



LESSON

Querying Data with Operators and Compound Conditions

Google slide deck available [here](#)

This work is licensed under the [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#)
(CC BY-NC-SA 3.0)

Overview



Learning Objectives

At the end of this lesson, learners will be able to:

- Execute more selective queries using MQL operators in conjunction with compound conditions.
- Use aggregation expressions in MQL to query complex data.
- Understand the benefit of combining operators “AND” and “OR” as well as “NOR” and “NOT” to return precise data.
- Describe the importance of cursors in query operations.

Suggested Uses

- A whole lecture or spread out across multiple lecture periods
- Handouts / asynchronous learning
- Supplemental reading material - read on your own / not part of formal teaching
- Complement to University courses [Introduction to MongoDB](#) and [MongoDB for SQL Professionals](#)

This lesson is a part of the courses [Querying in Non-Relational Databases](#) and [Introduction to Modern Databases with MongoDB](#).

At a Glance



Length:
45-55 minutes



Level:
Intermediate



Prerequisites:
[Querying Complex Data in MongoDB with MQL](#)

This work is licensed under the [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#)
(CC BY-NC-SA 3.0)

Share your feedback: We hope these curriculum materials will be a valuable resource for you and your learners. Let us know how the materials work for you, what we can improve on, and how MongoDB for Academia can support you via our brief [feedback form](#).

MongoDB for Academia: MongoDB for Academia offers resources for educators and students to support teaching and learning MongoDB. Check out our [educator resources](#) and join the Educator Community. Students can receive \$50 in Atlas credits and free certification through the [GitHub Student Developer Pack](#).

Last Update: March 2025



This lesson includes exercises

Follow along using these tools

Create a Database

- [MongoDB Atlas](#) (cloud)
- [MongoDB Community](#)
(local install)

Connect to Your Database

- [MongoDB Shell](#)
(open source)

For more instructions on how to use MongoDB Atlas with your students, see [Atlas for Educators](#)

These exercises were designed to work in the MongoDB Shell. If you prefer to use a GUI to interact with your data, please use [MongoDB Compass](#) or the [MongoDB Atlas UI](#).



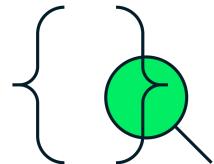
Using Operators and Compound Conditions

Combining operators

Cursors

Multiple compound conditions

Aggregation expressions



We are going to look at using MQL to execute complex queries where we use operators in conjunction with compound queries to further enhance the selectivity of our queries.



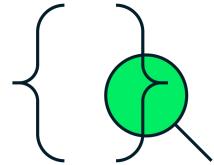
Using Operators and Compound Conditions

Combining operators

Cursors

Multiple compound conditions

Aggregation expressions



As an aside but a related topic, we are going to introduce Cursors to ensure we can take our learnings so far and apply them beyond the Mongo Shell and within MongoDB drivers when we are writing applications to connect to MongoDB.



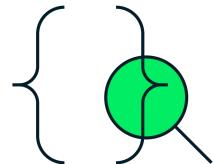
Using Operators and Compound Conditions

Combining operators

Cursors

Multiple compound conditions

Aggregation expressions



We'll then move into exploring multiple compound conditions. In MQL it is possible to use multiple compound conditions and we'll look at using the \$and operator to see how we do this.



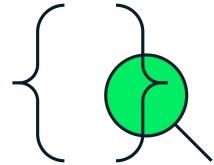
Using operators and compound conditions

Combining operators

Cursors

Multiple compound conditions

Aggregation expressions



Finally, we'll briefly review aggregation expressions as these can be used with MQL as well as within the MongoDB Aggregation Framework. There are some caveats when using them with MQL which will flag.



Combining Operators



Combining Operators

MongoDB operators can be combined allowing for more complex conditions to be included in the query

The boolean operators:

- AND
- OR

NOR and NOT can also be combined

Implied boolean “AND” style of query

The combination of operators in querying is important to allow you to model more complex query logic.

In MongoDB there are two specific operators, the boolean AND as well as the boolean OR operators. There are also the NOR and NOT boolean operators which can also be combined with query logic.

By default all queries in MongoDB match all the fields which implicitly gives a logical AND without using any operator.



