# Update Documents

This page provides examples of how to update documents in MongoDB. The examples use the following methods in the `mongo` shell:

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

The examples on this page use the `inventory` collection. To create and/or populate the `inventory` collection, run the following in the `mongo` shell:

```
db.inventory.insert( [
   { 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" }
]);
```

You can run the operation in the web shell below:

MongoDB Web Shell

Click to connect

Full

Reset

Clear

## 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>, ... },
  ...
}
```

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

### Update Documents via `db.collection.updateOne()`

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 }
   }
)
```

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.

For more information and examples, see `db.collection.updateOne()`.

### Update Documents via `db.collection.updateMany()`

*New in version 3.2.*

The following example uses the `db.collection.updateMany()` method on the `inventory` collection to update all documents where `qty` is less than `50`.

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

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.

For more information and examples, see `db.collection.updateMany()`.

### Update Documents via `db.collection.update`

The following example uses the `db.collection.update()` method on the `inventory` collection to update the *first* document where the status equals `"P"`.

```
db.inventory.update(
   { "status": "P" } ,
   {
     $set: { status: "D" },
     $currentDate: { lastModified: true }
   }
)
```

The update operation:

- uses the `$set` operator to update the value of the `status` field to `"D"`,
- uses the `$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.

To update multiple documents using the `db.collection.update()`, include the multi: true option:

```
db.inventory.update(
   { "status": "P" },
   {
     $set: { status: "D" },
     $currentDate: { lastModified: true }
   },
   { multi: true }
)
```

## Replace the Document

To replace the entire content of a document except for the `_id` field, pass an entirely new document as the second argument to `db.collection.replaceOne()` or `db.collection.update()`. When replacing a document, the replacement document must consist of only `<field> : <value>` pairs; i.e. do not include update operators expressions.

The replacement document can have different fields from the original document. In the replacement document, you can omit the `_id` field since the `_id` field is immutable; however, if you do include the `_id` field, it must have the same value as the current value.

### Replace a Document via `db.collection.replaceOne`

The following example uses the `db.collection.replaceOne()` method on the `inventory` collection to replace the *first* document matches the filter `item` equals `"paper"`:

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

### Replace a Document via `db.collection.update`

The following example uses the `db.collection.update()` method on the `inventory` collection to replace the *first* document that matches the filter `item` equals `"postcard"` with the new document:

```
db.inventory.update(
   { item: "postcard" },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

## 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.

### Document Size

When performing update operations that increase the document size beyond the allocated space for that document, the update operation relocates the document on disk.

### Field Order

MongoDB preserves the order of the document fields following write operations *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.

*Changed in version 2.6:* Starting in version 2.6, MongoDB actively attempts to preserve the field order in a document. Before version 2.6, MongoDB did not actively preserve the order of the fields in a document.

### `Upsert` Option

If `db.collection.update()`, `db.collection.updateOne()`, `db.collection.updateMany()`, or `db.collection.replaceOne()` includes `upsert : true` **and** no documents match the specified filter, then the operation creates a new document and inserts it. If there are matching documents, then the operation modifies or replaces the matching document or documents.

For details on the new document created, see the individual reference pages for the methods.

### Write Acknowledgement

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

[1] You can use the `DBQuery.shellBatchSize` to change the number of iteration from the default value `20`. See Working with the mongo Shell for more information.

**SEE ALSO:**

- `db.collection.update()`

- `db.collection.updateOne()`
- `db.collection.updateMany()`
- `db.collection.replaceOne()`
- Additional Methods