Install MongoDB Community Edition on macOS

NOTE

MongoDB Atlas

MongoDB Atlas is a hosted MongoDB service option in the cloud which requires no installation overhead and offers a free tier to get started.

Overview

Use this tutorial to install MongoDB 6.0 Community Edition on macOS using the third-party Homebrew package manager.

Starting with MongoDB 4.4.1, installing MongoDB via Homebrew also installs the MongoDB Database Tools. See Using the MongoDB Database Tools for more information.

MongoDB Version

This tutorial installs MongoDB 6.0 Community Edition. To install a different version of MongoDB Community, use the version drop-down menu in the upper-left corner of this page to select the documentation for that version.

Considerations

Platform Support

NOTE

EOL Notice

- MongoDB 5.0 Community Edition removes support for macOS 10.13

MongoDB 6.0 Community Edition supports macOS 10.14 or later.

See Platform Support for more information.

Production Notes

Before deploying MongoDB in a production environment, consider the Production Notes document which offers performance considerations and configuration recommendations for production MongoDB deployments.

Install MongoDB Community Edition

Prerequisites

Ensure your system meets each of the following prerequisites. You only need to perform each prerequisite step once on your system. If you have already performed the prerequisite steps as part of an earlier MongoDB installation using Homebrew, you can skip to the installation procedure.

Install Xcode Command-Line Tools

Homebrew requires the Xcode command-line tools from Apple's Xcode.

- Install the Xcode command-line tools by running the following command in your macOS Terminal:

  ```
  xcode-select --install
  ```

Install Homebrew

macOS does not include the Homebrew `brew` package by default.

- Install `brew` using the official Homebrew installation instructions.

Installing MongoDB 6.0 Community Edition

Follow these steps to install MongoDB Community Edition using Homebrew's `brew` package manager. Be sure that you have followed the installation prerequisites above before proceeding.

1. Tap the MongoDB Homebrew Tap to download the official Homebrew formula for MongoDB and the Database Tools, by running the following command in your macOS Terminal:

   brew tap mongodb/brew

2. If you have already done this for a previous installation of MongoDB, you can skip this step.
3. To update Homebrew and all existing formulae:

   brew update

4. To install MongoDB, run the following command in your macOS Terminal application:

   brew install mongodb-community@6.0

TIP

Alternatively, you can specify a previous version of MongoDB if desired. You can also maintain multiple versions of MongoDB side by side in this manner.

TIP

If you have previously installed an older version of the formula, you may encounter a ChecksumMismatchError. To resolve, see Troubleshooting ChecksumMismatchError.

The installation includes the following binaries:

- The `mongod` server
- The `mongos` sharded cluster query router
- The MongoDB Shell, `mongosh`

In addition, the installation creates the following files and directories at the location specified below, depending on your Apple hardware:

	Intel Processor	Apple Silicon Processor
configuration file	`/usr/local/etc/mongod.conf`	`/opt/homebrew/etc/mongod.conf`
`log directory`	`/usr/local/var/log/mongodb`	`/opt/homebrew/var/log/mongodb`
`data directory`	`/usr/local/var/mongodb`	`/opt/homebrew/var/mongodb`

See Apple's documentation for the current list of Apple hardware using the Apple Silicon processor. You can also run the following command to check where `brew` has installed these files and directories:

```
brew --prefix
```

Starting with MongoDB 4.4.1, the installation also includes the MongoDB Database Tools. See Using the MongoDB Database Tools for more information.

Run MongoDB Community Edition

Follow these steps to run MongoDB Community Edition. These instructions assume that you are using the default settings.

You can run MongoDB as a macOS service using `brew`, or you can run MongoDB manually as a background process. It is recommended to run MongoDB as a macOS service, as doing so sets the correct system `ulimit` values automatically (see ulimit settings for more information).

- To run MongoDB (i.e. the `mongod` process) **as a macOS service**, run:

brew services start mongodb-community@6.0

- To stop a `mongod` running as a macOS service, use the following command as needed:
  brew services stop mongodb-community@6.0

- To run MongoDB (i.e. the `mongod` process) **manually as a background process**, run:
  - For macOS running Intel processors:
    mongod --config /usr/local/etc/mongod.conf --fork
  - For macOS running on Apple Silicon processors:
    mongod --config /opt/homebrew/etc/mongod.conf --fork

- To stop a `mongod` running as a background process, connect to the `mongod` using mongosh, and issue the `shutdown` command as needed.