Combining Operators: Exercise

First, let's insert some data

```
>>> db.inventory.drop()  
  
>>> db.inventory.insertMany( [  
  
    { "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" },  
  
    { "item": "notebook", "qty": 50, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "A" },  
  
    { "item": "paper", "qty": 100, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "D" },  
  
    { "item": "planner", "qty": 75, "size": { "h": 22.85, "w": 30, "uom": "cm" }, "status": "D" },  
  
    { "item": "postcard", "qty": 45, "size": { "h": 10, "w": 15.25, "uom": "cm" }, "status": "A" }  
  
]);  
  
...
```

Follow along using the [MongoDB Shell](#). If you prefer to use a GUI to interact with your data, please use [MongoDB Compass](#) or the [MongoDB Atlas UI](#).

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen. We'll use this data for exploring how we can combine operators.

```
db.inventory.drop()  
db.inventory.insertMany( [  
    { "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" },  
    { "item": "notebook", "qty": 50, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "A" },  
    { "item": "paper", "qty": 100, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "D" },  
    { "item": "planner", "qty": 75, "size": { "h": 22.85, "w": 30, "uom": "cm" }, "status": "D" },  
    { "item": "postcard", "qty": 45, "size": { "h": 10, "w": 15.25, "uom": "cm" }, "status": "A" }  
]);
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/>



Combining Operators: Exercise

Let's query using an implicit AND and OR

```
>>> db.inventory.find( { status: "A", $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ] } )
```

```
{ "_id" : ObjectId("5f3bfffba63a92a8719c01243"), "item" : "postcard", "status" : "A", "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "instock" : [ { "warehouse" : "B", "qty" : 15 }, { "warehouse" : "C", "qty" : 35 } ] }
```

```
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944a"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
```

```
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944e"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here.

```
db.inventory.find( { status: "A", $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ] } )
```

This query is using an implicit AND combined with an OR condition. It will select all the documents where the status is “A” **and** either (**or**) the qty is less than 30 or the item starts with the character ‘p’. Note the use of anchoring and the regular expression. MongoDB supports regular expressions within queries.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Combining Operators: Exercise

Let's query using an explicit AND and OR

```
>>> db.inventory.find( { $and: [ { $or: [ { qty: { $lt : 10 } }, { qty : { $gt: 50 } } ] }, { $or: [ { status: "A" }, { "size.w" : { $lt : 15 } } ] } ] }
```

```
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here.

```
db.inventory.find( { $and: [ { $or: [ { qty: { $lt : 10 } }, { qty : { $gt: 50 } } ] }, { $or: [ { status: "A" }, { "size.w" : { $lt : 15 } } ] } ] }
```

This is a more complex statement combining AND and OR so let's break it down. The AND operator allows us to use the OR operator twice. The first OR operator is looking for documents where the qty field is less than 10 or greater than 50. The second OR operator is querying for documents with a status field equal to "A" or where the w field in the size document is less than 15. The AND operator looks from results from both of the OR operators to see where any documents have fulfill one or both of the query parameters within the specific OR statement and equally one or both in the other OR statement.

There is only one resulting document that fulfils these criteria, specifically in the first OR clause it has a qty field of 100 which is greater than 50 and for the second OR clause it has a w field in the size document less than 15.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Combining Operators: Exercise

Let's query using the logical NOT operator

```
>>> db.inventory.find( { item: { $not: { $regex: /^p.*/ } } } )  
  
{ "_id" : ObjectId("5f3bfffba63a92a8719c0123f"), "item" : "journal", "status" : "A", "size" : {  
"h" : 14, "w" : 21, "uom" : "cm" }, "instock" : [ { "warehouse" : "A", "qty" : 5 } ] }  
  
{ "_id" : ObjectId("5f3bfffba63a92a8719c01240"), "item" : "notebook", "status" : "A", "size" : {  
"h" : 8.5, "w" : 11, "uom" : "in" }, "instock" : [ { "warehouse" : "C", "qty" : 5 } ] }  
  
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944a"), "item" : "journal", "qty" : 25, "size" : { "h"  
: 14, "w" : 21, "uom" : "cm" }, "status" : "A" }  
  
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944b"), "item" : "notebook", "qty" : 50, "size" : {  
"h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here.

```
db.inventory.find( { item: { $not: { $regex: /^p.*/ } } } )
```

In this example, we use the \$not operator and the \$regex (regular expression) operator to find all the documents where the item field does not start with the letter 'p'.

