23 COMMON MONGODB QUERY OPERATORS & HOW TO USE THEM

In this article, we will take a look at the most commonly used MongoDB query operators. We'll explain what they do, then share examples so you can see how they work.

Learn more about how projection operators can optimize query performance:

(This article is part of our MongoDB Guide. Use the right-hand menu to navigate.)

About MongoDB query operators

What are operators in MongoDB?

MongoDB operators are special symbols or keywords that can be used to interact with the database. They direct a compiler or an interpreter to carry out mathematical or logical operations.

The query operators enhance the functionality of MongoDB by allowing developers to create complex queries to interact with data sets that match their applications.

Getting started with AIOps is easy. Manage escalating IT complexity with artificial intelligence

[for enterprise operation ›]

What are the different types of query operators in MongoDB?

MongoDB offers the following query operator types:

- Comparison operators
- Logical operators
- Element operators
- Evaluation operators
- Geospatial operators
- Array operators
- Bitwise operators
- Comments operators

How can you use query operators in MongoDB?

MongoDB operators can be used with any supported MongoDB command. Used correctly, Mongo Shell basic commands can reduce query load by controlling the included and excluded fields. The result is improved performance.

Examples of MongoDB query operators

Now, let's look at commonly used MongoDB query operators. (We won't touch on them all, there are so many.) We'll use the following dataset with the **find()** function to demonstrate each operator's functionality.

Database: supermarket

Collections: employees, inventory, payments, promo

```
use supermarket
db.employees.find()
db.inventory.find()
db.payments.find()
db.promo.find()
```

Dataset:

```
> use supermarket
switched to db supermarket
> db.employees.find()
{ "_id" : 312456, "emp_name" : "Barry Stevens", "emp_age" : 28, "job_role" : "Store Manager", "sala
ry" : 120000 }
{ "_id" : 345342, "emp_name" : "Martin Garrix", "emp_age" : 25, "job_role" : "Store Associate", "sa
lary" : 45000 }
{ "_id" : 334566, "emp_name" : "Linda Harris", "emp_age" : 35, "job_role" : "Cashier", "salary" : 6
7500 }
{ "_id" : 245345, "emp_name" : "Maggie Smith", "emp_age" : 40, "job_role" : "Senior Cashier", "sala
ry" : 72500 }
{ "_id" : 445634, "emp_name" : "Lucy Hale", "emp_age" : 22, "job_role" : "Store Associate", "salary
" : 35000 }
>
> db.inventory.find()
{ "_id" : "LS0000123", "name" : "XYZ Chocolate Bar - 100g", "price" : 5.23, "quantity" : 25000, "ca
tegory" : [ "chocolate", "sweets" ] }
{ "_id" : "LS0003123", "name" : "Milk Non-Fat - 1lt", "price" : 3, "quantity" : 1000, "category" :
[ "dairy", "healthy" ] }
{ "_id" : "LS0004566", "name" : "Eggs - 12 Pack", "price" : 6, "quantity" : 5000, "category" : [ "p
oultry", "generic" ] }
{ "_id" : "LS0008542", "name" : "Whole Chicken", "price" : 12.59, "quantity" : 1250, "category" : [
 "poultry", "meat" ] }
{ "_id" : "LS0009845", "name" : "Carrots (Packed) - 250g", "price" : 3.59, "quantity" : 3000, "cate
gory" : [ "vegetables", "healthy", "organic" ] }
{ "_id" : "LS0009846", "name" : "Beans (Packed) - 250g", "price" : 6.75, "quantity" : 6000, "catego
ry" : [ "vegetables", "healthy", "organic" ] }
{ "_id" : "LS0009100", "name" : "Bell Pepper (Packed) - 250g", "price" : 4.95, "quantity" : 12000,
"category" : [ "vegetables", "healthy", "organic" ] }
{ "_id" : "LS0002688", "name" : "ZZ Butter - 500g", "price" : 25, "quantity" : 500, "category" : [
"dairy", "healthy", "premium" ] }
>
> db.payments.find()
{ "_id" : "BL2021005", "gross_amount" : 105.65, "discounts" : 10, "net_amount" : 95.65, "date_time"
 : ISODate("2021-01-01T16:00:00Z") }
{ "_id" : "BL2021006", "gross_amount" : 45.25, "discounts" : 0, "net_amount" : 45.25, "date_time" :
 ISODate("2021-01-01T16:15:55Z") }
{ "_id" : "BL2021007", "gross_amount" : 153.33, "discounts" : 20.33, "net_amount" : 133, "date_time
" : ISODate("2021-01-01T16:31:08Z") }
{ "_id" : "BL2021008", "gross_amount" : 21, "discounts" : 0, "net_amount" : 21, "date_time" : ISODa
te("2021-01-01T20:25:52Z") }
{ "_id" : "BL2021009", "gross_amount" : 89.72, "discounts" : 0.72, "net_amount" : 89, "date_time" :
 ISODate("2021-01-02T08:45:12Z") }
{ "_id" : "BL2021010", "gross_amount" : 33.5, "discounts" : 20.5, "net_amount" : 13, "date_time" :
ISODate("2021-01-02T11:02:35Z") }
>
> db.promo.find()
{ "_id" : "PROMO01", "name" : "Sales Promo", "period" : 7, "daily_sales" : [ 20, 50, 12, 30, 45, 15
, 60 ] }
{ "_id" : "PROMO02", "name" : "Milk Promo", "period" : 2, "daily_sales" : [ 120, 200 ] }
{ "_id" : "PROMO03", "name" : "Meat Promo", "period" : 3, "daily_sales" : [ 101, 250 ] }
{ "_id" : "PROMO04", "name" : "New Year Promo", "period" : 7, "daily_sales" : [ 65, 88, 105, 188, 7
4, 278, 350 ] }
>
```