Both methods use the `mongod.conf` file created during the install. You can add your own MongoDB configuration options to this file as well.

NOTE

macOS Prevents mongod From Opening

macOS may prevent `mongod` from running after installation. If you receive a security error when starting `mongod` indicating that the developer could not be identified or verified, do the following to grant `mongod` access to run:

- Open *System Preferences*
- Select the *Security and Privacy* pane.
- Under the *General* tab, click the button to the right of the message about `mongod`, labelled either **Open Anyway** or **Allow Anyway** depending on your version of macOS.

To verify that MongoDB is running, perform one of the following:

- If you started MongoDB **as a macOS service**:

  brew services list

- You should see the service `mongodb-community` listed as `started`.
- If you started MongoDB **manually as a background process**:

  ps aux | grep -v grep | grep mongod

- You should see your `mongod` process in the output.

You can also view the log file to see the current status of your `mongod` process: `/usr/local/var/log/mongodb/mongo.log`.

Connect and Use MongoDB

To begin using MongoDB, connect `mongosh` to the running instance. From a new terminal, issue the following:

```
mongosh
```

NOTE

macOS Prevents mongosh From Opening

macOS may prevent `mongosh` from running after installation. If you receive a security error when starting `mongosh` indicating that the developer could not be identified or verified, do the following to grant `mongosh` access to run:

- Open *System Preferences*
- Select the *Security and Privacy* pane.
- Under the *General* tab, click the button to the right of the message about `mongosh`, labelled either **Open Anyway** or **Allow Anyway** depending on your version of macOS.

For information on CRUD (Create,Read,Update,Delete) operations, see:

- Insert Documents
- Query Documents
- Update Documents
- Delete Documents

Using the MongoDB Database Tools

Starting in MongoDB 4.4.1, installing MongoDB via `brew` also installs the MongoDB Database Tools.

The MongoDB Database Tools are a collection of command-line utilities for working with a MongoDB deployment, including data backup and import/export tools like `mongoimport` and `mongodump` as well as monitoring tools like `mongotop`.

Once you have installed the MongoDB Server in the steps above, the Database Tools are available directly from the command line in your macOS Terminal application. For example you could run `mongotop` against your running MongoDB instance by invoking it in your macOS Terminal like so:

```
mongotop
```

It should start up, connect to your running `mongod`, and start reporting usage statistics.

See the MongoDB Database Tools Documentation for usage information for each of the Database Tools.

Additional Information

Localhost Binding by Default

By default, MongoDB launches with `bindIp` set to `127.0.0.1`, which binds to the localhost network interface. This means that the `mongod` can only accept connections from clients that

are running on the same machine. Remote clients will not be able to connect to the `mongod`, and the `mongod` will not be able to initialize a replica set unless this value is set to a valid network interface.

This value can be configured either:

- in the MongoDB configuration file with `bindIp`, or
- via the command-line argument `--bind_ip`

WARNING

Before binding to a non-localhost (e.g. publicly accessible) IP address, ensure you have secured your cluster from unauthorized access. For a complete list of security recommendations, see Security Checklist. At minimum, consider enabling authentication and hardening network infrastructure.

For more information on configuring `bindIp`, see IP Binding.

Troubleshooting ChecksumMismatchError

If you have previously installed an older version of the formula, you may encounter a `ChecksumMismatchError` resembling the following:

```
Error: An exception occurred within a child process:
ChecksumMismatchError: SHA256 mismatch
Expected: c7214ee7bda3cf9566e8776a8978706d9827c1b09017e17b66a5a4e0c0731e1f
Actual: 6aa2e0c348e8abeec7931dced1f85d4bb161ef209c6af317fe530ea11bbac8f0
Archive: /Users/kay/Library/Caches/Homebrew/downloads/a6696157a9852f392ec6323b4bb697b86312f0c34
To retry an incomplete download, remove the file above.
```

To fix:

1. Remove the downloaded `.tgz` archive.
2. Retap the formula.

   ```
   brew untap mongodb/brew && brew tap mongodb/brew
   ```

3. Retry the install.

   brew install mongodb-community@6.0

Manage `mongod` Processes

MongoDB runs as a standard program. You can start MongoDB from a command line by issuing the `mongod` command and specifying options. For a list of options, see the `mongod` reference. MongoDB can also run as a Windows service. For details, see Start MongoDB Community Edition as a Windows Service. To install MongoDB, see Install MongoDB.

