# MongoDB CRUD Operations[IMG_256](https://www.mongodb.com/docs/manual/crud/#mongodb-crud-operations)

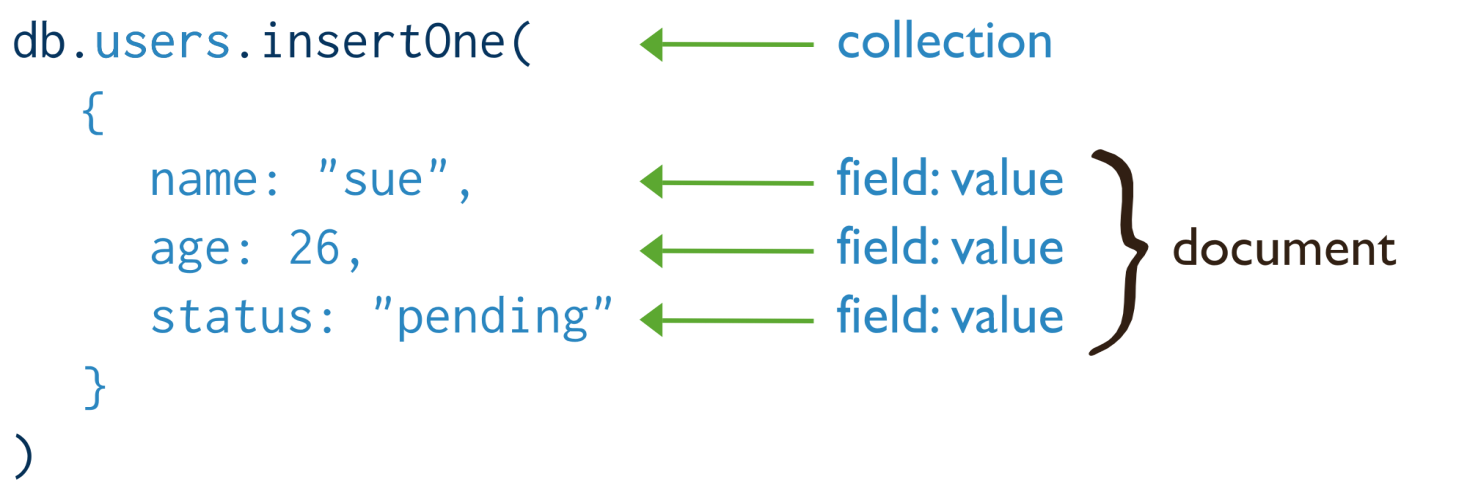
## Create Operations[IMG_256](https://www.mongodb.com/docs/manual/crud/#create-operations)

Create or insert operations add new [documents](https://www.mongodb.com/docs/manual/core/document/" \l "std-label-bson-document-format) to a [collection](https://www.mongodb.com/docs/manual/core/databases-and-collections/" \l "std-label-collections). If the collection does not currently exist, insert operations will create the collection.

[db.collection.insertOne()](https://www.mongodb.com/docs/manual/reference/method/db.collection.insertOne/" \l "mongodb-method-db.collection.insertOne) New in version 3.2

[db.collection.insertMany()](https://www.mongodb.com/docs/manual/reference/method/db.collection.insertMany/" \l "mongodb-method-db.collection.insertMany) New in version 3.2

In MongoDB, insert operations target a single [collection](https://www.mongodb.com/docs/manual/reference/glossary/" \l "std-term-collection). All write operations in MongoDB are [atomic](https://www.mongodb.com/docs/manual/core/write-operations-atomicity/) on the level of a single [document.](https://www.mongodb.com/docs/manual/core/document/)



### InsertOne:

|  |
| --- |
| db.inventory.insertOne( |
| { item: **"canvas"**, qty: 100, tags: [**"cotton"**], size: { h: 28, w: 35.5, uom: **"cm"** } } |
| ) |

### Find:

db.inventory.find( { item: **"canvas"** } )

### InsertMany:

|  |
| --- |
| db.inventory.insertMany([ |
| { item: **"journal"**, qty: 25, tags: [**"blank"**, **"red"**], size: { h: 14, w: 21, uom: **"cm"** } }, |
| { item: **"mat"**, qty: 85, tags: [**"gray"**], size: { h: 27.9, w: 35.5, uom: **"cm"** } }, |
| { item: **"mousepad"**, qty: 25, tags: [**"gel"**, **"blue"**], size: { h: 19, w: 22.85, uom: **"cm"** } } |
| ]) |

To retrieve the inserted documents, [query the collection:](https://www.mongodb.com/docs/manual/tutorial/query-documents/" \l "std-label-read-operations-query-document)

|  |
| --- |
| db.inventory.find( {} ) |

### Additional Methods for Inserts[IMG_256](https://www.mongodb.com/docs/manual/reference/insert-methods/#additional-methods-for-inserts)

The following methods can also add new documents to a collection:

[db.collection.updateOne()](https://www.mongodb.com/docs/manual/reference/method/db.collection.updateOne/" \l "mongodb-method-db.collection.updateOne) when used with the upsert: true option.

