Introduction to MongoDB findOne() method

The findOne() returns a single document from a collection that satisfies the specified condition.

The findOne() method has the following syntax:

db.collection.findOne(query, projection)

Code language: CSS (css)

The findOne() accepts two optional arguments: query and projection.

- The query is a document that specifies the selection criteria.
- The projection is a document that specifies the fields in the matching document that you want to return.

If you omit the query, the findOne() returns the first document in the collection according to the natural order which is the order of documents on the disk.

If you don't pass the projection argument, then findOne() will include all fields in the matching documents.

To specify whether a field should be included in the returned document, you use the following format:

{field1: value, field1: value, ... }

If the value is true or 1, MongoDB will include the field in the returned document. In case the value is false or 0, MongoDB won't include it.

By default, MongoDB always includes the _id field in the returned documents. To suppress it, you need to explicitly specify _id: 0 in the projection argument.

If multiple documents satisfy the query, the findOne() method returns the first document based on the order of documents stored on the data storage.

Note that there are other forms of projections such as array projection and aggregation projection which are not covered in this tutorial.

MongoDB findOne() method examples

We'll use the following products collection for the demonstration:

db.products.insertMany([

{ "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},

```
   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec"
: { "ram" : 16, "screen" : 9.5, "cpu" : 3.66
},"color":["white","black","purple"],"storage":[128,256,512]},

   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"),
"spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},

   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-
14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"),
"spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256]}

 ])
```

Code language: JavaScript (javascript)

1) Using MongoDB findOne() method with zero argument

The following example uses the findOne() method to select the first document from the products collection:

```
db.products.findOne()
```

Code language: CSS (css)

The statement returns all fields of the matching document:

```
{
  _id: 1,
  name: 'xPhone',
  price: 799,
  releaseDate: ISODate("2011-05-14T00:00:00.000Z"),
  spec: { ram: 4, screen: 6.5, cpu: 2.66 },
  color: [ 'white', 'black' ],
  storage: [ 64, 128, 256 ]
}
```

Code language: CSS (css)

Note that omitting the query is the same as passing the query as an empty document:

```
db.products.findOne({})
```

Code language: CSS (css)

2) Using MongoDB findOne() method with a filter

The following statement uses the findOne() method to find the document whose _id is 2.

```
 db.products.findOne({_id:2})
```

Code language: CSS (css)

It returns all the fields of the document with _id 2:

```
{
  _id: 2,
  name: 'xTablet',
  price: 899,
  releaseDate: ISODate("2011-09-01T00:00:00.000Z"),
  spec: { ram: 16, screen: 9.5, cpu: 3.66 },
  color: [ 'white', 'black', 'purple' ],
  storage: [ 128, 256, 512 ]
}
```

Code language: CSS (css)

3) Using MongoDB findOne() method to select some fields

The following example uses the findOne() method to find the document with _id 5. And it returns only the _id and name fields of the matching document:

```
db.products.findOne({_id: 5}, {name: 1})
```

Code language: JavaScript (javascript)

The query returned the following document:

```
{ "_id" : 5, "name" : "SmartPhone" }
```

Code language: JavaScript (javascript)

As you can see clearly from the output, MongoDB includes the _id field in the returned document by default.

To completely remove it from the returned document, you need to explicitly specify _id:0 in the projection document like this:

```
db.products.findOne({ _id: 5 }, {name: 1, _id: 0})
```

Code language: JavaScript (javascript)

It returned the following document:

```
{ "name" : "SmartPhone" }
```

Code language: JavaScript (javascript)

Summary

- Use the findOne() method to retrieve the first document in a collection that satisfies the specified condition.

Introduction to the MongoDB find() method

The find() method finds the documents that satisfy a specified condition and returns a cursor to the matching documents.

The following shows the syntax of the find() method:

```
db.collection.find(query, projection)
```

Code language: CSS (css)

Similar to the findOne() method, the find() method accepts two optional arguments.

1) query

The query is a document that specifies the criteria for selecting documents from the collection. If you omit the query or pass an empty document({}), the find() returns a cursor that returns all documents in the collection.

2) projection

The projection is a document that specifies the fields in the matching documents to return. If you omit the projection argument, the find() method returns all fields in the matching documents.

By default, the find() method includes the _id field in the matching documents unless you explicitly specify _id: false in the projection document.

Since the mongo shell automatically iterates the cursor returned by the find() method, you don't need to do any extra steps to get the document from the cursor.

By default, the mongo shell shows up the first 20 documents. To continue iteration, you type the it command in the shell.