The following examples assume the directory containing the `mongod` process is in your system paths. The `mongod` process is the primary database process that runs on an individual server. `mongos` provides a coherent MongoDB interface equivalent to a `mongod` from the perspective of a client. The `mongosh` binary provides the administrative shell.

This document discusses the `mongod` process; however, some portions of this document may be applicable to `mongos` instances.

Start `mongod` Processes

By default, MongoDB listens for connections from clients on port `27017`, and stores data in the `/data/db` directory.

On Windows, this path is on the drive from which you start MongoDB. For example, if you do not specify a `--dbpath`, starting a MongoDB server on the `C:\` drive stores all data files in `C:\data\db`.

To start MongoDB using all defaults, issue the following command at the system shell:

```
mongod
```

Specify a Data Directory

If you want `mongod` to store data files at a path *other than* `/data/db` you can specify a `dbPath`. The `dbPath` must exist before you start `mongod`. If it does not exist, create the directory and the permissions so that `mongod` can read and write data to this path. For more information on permissions, see the security operations documentation.

To specify a `dbPath` for `mongod` to use as a data directory, use the `--dbpath` option. The following invocation will start a `mongod` instance and store data in the `/srv/mongodb` path

```
mongod --dbpath /srv/mongodb/
```

Specify a TCP Port

Only a single process can listen for connections on a network interface at a time. If you run multiple `mongod` processes on a single machine, or have other processes that must use this port, you must assign each a different port to listen on for client connections.

To specify a port to `mongod`, use the `--port` option on the command line. The following command starts `mongod` listening on port `12345`:

```
mongod --port 12345
```

Use the default port number when possible, to avoid confusion.

Start `mongod` as a Daemon

To run a `mongod` process as a daemon (i.e. `fork`), *and* write its output to a log file, use the `--fork` and `--logpath` options. You must create the log directory; however, `mongod` will create the log file if it does not exist.

The following command starts `mongod` as a daemon and records log output to `/var/log/mongodb/mongod.log`.

```
mongod --fork --logpath /var/log/mongodb/mongod.log
```

Additional Configuration Options

For an overview of common configurations and deployments for common use cases, see [Run-time Database Configuration.]

Stop `mongod` Processes

In a clean shutdown a `mongod` completes all pending operations, flushes all data to data files, and closes all data files. Other shutdowns are *unclean* and can compromise the validity of the data files.

To ensure a clean shutdown, always shutdown `mongod` instances using one of the following methods:

Use `shutdownServer()`

Shut down the `mongod` from `mongosh` using the `db.shutdownServer()` method as follows:

```
use admin
db.shutdownServer()
```

Calling the same method from a init script accomplishes the same result.

For systems with `authorization` enabled, users may only issue `db.shutdownServer()` when authenticated to the `admin` database or via the localhost interface on systems without authentication enabled.

Use `--shutdown`

Supported on Linux only. From the command line, shut down the `mongod` using the `--shutdown` option:

```
mongod --shutdown
```

Use `CTRL-C`

When running the `mongod` instance in interactive mode (i.e. without `--fork`), issue `Control-C` to perform a clean shutdown.

Use `kill`

Supported on Linux and macOS only. From the command line, shut down a specific `mongod` instance using one of the following commands:

```
kill <mongod process ID>
kill -2 <mongod process ID>
```

`SIGTERM` and Replica Sets

If a replica set primary receives a `SIGTERM`, the primary attempts to step down before shutting down.

- If the step down succeeds, the instance does not vote in the ensuing election of the new primary, and continues its shutdown.
- If the step down fails, the instance continues its shutdown.

`SIGKILL`

WARNING

Never use `kill -9` (i.e. `SIGKILL`) to terminate a mongod instance.

Troubleshoot `mongod` Processes

Generate a Backtrace

Starting in MongoDB 4.4 running on Linux:

- When the `mongod` and `mongos` processes receive a `SIGUSR2` signal, backtrace details are added to the logs for each process thread.
- Backtrace details show the function calls for the process, which can be used for diagnostics and provided to MongoDB Support if required.