Comparison operators in MongoDB

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.
$gt	Matches if values are greater than the given value.
$lt	Matches if values are less than the given value.
$gte	Matches if values are greater or equal to the given value.
$lte	Matches if values are less or equal to the given value.
$in	Matches any of the values in an array.
$ne	Matches values that are not equal to the given value.
$nin	Matches none of the values specified in an array.

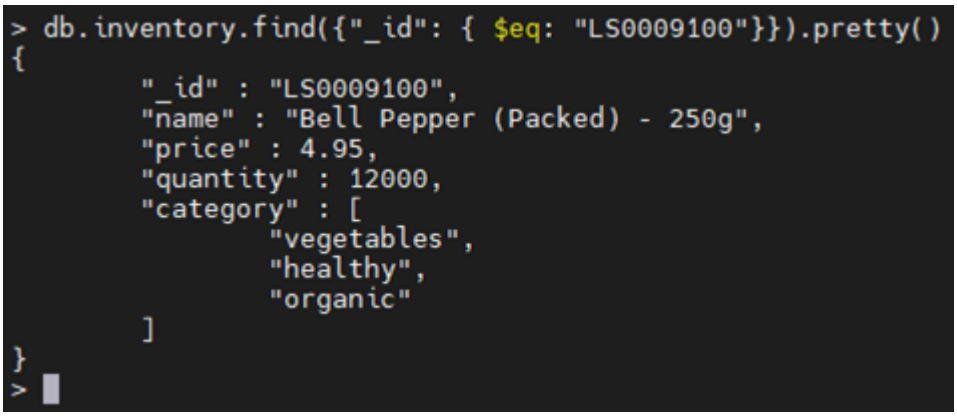
$eq Operator

The $eq operator adds clarity to the query, to match documents where a field's value is specific — your results will match a specified value.

In this $eq operator example, we retrieve the document with the exact _id value "LS0009100".

```
db.inventory.find({"_id": { $eq: "LS0009100"}}).pretty()
```

Result:



$gt and $lt Operators

These two related operators return results that are "greater than" and "less than" specified values.

In this $gt operator example, we retrieve the documents where the `quantity` is greater than 5000.

```
db.inventory.find({"quantity": { $gt: 5000}}).pretty()
```

Result:

```
> db.inventory.find({"quantity": { $gt: 5000}}).pretty()
{
        "_id" : "LS0000123",
        "name" : "XYZ Chocolate Bar - 100g",
        "price" : 5.23,
        "quantity" : 25000,
        "category" : [
                "chocolate",
                "sweets"
        ]
}
{
        "_id" : "LS0009846",
        "name" : "Beans (Packed) - 250g",
        "price" : 6.75,
        "quantity" : 6000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0009100",
        "name" : "Bell Pepper (Packed) - 250g",
        "price" : 4.95,
        "quantity" : 12000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
>
```

Let's find the documents with the 'quantity' less than 5000 by using the $lt operator.

```
db.inventory.find({"quantity": { $lt: 5000}}).pretty()
```

Result:

```
> db.inventory.find({"quantity": { $lt: 5000}}).pretty()
{
        "_id" : "LS0003123",
        "name" : "Milk Non-Fat - 1lt",
        "price" : 3,
        "quantity" : 1000,
        "category" : [
                "dairy",
                "healthy"
        ]
}
{
        "_id" : "LS0008542",
        "name" : "Whole Chicken",
        "price" : 12.59,
        "quantity" : 1250,
        "category" : [
                "poultry",
                "meat"
        ]
}
{
        "_id" : "LS0009845",
        "name" : "Carrots (Packed) - 250g",
        "price" : 3.59,
        "quantity" : 3000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0002688",
        "name" : "ZZ Butter - 500g",
        "price" : 25,
        "quantity" : 500,
        "category" : [
                "dairy",
                "healthy",
                "premium"
        ]
}
```

$gte and $lte Operators

These operators refine the two above by adding the parameter of “equal to,” so that you get results that include and are between two values.