See: <https://docs.mongodb.com/manual/reference/operator/query/not/>



Combining Operators: Exercise

Let's query using the logical NOT operator

Using the same window, change <A> to the and operator, to the less than operator, <C> to the greater than operator and <D> to the status field.

```
>>> db.inventory.find( { $and: [ { <A>: [ { qty: { <B> : 150 } }, { qty : {  
<C>: 50 } } ] }, { "<D>" : "D" } ] } )  
  
{ "_id" : ObjectId("5f55eeeea2d4b45b7f11b6d96"), "item" : "paper", "qty" : 100,  
"size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }  
  
{ "_id" : ObjectId("5f55eeeea2d4b45b7f11b6d97"), "item" : "planner", "qty" :  
75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
```

Now to use the MQL find() and you should follow the instructions to complete the query above. It is a good example of a more complex MQL statement using multiple operators. You can copy it from the slide or from the solution from the notes here

```
db.inventory.find( { $and: [ { $and: [ { qty: { $lt : 150  
} }, { qty : { $gt: 50 } } ] }, { "status" : "D" } ] } )  
{ "_id" : ObjectId("5f55eeeea2d4b45b7f11b6d96"), "item" :  
"paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11,  
"uom" : "in" }, "status" : "D" }  
{ "_id" : ObjectId("5f55eeeea2d4b45b7f11b6d97"), "item" :  
"planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30,  
"uom" : "cm" }, "status" : "D" }
```

Two documents are returned with exact matches for the query, where all the conditions of the query are explicitly matched.

See: <https://docs.mongodb.com/manual/reference/operator/query/not/>

Quiz





Quiz

Which of the following are true for combining operators in MQL?

- A. Multiple separate \$or clauses can be combined without any additional operators in MQL
- B. Implicit AND is the default logic for MongoDB query criteria
- C. \$NOR and \$NOT are boolean combination operators in MQL



Quiz

Which of the following are true for combining operators in MQL?

- A. Multiple separate \$or clauses can be combined without any additional operators in MQL
- B. Implicit AND is the default logic for MongoDB query criteria
- C. \$NOR and \$NOT are boolean combination operators in MQL

INCORRECT: Multiple separate \$or clauses can be combined without any additional operators in MQL - To combine multiple or clauses you need to use the AND operator

CORRECT: Implicit AND is the default logic for MongoDB query criteria - This is the standard / default in MQL queries.

CORRECT: \$NOR and \$NOT are boolean combination operators in MQL - These with \$OR and \$AND are the boolean operators in MQL.



Quiz

Which of the following are true for combining operators in MQL?

- A. Multiple separate \$or clauses can be combined without any additional operators in MQL
- B. Implicit AND is the default logic for MongoDB query criteria
- C. \$NOR and \$NOT are boolean combination operators in MQL

This incorrect. To combine multiple or clauses you need to use the \$and operator which is required as an additional operator.

INCORRECT: Multiple separate \$or clauses can be combined without any additional operators in MQL - To combine multiple or clauses you need to use the AND operator



Quiz

Which of the following are true for combining operators in MQL?

- A. Multiple separate \$or clauses can be combined without any additional operators in MQL
- B. Implicit AND is the default logic for MongoDB query criteria
- C. \$NOR and \$NOT are boolean combination operators in MQL

This is correct. This is the standard / default in MQL queries.

CORRECT: Implicit AND is the default logic for MongoDB query criteria - This is correct. This is the standard / default in MQL queries.



Quiz

Which of the following are true for combining operators in MQL?

- A. Multiple separate \$or clauses can be combined without any additional operators in MQL
- B. Implicit AND is the default logic for MongoDB query criteria
- C. \$NOR and \$NOT are boolean combination operators in MQL

This is correct. These with \$OR and \$AND are the boolean operators in MQL.

CORRECT: \$NOR and \$NOT are boolean combination operators in MQL - This is correct. These with \$OR and \$AND are the boolean operators in MQL.



Aggregation Expressions



Aggregation Expressions

\$expr operator allows the use of aggregation expressions in MQL

```
{ $expr: { <expression> } }
```

Expressions consist of field paths, literals, system variables, expression objects, and expression operators. These can be nested.

The \$expr operator was introduced to allow MQL to take advantages of the wider range of query functionality built into the Aggregation framework.



It is possible to use aggregation expressions within the MongoDB Query Language (MQL) but you can only do so by using the \$expr operator which allows for these powerful expressions and functionality to be used in MQL.

The \$expr takes an expression as its parameter. We'll look at various expressions and the range of expressions in the aggregation framework.



