

Chapter 3.

CRUD Operations

1

References

- MongoDB The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition
- <https://docs.mongodb.com/>
- <https://www.mongodb.com/docs/manual/>

Learning objectives

- Query, insert, update, delete document

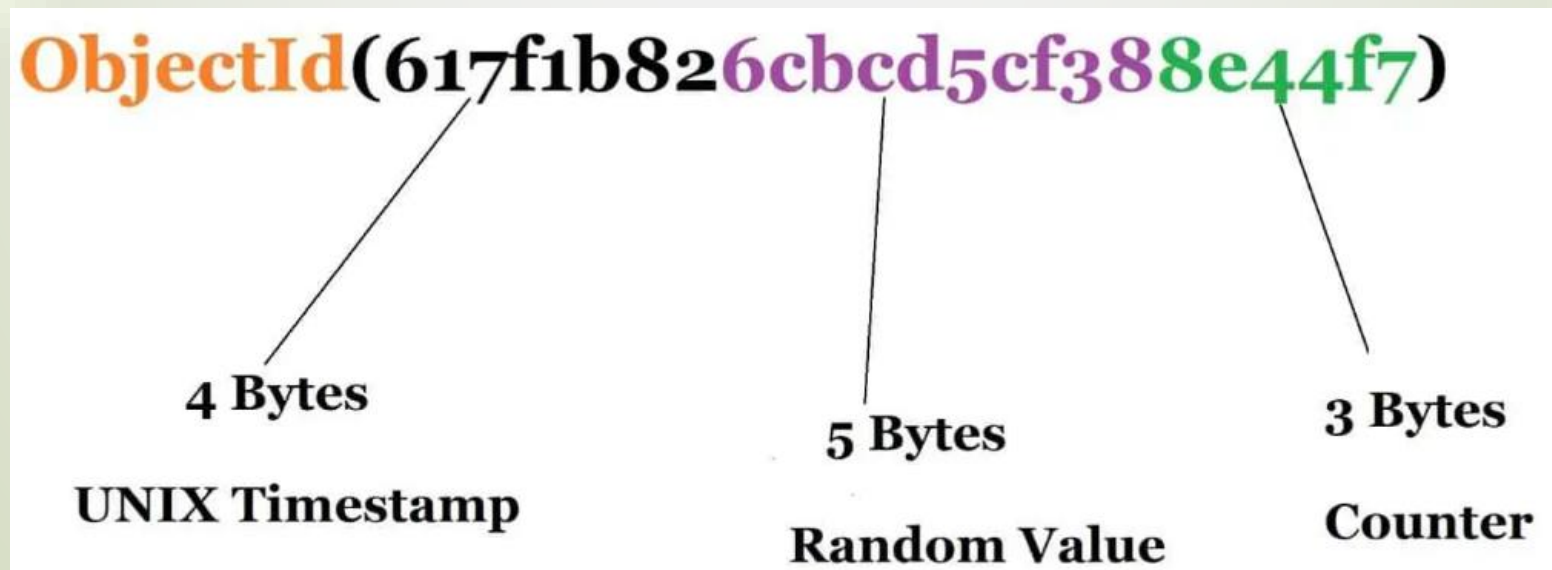
Contents

1. Insert document
2. Query document
3. Update document
4. Delete document

1. Insert document

_id, ObjectId()

- 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.



1. Insert documents

| Method | Description |
|-----------------------------------|--|
| db.collection.insert() | Inserts a <u>single</u> document or <u>multiple</u> documents into a collection. |
| db.collection.insertOne() | Inserts a single document into a collection. |
| db.collection.insertMany() | Inserts multiple documents into a collection. |

1. Insert documents

Insert() method

```
db.collection.insert(  
  <[documents]>,  
  {  
    writeConcern: <document>,  
    ordered: <boolean>    //only UpdateMany command supported  
  }  
)
```

| Parameter | Type | Description |
|---------------------------|----------|---|
| <[documents]> | document | <i>document</i> or <i>[array of documents]</i> to insert into the collection. |
| <code>writeConcern</code> | document | <u>Optional</u> . A document expressing the write concern. Omit to use the default write concern. |
| <code>ordered</code> | boolean | <u>Optional</u> . A boolean specifying whether the mongod instance should perform an ordered or unordered insert. Defaults to true. |

1. Insert documents

Example

- Insert a document without specifying an `_id` field:

```
db.products.insertOne( { item: "card", qty: 15 } )
```

→ mongod creates and adds the `_id` field.