The backtrace functionality is available for these architectures:

- `x86_64`
- `arm64` (starting in MongoDB 4.4.15, 5.0.10, and 6.0)

To issue a `SIGUSR2` signal to a running `mongod` process, use the following command:

```
kill -SIGUSR2 <mongod process ID>
```

The resulting backtrace data is written to the `mongod` logfile as configured with `--logpath`.

Stop a Replica Set

Procedure

If the `mongod` is the primary in a replica set, the shutdown process for this `mongod` instance has the following steps:

1. Check how up-to-date the secondaries are.
2. If no secondary is within 10 seconds of the primary, `mongod` will return a message that it will not shut down. You can pass the `shutdown` command a `timeoutSecs` argument to wait for a secondary to catch up.
3. If there is a secondary within 10 seconds of the primary, the primary will step down and wait for the secondary to catch up.
4. After 60 seconds or once the secondary has caught up, the primary will shut down.

Force Replica Set Shutdown

If there is no up-to-date secondary and you want the primary to shut down, issue the `shutdown` command with the `force` argument, as in the following `mongosh` operation:

```
db.adminCommand({shutdown : 1, force : true})
```

To keep checking the secondaries for a specified number of seconds if none are immediately up-to-date, issue `shutdown` with the `timeoutSecs` argument. MongoDB will keep checking the secondaries for the specified number of seconds if none are immediately up-to-date. If any of the secondaries catch up within the allotted time, the primary will shut down. If no secondaries catch up, it will not shut down.

The following command issues `shutdown` with `timeoutSecs` set to `5`:

```
db.adminCommand({shutdown : 1, timeoutSecs : 5})
```

Alternately you can use the `timeoutSecs` argument with the `db.shutdownServer()` method:

```
db.shutdownServer({timeoutSecs : 5})
```

Insert Documents

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the examples on this page.

This page provides examples of insert operations in MongoDB.

NOTE

Creating a Collection

If the collection does not currently exist, insert operations will create the collection.

Insert a Single Document

`db.collection.insertOne()` inserts a *single* document into a collection.

The following example inserts a new document into the `inventory` collection. If the document does not specify an `_id` field, MongoDB adds the `_id` field with an ObjectId value to the new document. See Insert Behavior.

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

`insertOne()` returns a document that includes the newly inserted document's `_id` field value. For an example of a return document, see db.collection.insertOne() reference.

To retrieve the document that you just inserted, query the collection:

```
db.inventory.find( { item: "canvas" } )
```
MongoDB Shell

Insert Multiple Documents

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the examples on this page.

`db.collection.insertMany()` can insert *multiple* documents into a collection. Pass an array of documents to the method.

The following example inserts three new documents into the `inventory` collection. If the documents do not specify an `_id` field, MongoDB adds the `_id` field with an ObjectId value to each document. See Insert Behavior.

```
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" } }
])
```
MongoDB Shell

`insertMany()` returns a document that includes the newly inserted documents `_id` field values. See the reference for an example.

To retrieve the inserted documents, query the collection:

```
db.inventory.find( {} )
```
MongoDB Shell

Insert Behavior

Collection Creation

If the collection does not currently exist, insert operations will create the collection.

`_id` Field

In MongoDB, each document stored in a collection requires a unique _id field that acts as a primary key. If an inserted document omits the `_id` field, the MongoDB driver automatically generates an ObjectId for the `_id` field.

This also applies to documents inserted through update operations with upsert: true.

Atomicity

All write operations in MongoDB are atomic on the level of a single document. For more information on MongoDB and atomicity, see Atomicity and Transactions

Write Acknowledgement

With write concerns, you can specify the level of acknowledgement requested from MongoDB for write operations. For details, see Write Concern.

TIP

See also:

- `db.collection.insertOne()`
- `db.collection.insertMany()`
- Additional Methods for Inserts

Query Documents

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the following examples.

This page provides examples of query operations using the `db.collection.find()` method in `mongosh`.

The examples on this page use the `inventory` collection. Connect to a test database in your MongoDB instance then create the `inventory` collection:

```
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" }
]);
```
MongoDB Shell

Select All Documents in a Collection

To select all documents in the collection, pass an empty document as the query filter parameter to the find method. The query filter parameter determines the select criteria:

```
db.inventory.find( {} )
```
MongoDB Shell

This operation uses a filter predicate of `{}`, which corresponds to the following SQL statement:

```
SELECT * FROM inventory
```

For more information on the syntax of the method, see `find()`.