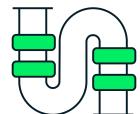
Aggregation Expressions

\$expr operator allows the use of aggregation expressions in MQL

```
{ $expr: { <expression> } }
```

Expressions consist of field paths, literals, system variables, expression objects, and expression operators. These can be nested.

The \$expr operator was introduced to allow MQL to take advantages of the wider range of query functionality built into the Aggregation framework.



Aggregation expressions belong to one of several categories, these include field paths, literals, system variables, expression objects, and expression operators.

It is useful to note that aggregation expressions can be nested.



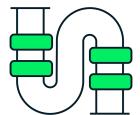
Aggregation Expressions

\$expr operator allows the use of aggregation expressions in MQL

```
{ $expr: { <expression> } }
```

Expressions consist of field paths, literals, system variables, expression objects, and expression operators. These can be nested.

The \$expr operator was introduced to allow MQL to take advantages of the wider range of query functionality built into the Aggregation framework.



Before we move into the various categories of aggregation expressions, it's important to reiterate the primary rationale for the \$expr operator was to allow MQL to leverage all of the query functionality that had been built into the Aggregation Framework.



Aggregation Expressions: Field paths

Field paths are strings where the field name is prefixed by a dollar sign (\$).

The field path will either be a field name or dotted field name (*if the field is an embedded document*).

Taking a field, ‘product’ to specify it as a field path it would be ‘\$product’. ‘\$product.price’ would be how you specify ‘product.price’ field if an embedded field.



Field paths are identified by a dollar sign prefix and are normal strings where the field name is prefixed by a dollar sign (\$).

The field path will either be a field name or a dotted field name, if the field is an embedded document.

Taking an example, the field ‘product’ would be represented as ‘\$product’ as a field path. If it was an embedded document with a price field so ‘product.price’ then it would be represented as ‘\$product.price’ for its field path.



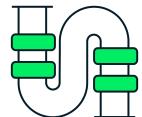
Aggregation Expressions: Variables

System variables, useful helpers such as NOW, ROOT, CLUSTER_TIME, etc.

User defined variables, hold any kind of BSON type data

`$$<variable>"`

`"$$<variable>.<field>"`



Variables in aggregation expressions are used to provide helper functions or to hold BSON type data. There are two types, system and user defined variables.

System variables, are helpers provided by the Aggregation Framework like NOW which gives the current time, ROOT which provides the root of the document, and CLUSTER_TIME which gives the logical time of the cluster at this point in time.

User defined variables are used to hold BSON type data.

They can be referred to by a double dollar and their field name or by a double dollar and the embedded document with it's field name if it is an embedded document.

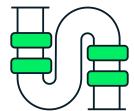


Aggregation Expressions: Literals

Literals can be of any type, they are used for values that you do not want parsed but rather interpreted as it is.

You can use \$literal operator

```
{ $literal: <value> }  
{ $literal: { $add: [ 2, 3 ] } }  
• { "$add": [ 2, 3 ] }
```



A literal is used where you want to return an expression without performing any processing on it. The value is passed as it is and is in no way interpreted.

The example of \$literal \$add 2,3 would be interpreted as { "\$add": [2, 3] }

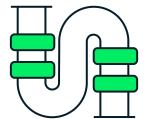


Aggregation Expressions: Objects

Expression objects, a set of expressions

```
{ <field1>: <expression1>, ... }
```

If an expression is a literal and either numeric or boolean, it will be treated as a projection flag unless combined with the \$literal operator.



Expression objects can be used to hold a set of expressions in a single container or object.

Expression objects often use the \$literal operator as otherwise expressions which are numeric or boolean can be interpreted as projection flags.



Aggregation Expressions: Operators

- Arithmetic Expression Operators
- Array Expression Operators
- Boolean Expression Operators
- Comparison Expression Operators
- Conditional Expression Operators
- Custom Aggregation Expression Operators
- Data Size Expression Operators
- Date Expression Operators
- Literal Expression Operator
- Object Expression Operators
- Set Expression Operators
- String Expression Operators
- Text Expression Operator
- Trigonometry Expression Operators
- Type Expression Operators
- Accumulators (\$group)
- Accumulators (in Other Stages)
- Variable Expression Operators

The Aggregation Framework has a large selection of aggregation expressions, this slides simply lists the operator categories, each of which has multiple aggregation expressions. You can see:

Arithmetic Expressions

To

Boolean Expressions

To

String Expressions amongst the wide range of aggregation expressions on this slide alone



Aggregation Expressions: Operators

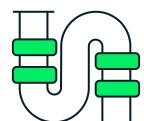
Like functions and take arguments, typically these are an array of arguments:

Array of arguments:

```
{ <operator>: [ <argument1>, <argument2> ... ] }
```

Single argument:

```
{ <operator>: <argument> }
```



Aggregation expressions work like functions and take arguments, either a single argument or more typically an array of arguments.