Find documents with ‘quantity’ greater than or equal to 12,000 using the following $gte operator example.

```
db.inventory.find({"quantity": { $gte: 12000}}).pretty()
```

Result:

```
> db.inventory.find({"quantity": { $gte: 12000}}).pretty()
{
        "_id" : "LS0000123",
        "name" : "XYZ Chocolate Bar - 100g",
        "price" : 5.23,
        "quantity" : 25000,
        "category" : [
                "chocolate",
                "sweets"
        ]
}
{
        "_id" : "LS0009100",
        "name" : "Bell Pepper (Packed) - 250g",
        "price" : 4.95,
        "quantity" : 12000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
>
```

The following query returns documents where the quantity is less than or equal to 1000.

```
db.inventory.find({"quantity": { $lte: 1000}}).pretty()
```

Result:

```
> db.inventory.find({"quantity": { $lte: 1000}}).pretty()
{
        "_id" : "LS0003123",
        "name" : "Milk Non-Fat - 1lt",
        "price" : 3,
        "quantity" : 1000,
        "category" : [
                "dairy",
                "healthy"
        ]
}
{
        "_id" : "LS0002688",
        "name" : "ZZ Butter - 500g",
        "price" : 25,
        "quantity" : 500,
        "category" : [
                "dairy",
                "healthy",
                "premium"
        ]
}
>
```

$in and $nin Operators

The $in and $nin operators return results where the value is in, or not in, a set of values — that it matches or does not match a specified array.

The following query returns documents where the price field contains the given values.

```
db.inventory.find({"price": { $in: }}).pretty()
```

Result:

```
> db.inventory.find({"price": { $in: [3, 6]}}).pretty()
{
        "_id" : "LS0003123",
        "name" : "Milk Non-Fat - 1lt",
        "price" : 3,
        "quantity" : 1000,
        "category" : [
                "dairy",
                "healthy"
        ]
}
{
        "_id" : "LS0004566",
        "name" : "Eggs - 12 Pack",
        "price" : 6,
        "quantity" : 5000,
        "category" : [
                "poultry",
                "generic"
        ]
}
>
```

If you want to find documents where the price fields do not contain the given values, use the following query.

`db.inventory.find({"price": { $nin: }}).pretty()`

Result:

```
> db.inventory.find({"price": { $nin: [5.23, 3, 6, 3.59, 4.95]}}).pretty()
{
        "_id" : "LS0008542",
        "name" : "Whole Chicken",
        "price" : 12.59,
        "quantity" : 1250,
        "category" : [
                "poultry",
                "meat"
        ]
}
{
        "_id" : "LS0009846",
        "name" : "Beans (Packed) - 250g",
        "price" : 6.75,
        "quantity" : 6000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0002688",
        "name" : "ZZ Butter - 500g",
        "price" : 25,
        "quantity" : 500,
        "category" : [
                "dairy",
                "healthy",
                "premium"
        ]
}
>
```

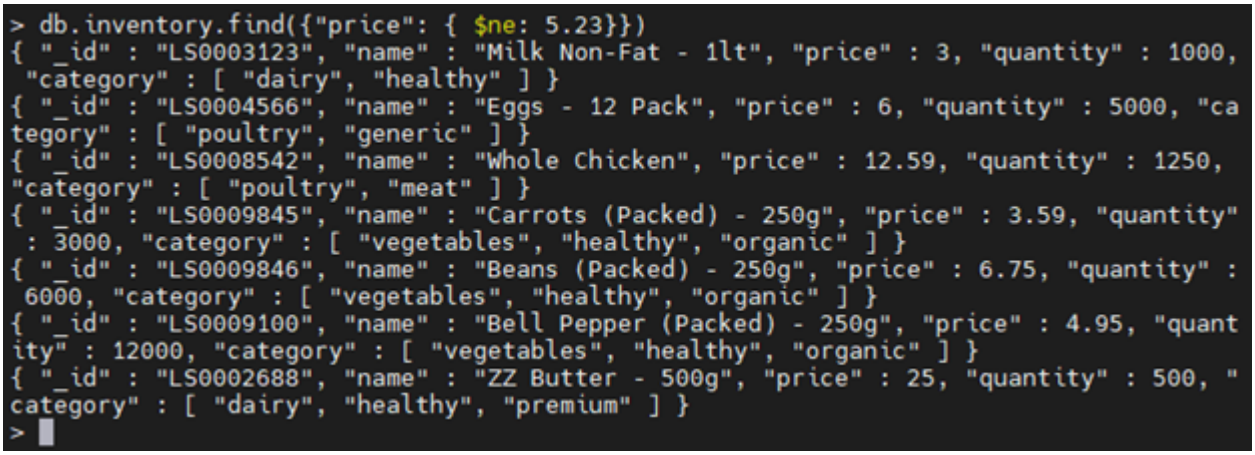
$ne Operator

The $ne operator returns results that match values that are not equal to the given value, which is helpful for excluding what is not relevant to your query.

Find documents where the value of the price field is not equal to 5.23 in the inventory collection.

```
db.inventory.find({"price": { $ne: 5.23}})
```

Result:



Logical operators in MongoDB

MongoDB logical operators can be used to filter data based on given conditions. These operators provide a way to combine multiple conditions. Each operator equates the given condition to a true or false value.

Here are the examples of MongoDB logical operators:

Operator	Description
$and	Joins two or more queries with a logical AND and returns the documents that match all the conditions.
$or	Join two or more queries with a logical OR and return the documents that match either query.
$nor	The opposite of the OR operator. The logical NOR operator will join two or more queries and return documents that do not match the given query conditions.
$not	Returns the documents that do not match the given query expression.