- Insert a document specifying an `_id` field:

```
db.products.insertOne( { _id: 10, item: "box", qty: 20 } )
```

- Insert several document:

```
db.products.insertMany( [ { _id: 10, item: "card", qty: 15, type: "no.2" },  
                           { item: "envelope", qty: 20 },  
                           { item: "stamps" , qty: 30 }  
                        ] )
```


1. Insert documents

Example

- Try to execute the commands below, then make your conclusion:

```
db.products.insert( [{id : 905, item: "tape"}, {id : 905, item: "glue"}] )
```

```
db.products.insert( [{ _id: 905, item: "tape", qty: 20}, { _id: 905, item: "bubble  
wrap", qty: 30}, { _id: 906, item: "bubble wrap", qty: 30}] )
```

```
db.products.insert( [{ _id: 805, item: "tape", qty: 20}, { _id: 806, item: "bubble  
wrap", qty: 30}, { _id: 807, item: " medium box", qty: 30}],  
{ ordered: false } )
```

```
db.products.insert( [{ _id: 808, item: "tape", qty: 20}, { _id: 809, item: "bubble  
wrap", qty: 30}] )
```

1. Insert documents

Create a field date

- `new Date()`: Returns a date and time as a Date object.
- `ISODate()`: Returns a date and time as a Date object.
- `Date()`: Returns a date and time as a string.

```
db.dates.insertMany([
  { name: "Created with `new Date()`",
    date: new Date(),
  },
  { name: "Created with `ISODate()`",
    date: ISODate(),
  },
  { name: "Created with `Date()`",
    date: Date(),
  }
])
```

```
db.dates.insertMany([
  { name: "Future date",
    date: ISODate("2040-10-28T23:58:18Z"),
  },
  { name: "Past date",
    date: new Date("1852-01-15T11:25"),
  }
])
```

Contents

1. Insert document
- 2. Query document**
3. Update document
4. Delete document

2. Query document find() method

- find() method selects documents in a collection or view and returns a cursor to the selected documents
- Syntax:

db.<collectionname>.**find**(<query>, <projection>)

| Parameter | Type | Description |
|-------------------|----------|--|
| query | document | Optional. Specifies selection filter using <u>query operators</u> . To return all documents in a collection, omit this parameter or pass an empty document ({}). |
| projection | document | Optional. Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter |

2. Query document find() method example

- The examples in this section use documents from the bios collection where the documents generally have the form:

```
{
  "_id" : value,
  "name" : { "first" : string, "last" : string },
  "birth" : ISODate,
  "death" : ISODate,
  "contribs" : [ string, ... ],
  "awards" : [
    { "award" : string, "year": number, "by": string }
    ...
  ]
}
```

2. Query document

find() method example

- Find all documents in a collection:
`db.bios.find()`
- Find all documents in the bios collection where `_id` equals 5:
`db.bios.find({ _id: 5 })`
- Find all documents in the bios collection where the field last in the name embedded document equals "Hopper":
`db.bios.find({ "name.last": "Hopper" })`

To access fields in an **embedded document**, use dot notation ("`<embedded document>.<field>`").

2. Query document find() method example

- To return all documents in the bios collection where the embedded document name is **exactly** {first: "Yukihiko", last: "Matsumoto"}, **including the order**:

```
db.bios.find( { name: { first: "Yukihiko", last: "Matsumoto" } } )
```

- To return all documents in the bios collection where the embedded document name contains a field first with the value "Yukihiko" and a field last with the value "Matsumoto":

```
db.bios.find( { "name.first": "Yukihiko", "name.last": "Matsumoto" } )
```

The query would match documents with name fields that held either of the following values:

```
{ first: "Yukihiko", aka: "Matz", last: "Matsumoto" }
```

```
{ last: "Matsumoto", first: "Yukihiko" }
```


2. Query document find() method example

- To returns documents in the bios collection where the **array** field contains the element "UNIX":

```
db.bios.find( { "contribs" : "UNIX" } )
```

- To returns documents in the bios collection where the awards **array** contains an element with award field equals "Turing Award":

```
db.bios.find( { "awards.award": "Turing Award" } )
```


2. Query document find() method example

- To query for all documents in inventory where the field tags value is an array with **exactly** two elements, "A" and "B", in the specified order:

```
db.inventory.find( { tags: ["A", "B"] } )    //match an array
```

Output: tags: ["A", "B"]
 tags: [["A", "B"], "C"]

- To find an array that contains both the elements "A" and "B", **without regard to order or other elements** in the array, use the \$all operator

```
db.inventory.find( { tags: { $all: ["A", "B"] } } )
```

Output: tags: ["A", "B", "C"]
 tags: ["B", "A"]
 tags: ["A", "B"]

2. Query document

Query operators

| Name | Syntax | Description and Example |
|-------|--|---|
| \$eq | {field: { \$eq: value }} | Match values that are equal to a specified value. db.inventory.find({ qty: { \$eq: 20 } }) |
| \$ne | {field: { \$ne: value }} | Match values that are not equal to a specified value. db.inventory.find({ qty: { \$ne: 20 } }) |
| \$in | {field: { \$in: [value1, value2, ... valueN] }} | Match any of the values specified in an array. db.inventory.find({ qty: { \$in: [5, 15] } }) db.bios.find({ contribs: { \$in: ["ALGOL", "Lisp"] } }) |
| \$nin | {field: { \$nin: [value1, value2, ... valueN] }} | Match none of the values specified in an array. db.inventory.find({ qty: { \$nin: [5, 15] } }) |

2. Query document

Query operators

| Name | Syntax | Description and Example |
|-------|-------------------------|--|
| \$gt | {field: {\$gt: value}} | Match values that are greater than a specified value. db.inventory.find({ qty: { \$gt: 20 } }) |
| \$gte | {field: {\$gte: value}} | Match values that are greater than or equal to a specified value. db.inventory.find({ qty: { \$gte: 20 } }) |
| \$lt | {field: {\$lt: value}} | Match values that are less than a specified value. db.inventory.find({ qty: { \$lt: 20 } }) |
| \$lte | {field: {\$lte: value}} | Match values that are less than or equal to a specified value. db.inventory.find({ qty: { \$lte: 20 } }) |