[db.collection.updateMany()](https://www.mongodb.com/docs/manual/reference/method/db.collection.updateMany/" \l "mongodb-method-db.collection.updateMany) when used with the upsert: true option.

[db.collection.findAndModify()](https://www.mongodb.com/docs/manual/reference/method/db.collection.findAndModify/" \l "mongodb-method-db.collection.findAndModify) when used with the upsert: true option.

[db.collection.findOneAndUpdate()](https://www.mongodb.com/docs/manual/reference/method/db.collection.findOneAndUpdate/" \l "mongodb-method-db.collection.findOneAndUpdate) when used with the upsert: true option.

[db.collection.findOneAndReplace()](https://www.mongodb.com/docs/manual/reference/method/db.collection.findOneAndReplace/" \l "mongodb-method-db.collection.findOneAndReplace) when used with the upsert: true option.

[db.collection.bulkWrite().](https://www.mongodb.com/docs/manual/reference/method/db.collection.bulkWrite/" \l "mongodb-method-db.collection.bulkWrite)

## Query Documents[IMG_256](https://www.mongodb.com/docs/manual/tutorial/query-documents/#query-documents)

|  |
| --- |
| [ |
| { **"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"** } |
| ] |

### Select ALL:

SELECT \* FROM inventory

{}

SELECT \* FROM inventory WHERE status = "D"

{ status: **"D"** }

The following example retrieves all documents from the inventory collection where status equals either "A" or "D":

### IN or Contains:

SELECT \* FROM inventory WHERE status in ("A", "D")

{ status: { $in: [ **"A"**, **"D"** ] } }

### AND:

SELECT \* FROM inventory WHERE status = "A" AND qty < 30

{ status: **"A"**, qty: { $lt: 30 } }

### OR:

{ $or: [ { status: "A" }, { qty: { $lt: 30 } } ] }

SELECT \* FROM inventory WHERE status = "A" OR qty < 30

### Specify AND as well as OR Conditions

{ status: **"A"**, $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ] }

### LIKE

The operation uses a filter predicate of:

|  |
| --- |
| { |
| status: **'A'**, |
| $or: [ |
| { qty: { $lt: 30 } }, { item: { $regex: **'^p'** } } |
| ] |
| } |

SELECT \* FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")

## Query on Embedded/Nested Documents

### **Specify Equality Match on a Nested Field[IMG_256](https://www.mongodb.com/docs/manual/tutorial/query-embedded-documents/#specify-equality-match-on-a-nested-field)**

The following example selects all documents where the field uom nested in the size field equals "in":

Copy the following filter into the Compass query bar and click **Find**:

|  |
| --- |
| { **"size.uom"**: **"in"** } |

### Query an Array

|  |
| --- |
| await db.collection(**'inventory'**).insertMany([ |
| { |
| item: **'journal'**, |
| qty: 25, |
| tags: [**'blank'**, **'red'**], |
| dim\_cm: [14, 21] |
| }, |
| { |
| item: **'notebook'**, |
| qty: 50, |
| tags: [**'red'**, **'blank'**], |
| dim\_cm: [14, 21] |
| }, |
| { |
| item: **'paper'**, |
| qty: 100, |
| tags: [**'red'**, **'blank'**, **'plain'**], |
| dim\_cm: [14, 21] |
| }, |
| { |
| item: **'planner'**, |
| qty: 75, |
| tags: [**'blank'**, **'red'**], |
| dim\_cm: [22.85, 30] |
| }, |
| { |
| item: **'postcard'**, |
| qty: 45, |
| tags: [**'blue'**], |
| dim\_cm: [10, 15.25] |
| } |
| ]); |

|  |
| --- |
| const cursor = db.collection(**'inventory'**).find({ |
| tags: [**'red'**, **'blank'**] |
| }); |

**Match all values:**

using $all

|  |
| --- |
| const cursor = db.collection(**'inventory'**).find({ |
| tags: { $all: [**'red'**, **'blank'**] } |
| }); |