$and Operator

To get results that respond to two or more queries, join them with the $and operator.

Find documents that match both the following conditions

- job_role is equal to “Store Associate”
- emp_age is between 20 and 30

```
db.employees.find({ $and: }).pretty()
```

Result:

```
> db.employees.find({ $and: [{"job_role": "Store Associate"}, {"emp_age": {$gte: 20, $lte: 30}}]}).
pretty()
{
        "_id" : 345342,
        "emp_name" : "Martin Garrix",
        "emp_age" : 25,
        "job_role" : "Store Associate",
        "salary" : 45000
}
{
        "_id" : 445634,
        "emp_name" : "Lucy Hale",
        "emp_age" : 22,
        "job_role" : "Store Associate",
        "salary" : 35000
}
>
```

$or and $nor Operators

For results that match one query, or one of many queries, use the $or logical operator. For results that match none of the queries, use the $nor operator.

Find documents that match either of the following conditions.

- job_role is equal to “Senior Cashier” or “Store Manager”

```
db.employees.find({ $or: }).pretty()
```

Result:

```
> db.employees.find({ $or: [{"job_role": "Senior Cashier"}, {"job_role": "Store Manager"}]}).pretty
()
{
        "_id" : 312456,
        "emp_name" : "Barry Stevens",
        "emp_age" : 28,
        "job_role" : "Store Manager",
        "salary" : 120000
}
{
        "_id" : 245345,
        "emp_name" : "Maggie Smith",
        "emp_age" : 40,
        "job_role" : "Senior Cashier",
        "salary" : 72500
}
>
```

Find documents that do not match either of the following conditions.

- job_role is equal to “Senior Cashier” or “Store Manager”

```
db.employees.find({ $nor: }).pretty()
```

Result:

```
> db.employees.find({ $nor: [{"job_role": "Senior Cashier"}, {"job_role": "Store Manager"}]}).prett
y()
{
        "_id" : 345342,
        "emp_name" : "Martin Garrix",
        "emp_age" : 25,
        "job_role" : "Store Associate",
        "salary" : 45000
}
{
        "_id" : 334566,
        "emp_name" : "Linda Harris",
        "emp_age" : 35,
        "job_role" : "Cashier",
        "salary" : 67500
}
{
        "_id" : 445634,
        "emp_name" : "Lucy Hale",
        "emp_age" : 22,
        "job_role" : "Store Associate",
        "salary" : 35000
}
>
```

$not Operator

When you want to exclude specific results, you can specify the exclusion parameter with the $not operator.

Find documents where they do not match the given condition.

- emp_age is not greater than or equal to 40

```
db.employees.find({ "emp_age": { $not: { $gte: 40}}})
```

Result:

```
> db.employees.find({ "emp_age": { $not: { $gte: 40}}})
{ "_id" : 312456, "emp_name" : "Barry Stevens", "emp_age" : 28, "job_role" : "Store Manager", "sala
ry" : 120000 }
{ "_id" : 345342, "emp_name" : "Martin Garrix", "emp_age" : 25, "job_role" : "Store Associate", "sa
lary" : 45000 }
{ "_id" : 334566, "emp_name" : "Linda Harris", "emp_age" : 35, "job_role" : "Cashier", "salary" : 6
7500 }
{ "_id" : 445634, "emp_name" : "Lucy Hale", "emp_age" : 22, "job_role" : "Store Associate", "salary
" : 35000 }
>
```

Element operators in MongoDB

The element MongoDB query operators are used to identify documents using the fields of the document. The table given below lists the current element operators.

Operator	Description
$exists	Matches documents that have the specified field.
$type	Matches documents according to the specified field type. These field types are specified BSON types and can be defined either by type number or alias.

$exists Operator

Use the $exists operator to filter for documents that have a specific field.

Find documents where the job_role field exists and equal to “Cashier”.

```
db.employees.find({ "emp_age": { $exists: true, $gte: 30}}).pretty()
```

Result:

```
> db.employees.find({ "emp_age": { $exists: true, $gte: 30}}).pretty()
{
        "_id" : 334566,
        "emp_name" : "Linda Harris",
        "emp_age" : 35,
        "job_role" : "Cashier",
        "salary" : 67500
}
{
        "_id" : 245345,
        "emp_name" : "Maggie Smith",
        "emp_age" : 40,
        "job_role" : "Senior Cashier",
        "salary" : 72500
}
>
```

Find documents with an address field. (As the current dataset does not contain an address field, the output will be null.)

```
db.employees.find({ "address": { $exists: true}}).pretty()
```

Result:

```
> db.employees.find({ "address": { $exists: true}}).pretty()
>
```

$type Operator

When database objects are noted for type in a BSON notation, use the $type operator to filter out results according to a specific, desired type of object.