2. Query document

Query operators

- **\$and**: Return all documents that satisfy **all** the expressions.
 - Syntax: { **\$and**: [{ exp1 }, { exp2 }, ..., { expN }] }
 - Example: select all documents in the inventory collection where the price field exists and not equal to 1.99:

```
db.inventory.find( { $and: [ { price: { $ne: 1.99 } }, { price: { $exists: true } } ] } )
```

```
db.inventory.find( { price: { $ne: 1.99, $exists: true } } )
```

2. Query document

Query operators

- **\$or**: Select the documents that satisfy **at least one** of the expressions.
 - Syntax: `{ $or: [{ exp1 }, { exp2 }, ... , { expN }] }`
 - Example: select all documents in the inventory collection where either the quantity field value is less than 20 or the price field value equals 10.

```
db.inventory.find( { $or: [ { quantity: { $lt: 20 } }, { price: 10 } ] } )
```
- **\$nor**: Select the documents that **fail** all the query expressions in the array
 - Syntax: `{ $nor: [{ exp1 }, { exp2 }, ... { expN }] }`
 - Example:

```
db.inventory.find( { $nor: [ { price: 1.99 }, { qty: 2 } ] } )
```

2. Query document

Query operators

- **\$not**: Return documents that do not match the <operator-expression>. This includes documents that do not contain the field.
 - Syntax: { **field**: { **\$not**: { *operator-expression* } } }
 - Example:
`db.inventory.find({ price: { $not: { $gt: 1.99 } } })`
This query will select all documents in the inventory collection where:
 - the price field value is less than or equal to 1.99 **or**
 - the price field does not exist
 - {**\$not**: { **\$gt**: 1.99 }} is different from the **\$lte** operator
 - {**\$lte**: 1.99} returns only the documents where price field exists and its value is less than or equal to 1.99

2. Query document

Query operators

| Name | Syntax | Description and Example |
|----------|--|---|
| \$exists | { field: { \$exists: <boolean> } } | <p>When <boolean> is true, \$exists matches the documents that contain the field, including documents where the field value is null</p> <p>db.inventory.find({ qty: { \$exists: true, \$nin: [5, 15] } })</p> |
| \$all | { field: { \$all: [value1, value2 ...] } } | <p>Select the documents where the value of a field is an array that contains all the specified elements, without regard to order or other elements in the array.</p> <p>Equivalent to \$and operation</p> <p>db.inventory.find({ tags: { \$all: ["A", "B"] } })</p> |
| \$size | { field: { \$size: number } } | <p>Select documents if the array field is a specified size.</p> <p>db.inventory.find({ tags: { \$size: 3 } })</p> |

2. Query document

Query operators

- **\$regex**: Select documents where values match a specified regular expression. MongoDB uses Perl Compatible Regular Expressions ("PCRE") library to match regular expressions.
 - Syntax:

```
{ <field>: { $regex: /pattern/, $options: '<options>' } }  
{ <field>: { $regex: 'pattern', $options: '<options>' } }  
{ <field>: { $regex: /pattern/<options> } }
```
 - Example: `db.products.find({ sku: { $regex: /abc/ } })`

| Option | Description |
|--------|---|
| i | Do case-insensitive pattern matching |
| m | Treat the string being matched against as multiple lines. For patterns that include anchors (i.e. ^ for the start, \$ for the end), match at the beginning or end of each line for strings with multiline values. |
| s | Treat the string as single line. Allows the dot character to match all characters <i>including</i> newline characters |

2. Query document

Query operators

- **\$regex**: Example:

- match all products documents where the sku field is ended with "789":
`db.products.find({ sku: { $regex: /789$/ } })`
- select documents with sku values starting with "ABC" case-insensitive:
`db.products.find({ sku: { $regex: /^ABC/i } })`
- match lines starting with the letter S for multiline strings:
`db.products.find({ description: { $regex: /^S/m } })`
- match all characters including new line characters:
`db.products.find({ description: { $regex: /m.*line/, $options: 'si' } })`

2. Query document

Query operators

- **\$elemMatch**: Match documents that contain an array field with **at least one element** that matches all the specified query criteria.
 - Syntax: `{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }`
 - Example 1: Suppose the scores collection:

```
{ _id: 1, results: [ 82, 85, 88 ] }
{ _id: 2, results: [ 75, 88, 89 ] }
```

Find documents where the results array contains at least one element that is both greater than or equal to 80 and is less than 85. ($80 \leq x < 85$)

```
db.scores.find( { results: { $elemMatch: { $gte: 80, $lt: 85 } } } )
```

The result is:

```
{ "_id" : 1, "results" : [ 82, 85, 88 ] }
```

2. Query document

Query operators

- \$elemMatch:

- Example 2: array of embedded documents

This query matches only those documents where the results array contains at least one element with both product equal to "xyz" and score greater than or equal to 8:

```
db.survey.insertMany([
  { "_id": 1, "results": [ { "product": "abc", "score": 10 },
                           { "product": "xyz", "score": 5 } ] },
  { "_id": 2, "results": [ { "product": "abc", "score": 8 },
                           { "product": "xyz", "score": 7 } ] },
  { "_id": 3, "results": [ { "product": "abc", "score": 7 },
                           { "product": "xyz", "score": 8 } ] },
  { "_id": 4, "results": [ { "product": "abc", "score": 7 },
                           { "product": "def", "score": 8 } ] }
])
```

```
survey.find({ results: { $elemMatch: { product: "xyz", score: { $gte: 8 } } } })
```

```
→ { "_id" : 3, "results" : [ { "product" : "abc", "score" : 7 }, { "product" : "xyz", "score" : 8 } ] }
```

2. Query document count() method

- `count()`: Count the number of documents referenced by a cursor. Append the `count()` method to a `find()` query to return the number of matching documents.

- Example:

```
db.restaurants.find({'address.zipcode': '11369'}).count()    // the result is 5
```

```
db.restaurants.count({'address.zipcode': '11369'})           // the result is 5
```

2. Query document limit() method

- `limit()`: To maximize performance and prevent MongoDB from returning more results than required for processing.

- Example:

```
db.restaurants.find({ 'address.zipcode': '11369' }).limit(3)    // the result is 3
```

2. Query document skip() method

- `skip()`: Control the starting point of the results set.

- Example:

```
db.restaurants.find( { 'address.zipcode': '11369' }).skip(1)    // the result is 4
```

2. Query document sort() method

- `sort()`: Order the documents in the result set

- Example:

```
db.restaurants.find({ 'address.zipcode': '11369' }).sort( {name: 1} )
```

```
db.restaurants.find({ 'address.zipcode': '11369', {name: 1} }).sort( {name: 1} )
```

1: ascending
-1: descending

projection

2. Query document Projection

- The projection parameter specifies which fields to return, syntax:
`{ field1: value, field2: value, ... }` with value is *<1 or true>* or *<0 or false>*
- Unless the `_id` field is explicitly excluded in the projection document `_id: 0`, the `_id` field is returned.
- Example:
 - Find all documents in the bios collection and returns only the name field, contribs field and `_id` field: `db.bios.find({ }, { name: 1, contribs: 1 })`
 - Find documents in the bios collection and returns only the name field and the contribs field: `db.bios.find({ }, { name: 1, contribs: 1, _id: 0 })`

Bài tập

restaurants.json

```
{
  address: {
    building: '7715',
    coord: [ -73.9973325, 40.611748899999999 ],
    street: '18 Avenue',
    zipcode: '11214' },
    borough: 'Brooklyn',
    cuisine: 'American ',
    grades:
    [
      {date: ISODate("2014-04-16T00:00:00.000Z"),grade: 'A',score: 5},
      {date: ISODate("2013-04-23T00:00:00.000Z"),grade: 'A', score: 2},
      {date: ISODate("2012-04-24T00:00:00.000Z"),grade: 'A', score: 5},
      {date: ISODate("2011-12-16T00:00:00.000Z"), grade: 'A',score: 2}
    ],
    name: 'C & C Catering Service',
    restaurant_id: '40357437'
}
```

Bài tập

1. Hiển thị tất cả các documents có trong collection restaurants.
2. Chèn thêm 1 document vào collection restaurants.
3. Hiển thị tất cả các documents có trong collection restaurants, tuy nhiên chỉ xuất các fields restaurant_id, name, borough and cuisine.
4. Hiển thị tất cả các documents có trong collection restaurants, tuy nhiên chỉ xuất các fields restaurant_id, name, borough and cuisine và không xuất field _id.
5. Hiển thị tất cả các documents có trong collection restaurants với field borough có giá trị là Bronx.
6. Hiển thị 5 documents đầu tiên có trong collection restaurants với field borough có giá trị là Bronx.
7. Hiển thị 5 documents tiếp theo sau khi bỏ qua 5 documents đầu tiên có trong collection restaurants với field borough có giá trị là Bronx.
8. Hiển thị tất cả các documents có trong collection restaurants với điều kiện score trong field grades lớn hơn 90.

Bài tập

9. Hiển thị tất cả các documents có trong collection restaurants với điều kiện score trong field grades lớn hơn 80 và nhỏ hơn 100.
10. Hiển thị tất cả các documents có trong collection restaurants, tuy nhiên chỉ xuất các fields restaurant Id, name, borough, cuisine với name có chứa 3 ký tự bắt đầu là 'Wil'.
11. Hiển thị tất cả các documents có trong collection restaurants, tuy nhiên chỉ xuất các fields restaurant Id, name, borough, cuisine với name có chứa 3 ký tự cuối cùng là 'ces'.
12. Hiển thị tất cả các documents có trong collection restaurants với field borough có giá trị là Bronx và field cuisine có giá trị là American hoặc Chinese.
13. Hiển thị tất cả các documents có trong collection restaurants với field borough có giá trị Staten Island or Queens or Bronx or Brooklyn, chỉ xuất các field restaurant Id, name, borough, cuisine.