### Projection

SELECT **item, status** from inventory WHERE status = "A"

|  |
| --- |
| const cursor = db |
| .collection(**'inventory'**) |
| .find({ |
| status: **'A'** |
| }) |
| .project({ **item: 1, status: 1, \_id: 0** }); |

## Manually Iterate the Cursor[IMG_256](https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/#manually-iterate-the-cursor)

In [mongosh](https://www.mongodb.com/docs/mongodb-shell/" \l "mongodb-binary-bin.mongosh" \t "https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/_self), when you assign the cursor returned from the [find()](https://www.mongodb.com/docs/manual/reference/method/db.collection.find/" \l "mongodb-method-db.collection.find) method to a variable using the var keyword, the cursor does not automatically iterate.

You can call the cursor variable in the shell to iterate up to 20 times [[1]](https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/" \l "footnote-set-shell-batch-size) and print the matching documents, as in the following example:

|  |
| --- |
| var myCursor = db.users.find( { type: 2 } ); |
|  |
| myCursor |

You can also use the cursor method [next()](https://www.mongodb.com/docs/manual/reference/method/cursor.next/" \l "mongodb-method-cursor.next) to access the documents, as in the following example:

|  |
| --- |
| var myCursor = db.users.find( { type: 2 } ); |
|  |
| while (myCursor.hasNext()) { |
| print(tojson(myCursor.next())); |
| } |

As an alternative print operation, consider the printjson() helper method to replace print(tojson()):

|  |
| --- |
| var myCursor = db.users.find( { type: 2 } ); |
|  |
| while (myCursor.hasNext()) { |
| printjson(myCursor.next()); |
| } |

You can use the cursor method [forEach()](https://www.mongodb.com/docs/manual/reference/method/cursor.forEach/" \l "mongodb-method-cursor.forEach) to iterate the cursor and access the documents, as in the following example:

|  |
| --- |
| var myCursor = db.users.find( { type: 2 } ); |
|  |
| myCursor.forEach(printjson); |

See [JavaScript cursor methods](https://www.mongodb.com/docs/manual/reference/method/" \l "std-label-js-query-cursor-methods) and your [driver](https://www.mongodb.com/docs/drivers/" \t "https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/_self) documentation for more information on cursor methods.

|  |  |
| --- | --- |
| [1] | ([1](https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/" \l "ref-set-shell-batch-size-id1), [2](https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/" \l "ref-set-shell-batch-size-id2)) You can set the DBQuery.shellBatchSize attribute to change the number of documents from the default value of 20. |

## Iterator Index[IMG_257](https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/#iterator-index)

In [mongosh](https://www.mongodb.com/docs/mongodb-shell/" \l "mongodb-binary-bin.mongosh" \t "https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/_self), you can use the [toArray()](https://www.mongodb.com/docs/manual/reference/method/cursor.toArray/" \l "mongodb-method-cursor.toArray) method to iterate the cursor and return the documents in an array, as in the following:

|  |
| --- |
| var myCursor = db.inventory.find( { type: 2 } ); |
| var documentArray = myCursor.toArray(); |
| var myDocument = documentArray[3]; |

The [toArray()](https://www.mongodb.com/docs/manual/reference/method/cursor.toArray/" \l "mongodb-method-cursor.toArray) method loads into RAM all documents returned by the cursor; the [toArray()](https://www.mongodb.com/docs/manual/reference/method/cursor.toArray/" \l "mongodb-method-cursor.toArray) method exhausts the cursor.

Additionally, some [Drivers](https://www.mongodb.com/docs/drivers/" \t "https://www.mongodb.com/docs/manual/tutorial/iterate-a-cursor/_self) provide access to the documents by using an index on the cursor (i.e. cursor[index]). This is a shortcut for first calling the [toArray()](https://www.mongodb.com/docs/manual/reference/method/cursor.toArray/" \l "mongodb-method-cursor.toArray) method and then using an index on the resulting array.

Consider the following example:

|  |
| --- |
| var myCursor = db.users.find( { type: 2 } ); |
| var myDocument = myCursor[1]; |

The myCursor[1] is equivalent to the following example:

|  |
| --- |
| myCursor.toArray() [1]; |

# Aggregation Pipeline[IMG_256](https://www.mongodb.com/docs/manual/core/aggregation-pipeline/#aggregation-pipeline)

An aggregation pipeline consists of one or more [stages](https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/" \l "std-label-aggregation-pipeline-operator-reference) that process documents:

Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.