The following query returns documents if the emp_age field is a double type. If we specify a different data type, no documents will be returned even though the field exists as it does not correspond to the correct field type.

```
db.employees.find({ "emp_age": { $type: "double"}})
```

Result:

```
> db.employees.find({ "emp_age": { $type: "double"}})
{ "_id" : 312456, "emp_name" : "Barry Stevens", "emp_age" : 28, "job_role" : "Store Manager", "sala
ry" : 120000 }
{ "_id" : 345342, "emp_name" : "Martin Garrix", "emp_age" : 25, "job_role" : "Store Associate", "sa
lary" : 45000 }
{ "_id" : 334566, "emp_name" : "Linda Harris", "emp_age" : 35, "job_role" : "Cashier", "salary" : 6
7500 }
{ "_id" : 245345, "emp_name" : "Maggie Smith", "emp_age" : 40, "job_role" : "Senior Cashier", "sala
ry" : 72500 }
{ "_id" : 445634, "emp_name" : "Lucy Hale", "emp_age" : 22, "job_role" : "Store Associate", "salary
" : 35000 }
>
```

```
db.employees.find({ "emp_age": { $type: "bool"}})
```

Result:

```
> db.employees.find({ "emp_age": { $type: "bool"}})
>
```

Evaluation operators in MongoDB

The MongoDB evaluation operators can evaluate the overall data structure or individual field in a document. We are only looking at the basic functionality of these operators as each of these operators can be considered an advanced MongoDB functionality. Here is a list of common evaluation operators in MongoDB.

Operator	Description
$jsonSchema	Validate the document according to the given JSON schema.
$mod	Matches documents where a given field's value is equal to the remainder after being divided by a specified value.
$regex	Select documents that match the given regular expression.
$text	Perform a text search on the indicated field. The search can only be performed if the field is indexed with a text index.
$where	Matches documents that satisfy a JavaScript expression.

$jsonSchema Operator

Validate a document by its JSON schema by using the $jsonSchema operator to specify your parameter.

Find documents that match the following JSON schema in the promo collection.

The $let aggregation is used to bind the variables to a results object for simpler output. In the JSON schema, we have specified the minimum value for the "period" field as 7, which will filter out any document with a lesser value.

```
let promoschema = {
bsonType: "object",
required: ,
properties: {
"name": {
bsonType: "string",
description: "promotion name"
},
"period": {
bsonType: "double",
description: "promotion period",
minimum: 7,
maximum: 30
},
"daily_sales": {
bsonType: "array"
}
}
}

db.promo.find({ $jsonSchema: promoschema }).pretty()
```

Result:

```
> let promoschema = {
...     bsonType: "object",
...     required: [ "name", "period", "daily_sales" ],
...     properties: {
...         "name": {
...             bsonType: "string",
...             description: "promotion name"
...         },
...         "period": {
...             bsonType: "double",
...             description: "promotion period",
...             minimum: 7,
...             maximum: 30
...         },
...         "daily_sales": {
...             bsonType: "array"
...         }
...     }
... }
>
> db.promo.find({ $jsonSchema: promoschema }).pretty()
{
        "_id" : "PROMO01",
        "name" : "Sales Promo",
        "period" : 7,
        "daily_sales" : [
                20,
                50,
                12,
                30,
                45,
                15,
                60
        ]
}
{
        "_id" : "PROMO04",
        "name" : "New Year Promo",
        "period" : 7,
        "daily_sales" : [
                65,
                88,
                105,
                188,
                74,
                278,
                350
        ]
}
>
```

$mod Operator

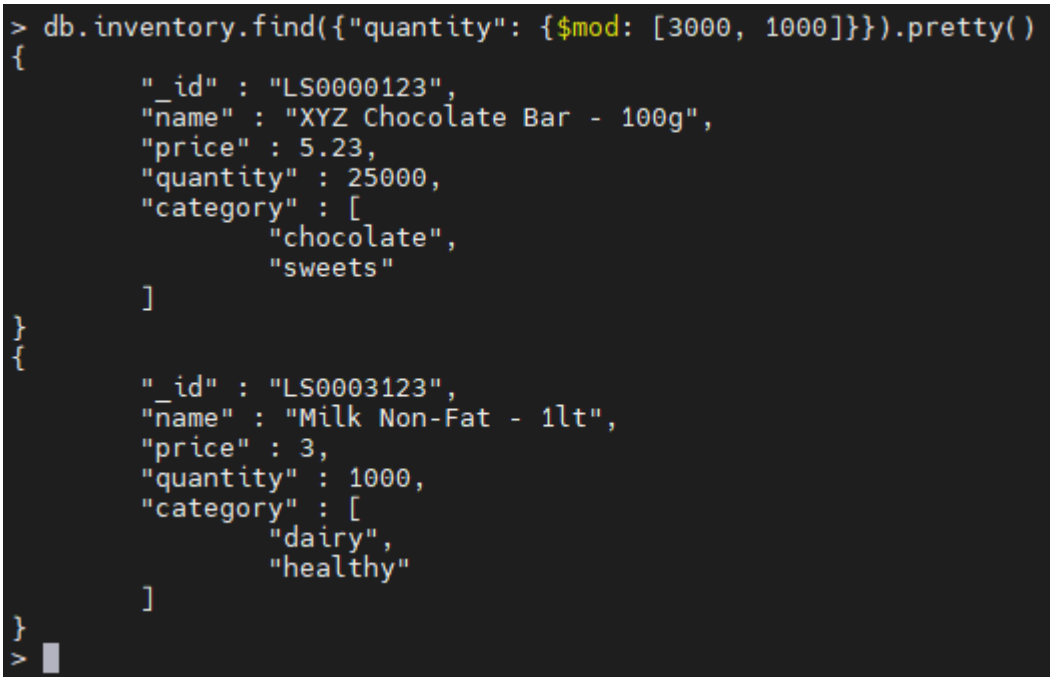
Select results with the $mod operator, which divides the value of a field by a specific number and remainder.