Contents

1. Insert document
2. Query document
- 3. Update document**
4. Delete document

3. Update documents

updateOne(), updateMany() method

- Syntax:

```
db.collection.updateOne/updateMany (  
  <filter>,  
  <update>,  
  {  
    upsert: <boolean>,  
    arrayFilters: [ <filterdocument1>, ... ],  
    ...  
  }  
)
```

| Parameter | Type | Description |
|--------------|-----------------------|---|
| filter | document | The selection criteria for the update, same as the find() method. |
| update | document/ pipeline | Update document Contains only update operator expressions : { \$set: { <field1>: <value1>, ... } } |
| | | Aggregation pipeline Contains only the following aggregation stages : |
| upsert | boolean | Defaults to false. When true, updateOne() creates a new document if no documents match the filter. |
| arrayFilters | array | Optional. An array of filter documents that determine which array elements to modify for an update operation on an array field. |

3. Update documents

updateOne(), updateMany() method

- The method returns a document that contains:
 - `matchedCount` containing the number of matched documents
 - `modifiedCount` containing the number of modified documents
 - `upsertedId` containing the `_id` for the upserted document.
 - A boolean `acknowledged` as true if the operation ran with write concern or false if write concern was disabled

3. Update documents

updateOne(), updateMany() method

- Example:

Create the products collection:

```
db.products.insertOne( {  
  _id: 1,  
  quantity: 250,  
  instock: true,  
  reorder: false,  
  details: { model: "14QQ", make: "Clothes Corp" },  
  tags: [ "apparel", "clothing" ],  
  ratings: [ { by: "Customer007", rating: 4 } ]  
} )
```

- Set top-level fields:

```
db.products.update(  
  { _id: 1 },  
  { $set: { qty: 500,  
    details: { model: "14Q3", make: "xyz" },  
    tags: [ "coats", "outerwear", "clothing" ] } }  
)
```

3. Update documents

updateOne(), updateMany() method

- **Set fields in embedded documents:** updates the *make* field in the *details* document.

```
db.products.updateOne(  
  { _id: 1 },  
  { $set: { "details.make": "zzz" } }  
)
```

- **Set elements in arrays:** updates the value second element (array index of 1) in the *tags* field and the *rating* field in the first element (array index of 0) of the *ratings* array.

```
db.products.update(  
  { _id: 1 },  
  { $set: { "tags.1": "rain gear", "ratings.0.rating": 2 } }  
)
```


3. Update documents

Field update operators

- **\$currentDate** operator: Set the value of a field to the current date, either as a Date or a timestamp. The default type is Date. If the field does not exist, \$currentDate will create the field.

{ **\$currentDate**: { <field1>: <typeSpecification1>, ... } }

<typeSpecification> can be either:

- a boolean true to set the field value to the current date as a Date, or
- a document { \$type: "timestamp" } or { \$type: "date" } which explicitly specifies the type.

○ Example:

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

3. Update documents

Field update operators

| Name | Description, syntax and example |
|---------|--|
| \$inc | <p><code>{ \$inc: { <field1>: <amount1>, ... } }</code></p> <p>Increment the value of the field by the specified amount, positive or negative values. If the field does not exist, \$inc creates the field and sets the field to the specified value.</p> <p><code>db.products.update({ sku: "abc123" }, { \$inc: { qty: -2, "metrics.orders": 1 } })</code></p> |
| \$mul | <p><code>{ \$mul: { <field1>: <number1>, ... } }</code></p> <p>Multiply the value of a field by a number. If the field does not exist, \$mul creates the field and sets the value to zero of the same numeric type as the multiplier.</p> <p><code>db.products.update({ sku: "abc123" }, { \$mul: { qty: 2 } })</code></p> <p>Apply \$mul operator to a non-existing field:</p> <p><code>db.products.update({ _id: 104 }, { \$mul: { unit_price: 100 } })</code></p> |
| \$unset | <p><code>{ \$unset: { <field1>: "", ... } }</code></p> <p>Remove the specified field from a document.</p> <p><code>db.products.update({ _id: 104 }, { \$unset: { unit_price: "" } })</code></p> |

3. Update documents

Field update operators

| Name | Description, syntax and example |
|------|---------------------------------|
|------|---------------------------------|

| | |
|-------|--|
| \$min | { \$min: { <field1>: <value1>, ... } } |
|-------|--|

Update the value of the field to a specified value *if* the specified value is **less than** the current value of the field.

If the field does not exist, the \$min operator sets the field to the specified value.

Example: Create the scores collection:

```
db.scores.insertOne( { _id: 1, highScore: 800, lowScore: 200 } )
```

```
db.scores.update( { _id: 1 }, { $min: { lowScore: 150 } } )
```

```
→ { _id: 1, highScore: 800, lowScore: 150 }
```

```
db.scores.update( { _id: 1 }, { $min: { lowScore: 250 } } )
```

```
→ { _id: 1, highScore: 800, lowScore: 150 }
```

3. Update documents

Field update operators

| Name | Description, syntax and example |
|------|---------------------------------|
|------|---------------------------------|

| | |
|--------------------|---|
| <code>\$max</code> | <code>{ \$max: { <field1>: <value1>, ... } }</code> |
|--------------------|---|

Update the value of the field to a specified value *if* the specified value is **greater than** the current value of the field.

If the field does not exist, the \$max operator sets the field to the specified value.