Specify Equality Condition

To specify equality conditions, use `<field>:<value>` expressions in the query filter document:

```
{ <field1>: <value1>, ... }
```
MongoDB Shell

The following example selects from the `inventory` collection all documents where the `status` equals `"D"`:

```
db.inventory.find( { status: "D" } )
```
MongoDB Shell

This operation uses a filter predicate of `{ status: "D" }`, which corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "D"
```

NOTE

The MongoDB Compass query bar autocompletes the current query based on the keys in your collection's documents, including keys in embedded sub-documents.

Specify Conditions Using Query Operators

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```
MongoDB Shell

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

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```
MongoDB Shell

NOTE

Although you can express this query using the `$or` operator, use the `$in` operator rather than the `$or` operator when performing equality checks on the same field.

The operation uses a filter predicate of `{ status: { $in: [ "A", "D" ] } }`, which corresponds to the following SQL statement:

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

Refer to the Query and Projection Operators document for the complete list of MongoDB query operators.

Specify `AND` Conditions

A compound query can specify conditions for more than one field in the collection's documents. Implicitly, a logical `AND` conjunction connects the clauses of a compound query so that the query selects the documents in the collection that match all the conditions.

The following example retrieves all documents in the `inventory` collection where the `status` equals `"A"` **and** `qty` is less than (`$lt`) `30`:

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```
MongoDB Shell

The operation uses a filter predicate of `{ status: "A", qty: { $lt: 30 } }`, which corresponds to the following SQL statement:

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

See comparison operators for other MongoDB comparison operators.

Specify `OR` Conditions

Using the `$or` operator, you can specify a compound query that joins each clause with a logical `OR` conjunction so that the query selects the documents in the collection that match at least one condition.

The following example retrieves all documents in the collection where the `status` equals `"A"` **or** `qty` is less than (`$lt`) `30`:

```
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```
MongoDB Shell

The operation uses a filter predicate of `{ $or: [ { status: 'A' }, { qty: { $lt: 30 } } ] }`, which corresponds to the following SQL statement:

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

NOTE

Queries that use comparison operators are subject to Type Bracketing.

Specify `AND` as well as `OR` Conditions

In the following example, the compound query document selects all documents in the collection where the `status` equals `"A"` **and** *either* `qty` is less than (`$lt`) `30` *or* `item` starts with the character `p`:

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

MongoDB Shell

The operation uses a filter predicate of:

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

which corresponds to the following SQL statement:

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

NOTE

MongoDB supports regular expressions `$regex` queries to perform string pattern matches.

Additional Query Tutorials

For additional query examples, see:

- Query on Embedded/Nested Documents
- Query an Array
- Query an Array of Embedded Documents
- Project Fields to Return from Query
- Query for Null or Missing Fields

Behavior

Cursor

The `db.collection.find()` method returns a cursor to the matching documents.

Read Isolation

For reads to replica sets and replica set shards, read concern allows clients to choose a level of isolation for their reads. For more information, see Read Concern.

Query Result Format

When you run a find operation with a MongoDB driver or `mongosh`, the command returns a cursor that manages query results. The query results are not returned as an array of documents.

To learn how to iterate through documents in a cursor, refer to your driver's documentation. If you are using `mongosh`, see Iterate a Cursor in `mongosh`.

Update Documents

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the following examples.

This page uses the following `mongosh` methods:

- `db.collection.updateOne(<filter>, <update>, <options>)`
- `db.collection.updateMany(<filter>, <update>, <options>)`
- `db.collection.replaceOne(<filter>, <update>, <options>)`

The examples on this page use the `inventory` collection. Connect to a test database in your MongoDB instance then create the `inventory` collection:

```
db.inventory.insertMany( [
{ item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },
{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
{ item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },
{ item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },
{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
{ 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" },
{ item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
{ item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status: "A" }
] );
```
MongoDB Shell

Update Documents in a Collection

To update a document, MongoDB provides update operators, such as `$set`, to modify field values.

To use the update operators, pass to the update methods an update document of the form:

```
{
<update operator>: { <field1>: <value1>, ... },
<update operator>: { <field2>: <value2>, ... },
...
}
```
MongoDB Shell

Some update operators, such as `$set`, will create the field if the field does not exist. See the individual update operator reference for details.