Find documents where the remainder is 1000 when divided by 3000 in the inventory collection.

Note that the document “Milk Non-Fat – 1lt” is included in the output because the quantity is 1000, which cannot be divided by 3000, and the remainder is 1000.

```
db.inventory.find({"quantity": {$mod: }}).pretty()
```

Result:



$regex Operator

Use the $regex operator for a regular expression with the capability of pattern matching within strings.

Find documents that contain the word “Packed” in the name field in the inventory collection.

```
db.inventory.find({"name": {$regex: '.Packed.'}}).pretty()
```

Result:

```
> db.inventory.find({"name": {$regex: '.Packed.'}}).pretty()
{
        "_id" : "LS0009845",
        "name" : "Carrots (Packed) - 250g",
        "price" : 3.59,
        "quantity" : 3000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0009846",
        "name" : "Beans (Packed) - 250g",
        "price" : 6.75,
        "quantity" : 6000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0009100",
        "name" : "Bell Pepper (Packed) - 250g",
        "price" : 4.95,
        "quantity" : 12000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
>
```

$text Operator

To find results that contain a word or phrase, use the $text operator for a text search.

Find documents by using a text searching for “Non-Fat” in the name field. If the field is not indexed, you must create a text index before searching.

```
db.inventory.createIndex({ "name": "text"})
```

Result:

```
> db.inventory.createIndex({ "name": "text"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

```
db.inventory.find({ $text: { $search: "Non-Fat"}}).pretty()
```

Result:

```
> db.inventory.find({ $text: { $search: "Non-Fat"}}).pretty()
{
        "_id" : "LS0003123",
        "name" : "Milk Non-Fat - 1lt",
        "price" : 3,
        "quantity" : 1000,
        "category" : [
                "dairy",
                "healthy"
        ]
}
>
```

$where Operator

To match results to a JavaScript expression, use the $where operator.

Find documents from the "payments" collection where the _id field is a string type and equals the given md5 hash defined as a JavaScript function.

```
db.payments.find({ $where: function() { var value =  isString(this._id) && hex_md5(this._id) == '57fee1331906c3a8f0fa583d37ebbea9'; return value; }}).pretty()
```

Result:

```
> db.payments.find({ $where: function() { var value =  isString(this._id) && hex_md5(this._id) == '57fee1331906c3a8f0fa583d37ebbea9'; return value; }}).pretty()
{
        "_id" : "BL2021005",
        "gross_amount" : 105.65,
        "discounts" : 10,
        "net_amount" : 95.65,
        "date_time" : ISODate("2021-01-01T16:00:00Z")
}
>
```

Array operators in MongoDB

MongoDB array operators are designed to query documents with arrays. Here are the array operators provided by MongoDB.

Operator	Description
$all	Matches arrays that contain all the specified values in the query condition.
$size	Matches the documents if the array size is equal to the specified size in a query.
$elemMatch	Matches documents that match specified $elemMatch conditions within each array element.

$all Operator

For an array that contains every specified value in a query condition, use the $all operator.

Find documents where the category array field contains "healthy" and "organic" values.

```
db.inventory.find({ "category": { $all: }}).pretty()
```

Result:

```
> db.inventory.find({ "category": { $all: ["healthy", "organic"]}}).pretty()
{
        "_id" : "LS0009845",
        "name" : "Carrots (Packed) - 250g",
        "price" : 3.59,
        "quantity" : 3000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0009846",
        "name" : "Beans (Packed) - 250g",
        "price" : 6.75,
        "quantity" : 6000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
{
        "_id" : "LS0009100",
        "name" : "Bell Pepper (Packed) - 250g",
        "price" : 4.95,
        "quantity" : 12000,
        "category" : [
                "vegetables",
                "healthy",
                "organic"
        ]
}
>
```

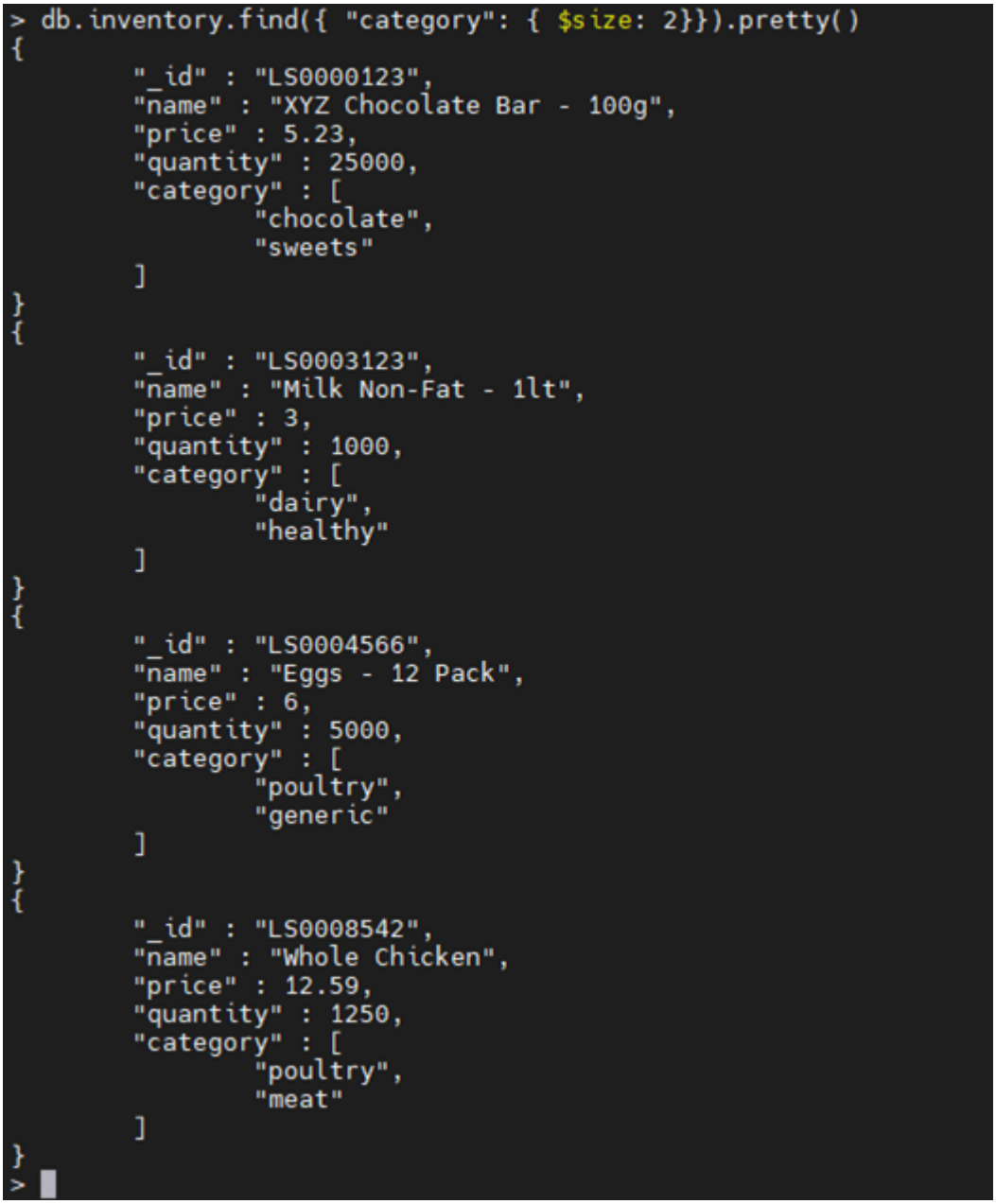
$size Operator

Use the $size operator to filter results to return only those that match a specific size.

Find documents where the category array field has two elements.