The documents that are output from a stage are passed to the next stage.

An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

Starting in MongoDB 4.2, you can update documents with an aggregation pipeline if you use the stages shown in [Updates with Aggregation Pipeline.](https://www.mongodb.com/docs/manual/tutorial/update-documents-with-aggregation-pipeline/)

## **NOTE**

Aggregation pipelines run with the [db.collection.aggregate()](https://www.mongodb.com/docs/manual/reference/method/db.collection.aggregate/" \l "mongodb-method-db.collection.aggregate) method do not modify documents in a collection, unless the pipeline contains a [$merge](https://www.mongodb.com/docs/manual/reference/operator/aggregation/merge/" \l "mongodb-pipeline-pipe.-merge) or [$out](https://www.mongodb.com/docs/manual/reference/operator/aggregation/out/" \l "mongodb-pipeline-pipe.-out) stage.

[You can](https://www.mongodb.com/docs/atlas/atlas-ui/agg-pipeline)[run aggregation pipelines in the UI](https://www.mongodb.com/docs/atlas/atlas-ui/agg-pipeline/" \t "https://www.mongodb.com/docs/manual/core/aggregation-pipeline/_self)[for deployments hosted in](https://www.mongodb.com/docs/atlas/atlas-ui/agg-pipeline)[MongoDB Atlas.](https://www.mongodb.com/docs/atlas?tck=docs_server" \t "https://www.mongodb.com/docs/manual/core/aggregation-pipeline/_self)

When you run aggregation pipelines on MongoDB Atlas deployments in the MongoDB Atlas UI, you can preview the results at each stage.

## Complete Aggregation Pipeline Examples[IMG_257](https://www.mongodb.com/docs/manual/core/aggregation-pipeline/#complete-aggregation-pipeline-examples)

This section shows aggregation pipeline examples that use the following pizza orders collection:

|  |
| --- |
| db.orders.insertMany( [ |
| { \_id: 0, name: **"Pepperoni"**, size: **"small"**, price: 19, |
| quantity: 10, date: ISODate( **"2021-03-13T08:14:30Z"** ) }, |
| { \_id: 1, name: **"Pepperoni"**, size: **"medium"**, price: 20, |
| quantity: 20, date : ISODate( **"2021-03-13T09:13:24Z"** ) }, |
| { \_id: 2, name: **"Pepperoni"**, size: **"large"**, price: 21, |
| quantity: 30, date : ISODate( **"2021-03-17T09:22:12Z"** ) }, |
| { \_id: 3, name: **"Cheese"**, size: **"small"**, price: 12, |
| quantity: 15, date : ISODate( **"2021-03-13T11:21:39.736Z"** ) }, |
| { \_id: 4, name: **"Cheese"**, size: **"medium"**, price: 13, |
| quantity:50, date : ISODate( **"2022-01-12T21:23:13.331Z"** ) }, |
| { \_id: 5, name: **"Cheese"**, size: **"large"**, price: 14, |
| quantity: 10, date : ISODate( **"2022-01-12T05:08:13Z"** ) }, |
| { \_id: 6, name: **"Vegan"**, size: **"small"**, price: 17, |
| quantity: 10, date : ISODate( **"2021-01-13T05:08:13Z"** ) }, |
| { \_id: 7, name: **"Vegan"**, size: **"medium"**, price: 18, |
| quantity: 10, date : ISODate( **"2021-01-13T05:10:13Z"** ) } |
| ] ) |

### **Calculate Total Order Quantity[IMG_258](https://www.mongodb.com/docs/manual/core/aggregation-pipeline/#calculate-total-order-quantity)**

The following aggregation pipeline example contains two [stages](https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/" \l "std-label-aggregation-pipeline-operator-reference) and returns the total order quantity of medium size pizzas grouped by pizza name:

|  |
| --- |
| db.orders.aggregate( [ |
|  |
| *// Stage 1: Filter pizza order documents by pizza size* |
| { |
| $match: { size: **"medium"** } |
| }, |
|  |
| *// Stage 2: Group remaining documents by pizza name and calculate total quantity* |
| { |
| $group: { \_id: **"$name"**, totalQuantity: { $sum: **"$quantity"** } } |
| } |
|  |
| ] ) |

The [$match](https://www.mongodb.com/docs/manual/reference/operator/aggregation/match/" \l "mongodb-pipeline-pipe.-match) stage:

Filters the pizza order documents to pizzas with a size of medium.

Passes the remaining documents to the [$group](https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/" \l "mongodb-pipeline-pipe.-group) stage.