NOTE

Starting in MongoDB 4.2, MongoDB can accept an aggregation pipeline to specify the modifications to make instead of an update document. See the method reference page for details.

Update a Single Document

The following example uses the `db.collection.updateOne()` method on the `inventory` collection to update the *first* document where `item` equals `"paper"`:

```
db.inventory.updateOne(
{ item: "paper" },
{
$set: { "size.uom": "cm", status: "P" },
$currentDate: { lastModified: true }
}
)
```

MongoDB Shell

The update operation:

- uses the `$set` operator to update the value of the `size.uom` field to `"cm"` and the value of the `status` field to `"P"`,
- uses the `$currentDate` operator to update the value of the `lastModified` field to the current date. If `lastModified` field does not exist, `$currentDate` will create the field. See `$currentDate` for details.

```
db.inventory.updateMany(
{ "qty": { $lt: 50 } },
{
$set: { "size.uom": "in", status: "P" },
$currentDate: { lastModified: true }
}
)
```

MongoDB Shell

The update operation:

- uses the `$set` operator to update the value of the `size.uom` field to `"in"` and the value of the `status` field to `"P"`,
- uses the `$currentDate` operator to update the value of the `lastModified` field to the current date. If `lastModified` field does not exist, `$currentDate` will create the field. See `$currentDate` for details.

```
db.inventory.replaceOne(
{ item: "paper" },
{ item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
)
```
MongoDB Shell

Behavior

Atomicity

All write operations in MongoDB are atomic on the level of a single document. For more information on MongoDB and atomicity, see Atomicity and Transactions.

`_id` Field

Once set, you cannot update the value of the `_id` field nor can you replace an existing document with a replacement document that has a different `_id` field value.

Field Order

For write operations, MongoDB preserves the order of the document fields *except* for the following cases:

- The `_id` field is always the first field in the document.
- Updates that include `renaming` of field names may result in the reordering of fields in the document.

Write Acknowledgement

With write concerns, you can specify the level of acknowledgement requested from MongoDB for write operations. For details, see Write Concern.

TIP

See also:

- Updates with Aggregation Pipeline
- `db.collection.updateOne()`
- `db.collection.updateMany()`
- `db.collection.replaceOne()`
- Additional Methods

Delete Documents

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the following examples.

This page uses the following `mongosh` methods:

- `db.collection.deleteMany()`

- `db.collection.deleteOne()`

The examples on this page use the `inventory` collection. To populate the `inventory` collection, run the following:

```
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: "P" },
{ 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" },
] );
```
MongoDB Shell

Delete All Documents

To delete all documents from a collection, pass an empty filter document `{}` to the `db.collection.deleteMany()` method.

The following example deletes *all* documents from the `inventory` collection:

```
db.inventory.deleteMany({})
```
MongoDB Shell

The method returns a document with the status of the operation. For more information and examples, see `deleteMany()`.

Delete All Documents that Match a Condition

You can specify criteria, or filters, that identify the documents to delete. The filters use the same syntax as read operations.

To specify equality conditions, use `<field>:<value>` expressions in the query filter document:

```
{ <field1>: <value1>, ... }
```
MongoDB Shell

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```
MongoDB Shell

To delete all documents that match a deletion criteria, pass a filter parameter to the `deleteMany()` method.

The following example removes all documents from the `inventory` collection where the `status` field equals `"A"`:

```
db.inventory.deleteMany({ status : "A" })
```
MongoDB Shell

The method returns a document with the status of the operation. For more information and examples, see `deleteMany()`.

Delete Only One Document that Matches a Condition

To delete at most a single document that matches a specified filter (even though multiple documents may match the specified filter) use the `db.collection.deleteOne()` method.

The following example deletes the *first* document where `status` is `"D"`:

```
db.inventory.deleteOne( { status: "D" } )
```
MongoDB Shell

Delete Behavior

Indexes

Delete operations do not drop indexes, even if deleting all documents from a collection.

Atomicity

All write operations in MongoDB are atomic on the level of a single document. For more information on MongoDB and atomicity, see Atomicity and Transactions.

Write Acknowledgement

With write concerns, you can specify the level of acknowledgement requested from MongoDB for write operations. For details, see Write Concern.

TIP

See also:

- `db.collection.deleteMany()`
- `db.collection.deleteOne()`
- Additional Methods