Aggregation expressions: Operators

- `$abs`
- `$accumulator`
- `$acos`
- `$acosh`
- `$add`
- `$addToSet`
- `$allElementsTrue`
- `$and`
- `$anyElementTrue`
- `$arrayElemAt`
- `$arrayToObject`
- `$asin`
- `$asinh`
- `$atan`
- `$atan2`
- `$atanh`
- `$avg`
- `$binarySize`
- `$bsonSize`
- `$ceil`
- `$cmp`
- `$concat`
- `$concatArrays`
- `$cond`
- `$convert`
- `$cos`
- `$dateFromParts`
- `$dateFromString`
- `$dateToParts`
- `$dateToString`
- `$dayOfMonth`
- `$dayOfWeek`
- `$dayOfYear`
- `$degreesToRadians`
- `$divide`
- `$eq`
- `$exp`
- `$filter`
- `$first(array)`
- `$first(accumulator)`
- `$floor`
- `$function`
- `$gt`
- `$gte`
- `$hour`
- `$ifNull`
- `$in`
- `$indexFromArray`
- `$indexOfBytes`
- `$indexOfCP`

There are a lot of aggregation expression operators, we'll cover some of these in more depth in the Aggregation Framework lesson. In this section, we'll just highlight that you can use these with MQL via the `$expr` operator.

The expression operators cover arithmetic, array, boolean, comparison, conditional, date, object, string, accumulators and other functionality.



Aggregation expressions: Operators

- `$isArray`
- `$isNumber`
- `$isoDayOfWeek`
- `$isoWeek`
- `$isoWeekYear`
- `$last (array)`
- `$last (accumulator)`
- `$let`
- `$literal`
- `$ln`
- `$log`
- `$log10`
- `$lt`
- `$lte`
- `$ltrim`
- `$map`
- `$max`
- `$mergeObjects`
- `$meta`
- `$millisecond`
- `$min`
- `$minute`
- `$mod`
- `$month`
- `$multiply`
- `$ne`
- `$not`
- `$objectToArray`
- `$or`
- `$pow`
- `$push`
- `$radiansToDegrees`
- `$range`
- `$reduce`
- `$regexFind`
- `$regexFindAll`
- `$regexMatch`
- `$replaceOne`
- `$replaceAll`
- `$reverseArray`
- `$round`
- `$rtrim`

Here is yet more aggregation expression operators cover arithmetic, array, boolean, comparison, conditional, date, object, string, accumulators and other functionality.

The purpose of this slide isn't to examine any operator in depth but rather to give a flavour of the breath/spectrum of the available aggregation expressions.



Aggregation expressions: Operators

- `$second`
- `$setDifference`
- `$setEquals`
- `$setIntersection`
- `$setIsSubset`
- `$setUnion`
- `$sin`
- `$size`
- `$slice`
- `$split`
- `$sqrt`
- `$stdDevPop`
- `$stdDevSamp`
- `$strLenBytes`
- `$strLenCP`
- `$strcasecmp`
- `$substr`
- `$substrBytes`
- `$substrCP`
- `$subtract`
- `$sum`
- `$switch`
- `$tan`
- `$toBool`
- `$toDate`
- `$toDouble`
- `$toInt`
- `$toLong`
- `$toLowerCase`
- `$toObjectId`
- `$toString`
- `$toUpperCase`
- `$trim`
- `$trunc`
- `$type`
- `$week`
- `$year`
- `$zip`

Here is the final set of functionality available in aggregation expressions.

The Aggregation Framework and particularly aggregation expression are an area within the MongoDB database that's seen a significant improved in terms of functionality with new operators added in almost every release since the Aggregation Framework was introduced.