Example: Create the scores collection:

```
db.scores.insertOne( { _id: 1, highScore: 800, lowScore: 200 } )
```

```
db.scores.update( { _id: 1 }, { $max: { highScore: 950 } } )
```

```
→ { _id: 1, highScore: 950, lowScore: 200 }
```

```
db.scores.update( { _id: 1 }, { $max: { highScore: 870 } } )
```

```
→ { _id: 1, highScore: 950, lowScore: 200 }
```

3. Update documents

Array update operators (\$)

| Name | Description | Syntax and Example |
|------|--|--------------------------|
| \$ | update without explicitly specifying the position of the element in the array. It update the first element that matches the query condition. | { "<array>.\$" : value } |

Consider the following document in the students collection:

```
{ _id : 1, grades : [85, 80, 80] },  
{ _id : 2, grades : [88, 90, 92] },  
{ _id : 3, grades : [85, 100, 90] }
```

```
{ _id: 4, grades: [ { grade: 80, mean: 75, std: 8 },  
                   { grade: 85, mean: 90, std: 5 },  
                   { grade: 85, mean: 85, std: 8 } ] }
```

- **Update values in an array:** update the first element whose grades value is 80 to 82:

```
db.students.update ( { _id: 1, grades: 80 }, { $set: { "grades.$" : 82 } } )
```

```
→ { "_id" : 1, "grades" : [ 85, 82, 80 ] }
```

- **Update documents in an array:** update the *std* field of the first array element that matches the grade equal to 85 condition:

```
db.students.updateOne( { _id: 4, "grades.grade": 85 }, { $set: { "grades.$.std" : 6 } } )
```

```
→ { "_id" : 4, "grades" : [ { "grade" : 80, "mean" : 75, "std" : 8 }, { "grade" : 85, "mean" : 90, "std" : 6 }, { "grade" : 85, "mean" : 85, "std" : 8 } ] }
```


3. Update documents

Array update operators (\$)

Consider the following document in the students collection:

```
db.students.insertMany( { _id: 5, grades: [ { grade: 80, mean: 75, std: 8 },  
                                             { grade: 85, mean: 90, std: 5 },  
                                             { grade: 90, mean: 85, std: 3 } ] } )
```

Update embedded documents using multiple field matches: updates the value of the *std* field in the first embedded document that has *grade* field with a value less than or equal to 90 and a *mean* field with a value greater than 80:

```
db.students.updateOne( { _id: 5, grades: { $elemMatch: { grade: { $lte: 90 }, mean: { $gt: 80 } } } },  
                      { $set: { "grades.$.std" : 6 } }  
)
```

→ { _id: 5, grades: [{ grade: 80, mean: 75, std: 8 },
 { grade: 85, mean: 90, std: 6 },
 { grade: 90, mean: 85, std: 3 }] }

3. Update documents

Array update operators (\$[])

| Name | Description | Syntax and Example |
|------|--|---|
| \$[] | Update all elements in the specified array field | { <update operator>: { "<array>.\$[]" : value } } |

Consider the following document in the students collection:

```
db.students.insertMany( { _id : 1, grades : [85, 82, 80] },  
                        { _id : 2, grades : [88, 90, 92] },  
                        { _id : 3, grades : [85, 100, 90] } )
```

Update all elements in an array:

- Increment all elements in the grades array by 10 for all documents in the collection:

```
db.students.updateMany( { }, { $inc: { "grades.$[]" : 10 } } )
```
- Increment all elements in the grades array by 10 for all documents except those with the value 100 in the grades array:

```
db.students.updateMany( { grades : { $ne: 100 } }, { $inc: { "grades.$[]" : 10 } } )
```


3. Update documents

Array update operators (\$[<identifier>])

| Name | Description | Syntax and Example |
|-------------------------------------|---|---|
| <code>\$[<identifier>]</code> | Identify the array elements that match the <code>arrayFilters</code> conditions | <pre>{ <update operator>: { "<array>.\$[<identifier>]" : value } }, { arrayFilters: [{ <identifier>: <condition> }] }</pre> |

Consider the following document in the students collection:

```
db.students.insertMany( [ { "_id" : 1, "grades" : [ 95, 92, 90 ] },  
                          { "_id" : 2, "grades" : [ 98, 100, 102 ] },  
                          { "_id" : 3, "grades" : [ 95, 110, 100 ] } ] )
```

Update all array elements that match `arrayFilters`: update all elements that are greater than or equal to 100 in the grades array:

```
db.students.updateMany(  
  { },  
  { $set: { "grades.$[dieukien]" : 100 } },  
  { arrayFilters: [ { "dieukien": { $gte: 100 } } ] }  
)
```

```
→ { _id: 1, grades: [ 95, 92, 90 ] },  
   { _id: 2, grades: [ 98, 100, 100 ] },  
   { _id: 3, grades: [ 95, 100, 100 ] }
```

3. Update documents

Array update operators (\$[<identifier>])

Consider the following document in the students collection:

```
db.students.insertMany( { "_id" : 1, "grades" : [ { "grade" : 80, "mean" : 75, "std" : 6 },  
                                                    { "grade" : 85, "mean" : 90, "std" : 4 },  
                                                    { "grade" : 85, "mean" : 85, "std" : 6 } ] },  
                        { "_id" : 2, "grades" : [ { "grade" : 90, "mean" : 75, "std" : 6 },  
                                                    { "grade" : 87, "mean" : 90, "std" : 3 },  
                                                    { "grade" : 85, "mean" : 85, "std" : 4 } ] } )
```