The [$group](https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/" \l "mongodb-pipeline-pipe.-group) stage:

Groups the remaining documents by pizza name.

Uses [$sum](https://www.mongodb.com/docs/manual/reference/operator/aggregation/sum/" \l "mongodb-group-grp.-sum) to calculate the total order quantity for each pizza name. The total is stored in the totalQuantity field returned by the aggregation pipeline.

Example output:

|  |
| --- |
| [ |
| { \_id: **'Cheese'**, totalQuantity: 50 }, |
| { \_id: **'Vegan'**, totalQuantity: 10 }, |
| { \_id: **'Pepperoni'**, totalQuantity: 20 } |
| ] |

### **Calculate Total Order Value and Average Order Quantity[IMG_259](https://www.mongodb.com/docs/manual/core/aggregation-pipeline/#calculate-total-order-value-and-average-order-quantity)**

The following example calculates the total pizza order value and average order quantity between two dates:

|  |
| --- |
| db.orders.aggregate( [ |
|  |
| *// Stage 1: Filter pizza order documents by date range* |
| { |
| $match: |
| { |
| **"date"**: { $gte: new ISODate( **"2020-01-30"** ), $lt: new ISODate( **"2022-01-30"** ) } |
| } |
| }, |
|  |
| *// Stage 2: Group remaining documents by date and calculate results* |
| { |
| $group: |
| { |
| \_id: { $dateToString: { format: **"%Y-%m-%d"**, date: **"$date"** } }, |
| totalOrderValue: { $sum: { $multiply: [ **"$price"**, **"$quantity"** ] } }, |
| averageOrderQuantity: { $avg: **"$quantity"** } |
| } |
| }, |
|  |
| *// Stage 3: Sort documents by totalOrderValue in descending order* |
| { |
| $sort: { totalOrderValue: -1 } |
| } |
|  |
| ] ) |

The [$match](https://www.mongodb.com/docs/manual/reference/operator/aggregation/match/" \l "mongodb-pipeline-pipe.-match) stage:

Filters the pizza order documents to those in a date range specified using [$gte](https://www.mongodb.com/docs/manual/reference/operator/aggregation/gte/" \l "mongodb-expression-exp.-gte) and [$lt.](https://www.mongodb.com/docs/manual/reference/operator/aggregation/lt/" \l "mongodb-expression-exp.-lt)

Passes the remaining documents to the [$group](https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/" \l "mongodb-pipeline-pipe.-group) stage.

The [$group](https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/" \l "mongodb-pipeline-pipe.-group) stage:

Groups the documents by date using [$dateToString.](https://www.mongodb.com/docs/manual/reference/operator/aggregation/dateToString/" \l "mongodb-expression-exp.-dateToString)

For each group, calculates:

Total order value using [$sum](https://www.mongodb.com/docs/manual/reference/operator/aggregation/sum/" \l "mongodb-group-grp.-sum) and [$multiply.](https://www.mongodb.com/docs/manual/reference/operator/aggregation/multiply/" \l "mongodb-expression-exp.-multiply)

Average order quantity using [$avg.](https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/" \l "mongodb-group-grp.-avg)

Passes the grouped documents to the [$sort](https://www.mongodb.com/docs/manual/reference/operator/aggregation/sort/" \l "mongodb-pipeline-pipe.-sort) stage.

The [$sort](https://www.mongodb.com/docs/manual/reference/operator/aggregation/sort/" \l "mongodb-pipeline-pipe.-sort) stage:

Sorts the documents by the total order value for each group in descending order (-1).

Returns the sorted documents.

Example output:

|  |
| --- |
| [ |
| { \_id: **'2022-01-12'**, totalOrderValue: 790, averageOrderQuantity: 30 }, |
| { \_id: **'2021-03-13'**, totalOrderValue: 770, averageOrderQuantity: 15 }, |
| { \_id: **'2021-03-17'**, totalOrderValue: 630, averageOrderQuantity: 30 }, |
| { \_id: **'2021-01-13'**, totalOrderValue: 350, averageOrderQuantity: 10 } |
| ] |

### **See also:**

[Aggregation with User Preference Data](https://www.mongodb.com/docs/manual/tutorial/aggregation-with-user-preference-data/)

[Aggregation with the Zip Code Data Set](https://www.mongodb.com/docs/manual/tutorial/aggregation-zip-code-data-set/)

[Updates with Aggregation Pipeline](https://www.mongodb.com/docs/manual/tutorial/update-documents-with-aggregation-pipeline/)

