{2}$/i}})
14) db.employees.find({name:{$regex:/ric/i}})
15) db.employees.find({name:{$regex:/^[A-Za-z]{4,5}$/i}})
16) db.employees.find({name:{$regex:/[0-9]$/}})

Example3

Insert 10 documents with random data with fields _id,brand,price,cat as shown below.

```
db.product.insertMany([
{_id:1,brand:"samsung",price:29000,cat:"mobile"},
{_id:2,brand:"nokia",price:5000,cat:"mobile"},
{_id:3,brand:"vivo",price:16000,cat:"mobile"},
{_id:4,brand:"samsung",price:60000,cat:"tv"},
{_id:5,brand:"samsung",price:40000,cat:"washing machine"},
{_id:6,brand:"ifb",price:45000,cat:"wasing machine"},
{_id:7,brand:"apple",price:120000,cat:"mobile"},
{_id:8,brand:"oppo",price:20000,cat:"mobile"},
{_id:9,brand:"sony",price:80000,cat:"tv"},
{_id:10,brand:"vivo",price:31000,cat:"mobile"},
])
```

1) **Display price and brand of product which are of mobile cat.**
2) **Increase price of each Samsung products by 1000.**
3) **Update all vivo product by adding field quantity and add random value**
4) **Display price of products which are of vivo or oppo brand.**
5) **Display brand and cat of products which are less than 80000 and greater than or equal to 30000.**

Answers:

1) db.product.find({cat:”mobile”},{cat:0,_id:0})
2) db.product.updateMany({brand:"samsung"},{$inc:{price:1000}})

3) db.product.updateMany({brand:"vivo"},{$set:{quantity:5}})
4) db.product.find({$or:[{brand:"vivo"},{brand:"oppo"}]},{price:1,_id:0})
5) db.product.find({price:{$lt:80000,$gte:30000}},{price:0,_id:0})

Example4

Consider following student collection:

```
[
 {_id:123433,name: "SSS",age:22},
 {_id:123434,name: "YYY",age:2},
 {_id:123435,name: "PPP",age:32},
]
```

Do as directed:

(1) Update name=”JJJ” and age=40, where age=20 occurs. Insert new document, if record is not found.

(2) To retrieve age and name fields of documents having names “YYY” & “SSS”. Don’t project _id field.

Answers:

1) db.info.updateMany({age:20},{$set:{name:'JJJ',age:40}},{upsert:true})
2) db.info.find({$or:[{name:'YYY'},{name:'SSS'}]},{_id:0})