Update all documents that match arrayFilters in an array: modify the value of the mean field for all elements in the grades array where the grade is greater than or equal to 85:

```
db.students2.updateMany(                                → { "_id" : 1, "grades" : [ { "grade" : 80, "mean" : 75, "std" : 6 },  
  { },                                                  { "grade" : 85, "mean" : 100, "std" : 4 },  
  { $set: { "grades.$[elem].mean" : 100 } },           { "grade" : 85, "mean" : 100, "std" : 6 } ] },  
  { arrayFilters: [ { "elem.grade": { $gte: 85 } } ] } { "_id" : 2, "grades" : [ { "grade" : 90, "mean" : 100, "std" : 6 },  
  )                                                    { "grade" : 87, "mean" : 100, "std" : 3 },  
                                                       { "grade" : 85, "mean" : 100, "std" : 4 } ] }
```

3. Update documents

Array update operators (\$[<identifier>])

Consider the following document in the students collection:

```
db.students.insertMany( { "_id" : 1, "grades" : [ { "grade" : 80, "mean" : 75, "std" : 6 },  
                                                    { "grade" : 85, "mean" : 100, "std" : 4 },  
                                                    { "grade" : 85, "mean" : 100, "std" : 6 } ] },  
                        { "_id" : 2, "grades" : [ { "grade" : 90, "mean" : 100, "std" : 6 },  
                                                    { "grade" : 87, "mean" : 100, "std" : 3 },  
                                                    { "grade" : 85, "mean" : 100, "std" : 4 } ] } )
```

Update all array elements that match multiple conditions: modify the value of the *std* field for all elements in the *grades* array where both the *grade* is greater than or equal to 80 and the *std* is greater than or equal to 5:

```
db.students.updateMany(  
  { },  
  { $inc: { "grades.$[elem].std" : -1 } },  
  { arrayFilters: [ { "elem.grade": { $gte: 80 }, "elem.std": { $gt: 5 } } ] }  
)
```

3. Update documents

Array update operators (\$[<identifier>])

Consider the following document in the students collection:

```
db.students.insertMany( { _id : 1, grades : [ { type: "quiz", questions: [ 10, 8, 5 ] },  
                                              { type: "quiz", questions: [ 8, 9, 6 ] },  
                                              { type: "hw", questions: [ 5, 4, 3 ] },  
                                              { type: "exam", questions: [ 25, 10, 23, 0 ] } ] } )
```

Update nested arrays in conjunction with \$[<identifier>]: updates the values where *grades.questions* array are greater than or equal to 8 if the associated *grades.type* field is "quiz":

```
db.students.updateMany(  
  {},  
  { $inc: { "grades.$[t].questions.$[score]": 2 } },  
  { arrayFilters: [ { "t.type": "quiz" },  
                    { "score": { $gte: 8 } } ] }  
)
```

→

```
{ "_id" : 1,  
  "grades" : [  
    { "type" : "quiz", "questions" : [ 12, 10, 5 ] },  
    { "type" : "quiz", "questions" : [ 10, 11, 6 ] },  
    { "type" : "hw", "questions" : [ 5, 4, 3 ] },  
    { "type" : "exam", "questions" : [ 25, 10, 23, 0 ] } ] }
```


3. Update documents

Array update operators (\$addToSet)

| Name | Description | Syntax and Example |
|------------|--|---|
| \$addToSet | <p>Add elements to an array only <u>if they do not already exist</u> in the set.</p> <p>If you use \$addToSet on a field that is absent in the document to update, \$addToSet creates the array field with the specified value as its element.</p> | { \$addToSet: { <field1>: <value1>, ... } } |

Consider a collection inventory with the following document:

```
db.inventory.insertOne( { _id: 1, item: "polarizing_filter", tags: [ "electronics", "camera" ] } )
```

- **Add a value to the array field:** adds the element "accessories" to the tags array:

```
db.inventory.updateOne( { _id: 1 }, { $addToSet: { tags: "accessories" } } )
```

- **Add multiple values to the array field:** use \$each modifier:

```
db.inventory.updateOne(  
  { _id: 2 },  
  { $addToSet : { tags : { $each : [ "camera", "electronics", "accessories" ] } } }  
)
```

3. Update documents

Array update operators (\$pop)

| Name | Description | Syntax and Example |
|-------|--|---------------------------------------|
| \$pop | Removes the first or last item of an array | { \$pop: { <field>: <-1 1>, ... } } |

Consider the following document in the students collection:

```
db.students.insertOne( { _id: 1, scores: [ 8, 9, 10 ] } )
```

- **Remove the first item of an array:**

```
db.students.update( { _id: 1 }, { $pop: { scores: -1 } } )
```

- **Remove the last item of an array:**

```
db.students.update( { _id: 1 }, { $pop: { scores: 1 } } )
```