Example: Aggregation Expression

```
>>> db.inventory.find( { $expr: { $gt: [ "$qty" , 50 ] } } )  
  
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944c"), "item" :  
"paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" :  
"in" }, "status" : "D" }  
  
{ "_id" : ObjectId("5f3cfb9cc7701a1b3b6e944d"), "item" :  
"planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" :  
"cm" }, "status" : "D" }
```

In this query, the \$expr operator is using the aggregation expression \$gt (greater than) operator to only return documents where the quantity ('qty') field is greater than 50.

This shows how to use an aggregation expression in this case with the greater than (\$gt) operator to find all documents where the “qty” (quantity) field is greater than 50.

If you need further details on \$expr see:

https://docs.mongodb.com/manual/reference/operator/query/expr/#op._S_expr.



Quiz



Quiz

Which of the following are true for aggregation expression in MQL?

- A. Can have one or more expressions in a query
- B. Use dotted names instead of field paths
- C. Works with system or with user defined variables
- D. Object expressions are not usable in MQL and are only usable in the Aggregation Framework



Quiz

Which of the following are true for aggregation expression in MQL?

- A. Can have one or more expressions in a query
- B. Use dotted names instead of field paths
- C. Works with system or with user defined variables
- D. Object expressions are not usable in MQL and are only usable in the Aggregation Framework

CORRECT: Can have one or more expressions in a query - This is correct, the syntax varies slightly as noted.

INCORRECT: Use dotted names instead of field paths - This is incorrect, field paths are used with aggregation expressions. Dotted names as a concept are used within field paths.

CORRECT: Works with system or with user defined variables - This is correct.

INCORRECT: Object expressions are not usable in MQL and are only usable in the Aggregation Framework - This is incorrect, you can use singular aggregation expressions or an object consisting of aggregation expressions.



Quiz

Which of the following are true for aggregation expression in MQL?

- A. Can have one or more expressions in a query
- B. Use dotted names instead of field paths
- C. Works with system or with user defined variables
- D. Object expressions are not usable in MQL and are only usable in the Aggregation Framework

This is correct. It is possible to have multiple expressions in a query, however the syntax varies slightly as noted.

CORRECT: Can have one or more expressions in a query - This is correct. It is possible to have multiple expressions in a query, however the syntax varies slightly as noted.



Quiz

Which of the following are true for aggregation expression in MQL?

- A. Can have one or more expressions in a query
- B. Use dotted names instead of field paths
- C. Works with system or with user defined variables
- D. Object expressions are not usable in MQL and are only usable in the Aggregation Framework

This incorrect. Field paths are used with aggregation expressions. Dotted names as a concept are used within field paths.

INCORRECT: Use dotted names instead of field paths - This is incorrect. Field paths are used with aggregation expressions. Dotted names as a concept are used within field paths.



Quiz

Which of the following are true for aggregation expression in MQL?

- A. Can have one or more expressions in a query
- B. Use dotted names instead of field paths
- C. Works with system or with user defined variables
- D. Object expressions are not usable in MQL and are only usable in the Aggregation Framework

This is correct. System or user defined variables can be used in aggregation expressions in MQL.

CORRECT: Works with system or with user defined variables - This is correct. System or user defined variables can be used in aggregation expressions in MQL.



Quiz

Which of the following are true for aggregation expression in MQL?

- A. Can have one or more expressions in a query
- B. Use dotted names instead of field paths
- C. Works with system or with user defined variables
- D. Object expressions are not usable in MQL and are only usable in the Aggregation Framework

This incorrect. You can use singular aggregation expressions or an object consisting of aggregation expressions.

INCORRECT: Object expressions are not usable in MQL and are only usable in the Aggregation Framework - This is incorrect. You can use singular aggregation expressions or an object consisting of aggregation expressions.



Cursors



Cursors

MongoDB queries return a cursor, a pointer to the result set of documents

Cursors can be iterated through to retrieve the results

10 minutes of inactivity before timing out

Various options available within the MongoDB Shell or via MongoDB's drivers

count(), hint(), limit(), readPreference(), readConcern(), skip(), sort(), etc.

The cursor is the underlying mechanism used to return the documents when you query a MongoDB database.



Cursors

MongoDB queries return a cursor, a pointer to the result set of documents

Cursors can be iterated through the results

10 minutes of inactivity before timing out

Various options available within the MongoDB Shell or via MongoDB's drivers

count(), hint(), limit(), readPreference(), readConcern(), skip(), sort(), etc.

You can iterate through the result set of your query using the cursor, it is the helper functionality that allows for you to easily access and traverse the results.