```
db.inventory.find({ "category": { $size: 2}}).pretty()
```

Result:



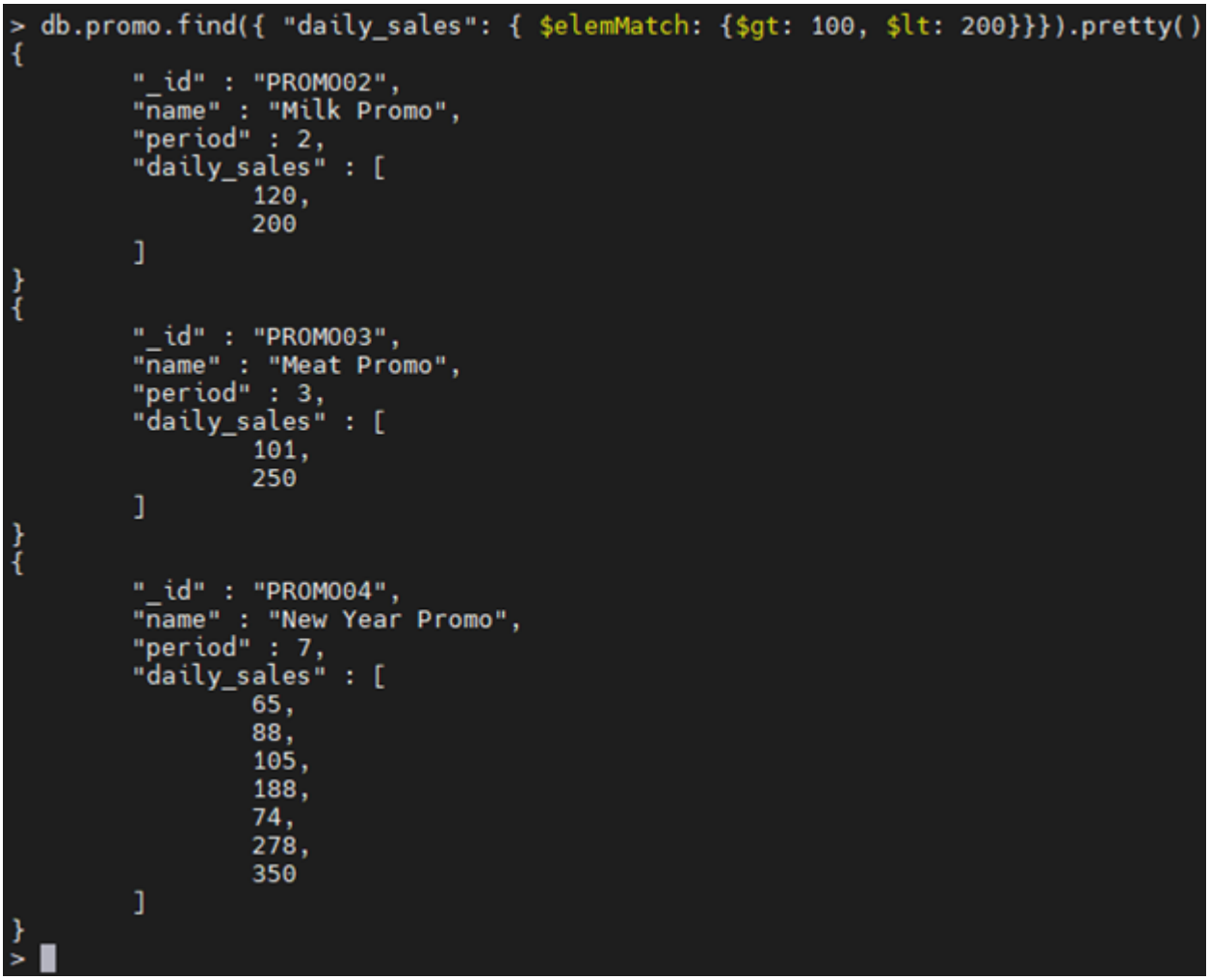
$elemMatch Operator

Use the $elemMatch operator to match results where at least one element satisfies all specified query criteria in an array.

Find documents where at least a single element in the “daily_sales” array is less than 200 and greater than 100.

```
db.promo.find({ "daily_sales": { $elemMatch: {$gt: 100, $lt: 200}}}).pretty()
```

Result:



Comment operator in MongoDB

The MongoDB comment query operator associates a comment to any expression taking a query predicate. Adding comments to queries enables database administrators to trace and interpret MongoDB logs using the comments easily.

$comment Operator

Find documents where the period is equal to 7 in promo collection while adding a comment to the find operation.

```
db.promo.find({ "period": { $eq: 7}, $comment: "Find Weeklong Promos"}).pretty()
```

Result:

```
> db.promo.find({ "period": { $eq: 7}, $comment: "Find Weeklong Promos"}).pretty()
{
        "_id" : "PROMO01",
        "name" : "Sales Promo",
        "period" : 7,
        "daily_sales" : [
                20,
                50,
                12,
                30,
                45,
                15,
                60
        ]
}
{
        "_id" : "PROMO04",
        "name" : "New Year Promo",
        "period" : 7,
        "daily_sales" : [
                65,
                88,
                105,
                188,
                74,
                278,
                350
        ]
}
>
```

Adding comments lets users easily identify commands in MongoDB logs. The above operation will be logged as follows.

```
db.adminCommand( { getLog:'global'} ).log.forEach(x => {print(x)})
```

Result:


```
{"t":{"$date":"2021-01-10T18:21:07.951+00:00"},"s":"I",  "c":"COMMAND",  "id":51803,   "ctx":"conn2
","msg":"Slow query","attr":{"type":"command","ns":"supermarket.promo","appName":"MongoDB Shell","c
ommand":{"find":"promo","filter":{"period":{"$eq":7.0},"$comment":"Find Weeklong Promos"},"comment"
:"Find Weeklong Promos","lsid":{"id":{"$uuid":"5369321e-8385-4de3-9d00-1b5c14a33a70"}},"$db":"super
market"},"planSummary":"COLLSCAN","keysExamined":0,"docsExamined":4,"cursorExhausted":true,"numYiel
ds":0,"nreturned":2,"queryHash":"7A39D598","planCacheKey":"7A39D598","reslen":425,"locks":{"Replica
tionStateTransition":{"acquireCount":{"w":1}},"Global":{"acquireCount":{"r":1}},"Database":{"acquir
eCount":{"r":1}},"Collection":{"acquireCount":{"r":1}},"Mutex":{"acquireCount":{"r":1}}},"storage":
{},"protocol":"op_msg","durationMillis":0}}
```


Discover more MongoDB operators

In this article, we have only scratched the surface of MongoDB operators. We can further extend the overall functionality of the database using projection operators and aggregate functions.

Related reading

- BMC Machine Learning & Big Data Blog
- MongoDB Sharding: Concepts, Examples & Tutorials, part of our MongoDB Guide
- BMC Guides, which offers series of articles on a variety of topics, including Apache Cassandra and Spark, AWS, Machine Learning, and Data Visualization
- Data Storage Explained: Data Lake vs Warehouse vs Database