3. Update documents

Array update operators (\$pull, \$pullAll)

| Name | Description | Syntax and Example |
|-----------|--|--|
| \$pull | Removes all array elements that match a specified query | { \$pull: { <field1>: <value condition>, <field2>: <value condition>, ... } } |
| \$pullAll | Removes all instances of the specified values from an existing array | { \$pullAll: { <field1>: [<value1>, <value2> ...], ... } } |

Given the following document in a collection stores:

```
{ _id: 1,  
  fruits: [ "apples", "pears", "oranges", "grapes", "bananas" ],  
  vegetables: [ "carrots", "celery", "squash", "carrots" ] },  
{ _id: 2,  
  fruits: [ "plums", "kiwis", "oranges", "bananas", "apples" ],  
  vegetables: [ "broccoli", "zucchini", "carrots", "onions" ] }
```

- **Remove all items that equal a specified value:**

```
db.stores.updateMany( { }, { $pull: { fruits: { $in: [ "apples", "oranges" ] }, vegetables: "carrots" } } )
```

3. Update documents

Array update operators (\$pull, \$pullAll)

Given the following document in the profiles collection: { _id: 1, votes: [3, 5, 6, 7, 7, 8] }

- **Remove all items that match a specified \$pull condition:**

db.profiles.update({ _id: 1 }, { \$pull: { votes: { \$gte: 6 } } }) → { _id: 1, votes: [3, 5] }

- **Removes all instances of the specified values:** removes all instances of the values "0" and "5" from the votes array:

db.profiles.updateOne({ _id: 1 }, { \$pullAll: { votes: [3, 7] } }) → { _id : 1, votes: [5, 6, 8] }

Given the following document in the profiles collection:

{ _id: 1, results: [{ item: "A", score: 5 }, { item: "B", score: 8 }] },
{ _id: 2, results: [{ item: "C", score: 8 }, { item: "B", score: 4 }] }

- **Remove items from an array of documents:**

db.survey.updateMany({ },
 { \$pull: { results: { score: 8 , item: "B" } } }
)

→ { _id: 1, results: [{ item: 'A', score: 5 }] },
 { _id: 2, results: [{ item: 'C', score: 8 }, { item: 'B', score: 4 }] }

3. Update documents

Array update operators (\$push)

| Name | Description | Syntax and Example |
|--------|--|---|
| \$push | Append a specified value to an array. If the field is absent, \$push adds the array field with the value as its element. | { \$push: { <field1>: <value1>, ... } } |

Given the following document in the students collection:

```
db.students.insertOne( { _id: 1, scores: [ 44, 78, 38, 80 ] } )
```

- **Append a value to an array:**

```
db.students.update( { _id: 1 }, { $push: { scores: 89 } } )
```

- **Append multiple values to an array:** Use \$push with the \$each modifier

```
db.students.update( { name: "joe" }, { $push: { scores: { $each: [ 90, 92, 85 ] } } } )
```

3. Update documents

Array update operators

| Name | Description |
|-----------------------------------|--|
| <u>\$each</u> | Modifies the <u>\$push</u> and <u>\$addToSet</u> operators to append multiple items for array updates. |
| <u>\$position</u> | Modifies the <u>\$push</u> operator to specify the position in the array to add elements. |
| <u>\$slice</u> | Modifies the <u>\$push</u> operator to limit the size of updated arrays. |
| <u>\$sort</u> | Modifies the <u>\$push</u> operator to reorder documents stored in an array. |

self-study

3. Update documents

Array update operators

Use **\$push** operator with multiple modifiers:

Given the following document in the students collection:

```
{ "_id" : 5, "quizzes" : [ { "wk": 1, "score" : 10 }, { "wk": 2, "score" : 8 },  
                           { "wk": 3, "score" : 5 }, { "wk": 4, "score" : 6 } ] }
```

The following **\$push** operation uses:

- the **\$each** modifier to add multiple documents to the *quizzes* array,
- the **\$sort** modifier to sort all the elements of the modified *quizzes* array by the *score* field in descending order, and
- the **\$slice** modifier to keep only the **first** three sorted elements of the quizzes array.

```
db.students.updateOne(  
  { _id: 5 },  
  { $push: {  
    quizzes: {  
      $each: [ { wk: 5, score: 8 }, { wk: 6, score: 7 }, { wk: 7, score: 6 } ],  
      $sort: { score: -1 },  
      $slice: 3 } } }  
)  
→ { "_id" : 5, "quizzes" : [ { "wk" : 1, "score" : 10 },  
                             { "wk" : 2, "score" : 8 },  
                             { "wk" : 5, "score" : 8 } ] }
```

Contents

1. Insert document
2. Query document
3. Update document
4. **Delete document**

4. Delete documents

| Method | Description |
|---|---|
| db.collection.deleteOne() | Remove a single document from a collection, removes the first document that matches the filter. |
| db.collection.deleteMany() | Remove all documents that match the <filter> from a collection. |
| <pre>db.collection.deleteOne(<filter>, //Specify an empty document { } to delete the first { writeConcern: <document>, collation: <document>, hint: <document string> // Available starting in MongoDB 4.4 })</pre> | |

4. Delete documents

Delete collections/database

| Method | Description |
|-----------------------------------|--------------------------------------|
| <code>db.collection.drop()</code> | Delete a collection from a database. |
| <code>db.dropDatabase()</code> | Delete a database. |