Cursors

MongoDB queries return a cursor, a pointer to the result set of documents

Cursors can be iterated through to retrieve the results

10 minutes of inactivity before timing out

Various options available within the MongoDB Shell or via MongoDB's drivers

count(), hint(), limit(), readPreference(), readConcern(), skip(), sort(), etc.

Cursors only remain active for 10 minutes after which they time out and you will need to make another request of the database. This helps avoid situations where idle cursors stay open and use machine resources but actually aren't doing any querying.



Cursors

MongoDB queries return a cursor, a pointer to the result set of documents

Cursors can be iterated through to retrieve the results

10 minutes of inactivity before timing out

Various options available within the MongoDB Shell or via MongoDB's drivers `count()`, `hint()`, `limit()`, `readPreference()`, `readConcern()`, `skip()`, `sort()`, etc.

Cursors have a host of functionality and options that can be used with them including index hinting, read concerns and read preferences as well as more typical functionality around skipping, limiting, and count the number of documents in the result set.

These options exist in both the MongoDB Shell and in the various drivers, however they may have slightly different naming conventions for the functionality depending on the language.



Quiz



Quiz

Which of the following are true for cursors in MQL?

- A. Stay active/open indefinitely by default
- B. Supports read concerns and read preferences
- C. Does not support index hinting
- D. Uses exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell



Quiz

Which of the following are true for cursors in MQL?

- A. Stay active/open indefinitely by default
- B. Supports read concerns and read preferences
- C. Does not support index hinting
- D. Uses exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell

INCORRECT: Stay active/open indefinitely by default - Cursors only stay active for 10 minutes without any activity before being closed

CORRECT: Supports read concerns and read preferences - This is correct

INCORRECT: Does not support index hinting - Cursors can and often do use hinting to improve performance with a selected index

INCORRECT: Use exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell - The Drivers will offer the same functionality but the naming conventions may differ slightly depending on the programming language.



Quiz

Which of the following are true for cursors in MQL?

- A. Stay active/open indefinitely by default
- B. Supports read concerns and read preferences
- C. Does not support index hinting
- D. Uses exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell

This incorrect. Cursors only stay active for 10 minutes without any activity before being closed.

INCORRECT: Stay active/open indefinitely by default - This is incorrect. Cursors only stay active for 10 minutes without any activity before being closed.



Quiz

Which of the following are true for cursors in MQL?

- A. Stay active/open indefinitely by default
- B. Supports read concerns and read preferences
- C. Does not support index hinting
- D. Uses exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell

This is correct. Cursors support read concerns as well as read preferences.

CORRECT: Supports read concerns and read preferences - This is correct. Cursors support read concerns as well as read preferences.



Quiz

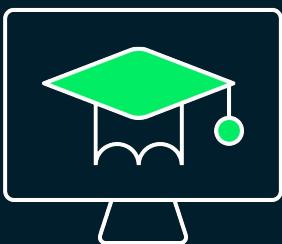
Which of the following are true for cursors in MQL?

- A. Stay active/open indefinitely by default
- B. Supports read concerns and read preferences
- C. Does not support index hinting
- D. Uses exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell

This incorrect. The Drivers will offer the same functionality but the naming conventions may differ slightly depending on the programming language.

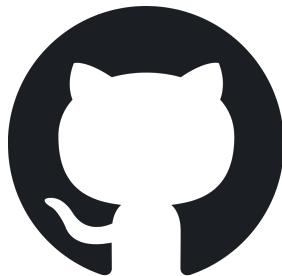
INCORRECT: Use exactly the same naming conventions across the MongoDB Drivers and the MongoDB Shell - This is incorrect. The Drivers will offer the same functionality but the naming conventions may differ slightly depending on the programming language.

Continue Learning!



[MongoDB University](#) has free self-paced courses and labs ranging from beginner to advanced levels.

GitHub Student Developer Pack



Sign up for the [MongoDB Student Pack](#) to receive \$50 in Atlas credits and free certification!

This concludes the material for this lesson. However, there are many more ways to learn about MongoDB and non-relational databases, and they are all free! Check out [MongoDB's University](#) page to find free courses that go into more depth about everything MongoDB and non-relational. For students and educators alike, MongoDB for Academia is here to offer support in many forms. Check out our [educator resources](#) and join the Educator Community. Students can receive \$50 in Atlas credits and free certification through the [GitHub Student Developer Pack](#).