The MongoDB find() method examples

We'll use the following books collection for the demonstration:

```
db.books.insertMany([
    { "_id" : 1, "title" : "Unlocking Android", "isbn" : "1933988673", "categories" : [ "Open Source", "Mobile" ] },
    { "_id" : 2, "title" : "Android in Action, Second Edition", "isbn" : "1935182722", "categories" : [ "Java" ] },
    { "_id" : 3, "title" : "Specification by Example", "isbn" : "1617290084", "categories" : [ "Software Engineering" ] },
    { "_id" : 4, "title" : "Flex 3 in Action", "isbn" : "1933988746", "categories" : [ "Internet" ] },
    { "_id" : 5, "title" : "Flex 4 in Action", "isbn" : "1935182420", "categories" : [ "Internet" ] },
    { "_id" : 6, "title" : "Collective Intelligence in Action", "isbn" : "1933988312", "categories" : [ "Internet" ] },
    { "_id" : 7, "title" : "Zend Framework in Action", "isbn" : "1933988320", "categories" : [ "Web Development" ] },
    { "_id" : 8, "title" : "Flex on Java", "isbn" : "1933988797", "categories" : [ "Internet" ] },
    { "_id" : 9, "title" : "Griffon in Action", "isbn" : "1935182234", "categories" : [ "Java" ] },
    { "_id" : 10, "title" : "OSGi in Depth", "isbn" : "193518217X", "categories" : [ "Java" ] },
    { "_id" : 11, "title" : "Flexible Rails", "isbn" : "1933988509", "categories" : [ "Web Development" ] },
    { "_id" : 13, "title" : "Hello! Flex 4", "isbn" : "1933988762", "categories" : [ "Internet" ] },
    { "_id" : 14, "title" : "Coffeehouse", "isbn" : "1884777384", "categories" : [ "Miscellaneous" ] },
    { "_id" : 15, "title" : "Team Foundation Server 2008 in Action", "isbn" : "1933988592", "categories" : [ "Microsoft .NET" ] },
    { "_id" : 16, "title" : "Brownfield Application Development in .NET", "isbn" : "1933988711", "categories" : [ "Microsoft" ] },
    { "_id" : 17, "title" : "MongoDB in Action", "isbn" : "1935182870", "categories" : [ "Next Generation Databases" ] },
```

```
{ "_id" : 18, "title" : "Distributed Application Development with PowerBuilder 6.0", "isbn" : "1884777686", "categories" : [ "PowerBuilder" ] },

{ "_id" : 19, "title" : "Jaguar Development with PowerBuilder 7", "isbn" : "1884777864", "categories" : [ "PowerBuilder", "Client-Server" ] },

{ "_id" : 20, "title" : "Taming Jaguar", "isbn" : "1884777686", "categories" : [ "PowerBuilder" ] },

{ "_id" : 21, "title" : "3D User Interfaces with Java 3D", "isbn" : "1884777902", "categories" : [ "Java", "Computer Graphics" ] },

{ "_id" : 22, "title" : "Hibernate in Action", "isbn" : "193239415X", "categories" : [ "Java" ] },

{ "_id" : 23, "title" : "Hibernate in Action (Chinese Edition)", "categories" : [ "Java" ] },

{ "_id" : 24, "title" : "Java Persistence with Hibernate", "isbn" : "1932394885", "categories" : [ "Java" ] },

{ "_id" : 25, "title" : "JSTL in Action", "isbn" : "1930110529", "categories" : [ "Internet" ] },

{ "_id" : 26, "title" : "iBATIS in Action", "isbn" : "1932394826", "categories" : [ "Web Development" ] },

{ "_id" : 27, "title" : "Designing Hard Software", "isbn" : "133046192", "categories" : [ "Object-Oriented Programming", "S" ] },

{ "_id" : 28, "title" : "Hibernate Search in Action", "isbn" : "1933988649", "categories" : [ "Java" ] },

{ "_id" : 29, "title" : "jQuery in Action", "isbn" : "1933988355", "categories" : [ "Web Development" ] },

{ "_id" : 30, "title" : "jQuery in Action, Second Edition", "isbn" : "1935182323", "categories" : [ "Java" ] }

]);
```

Code language: JavaScript (javascript)

1) Using MongoDB find() method to retrieve all documents from a collection

The following example uses the find() method with no parameters to return all documents from the books collection:

```
db.books.find()
```

Code language: CSS (css)

In the mongo shell, the statement returns the first 20 documents with all available fields in the matching documents.

If you type it command and press enter, you'll see the next 20 documents.

2) Using MongoDB find() method to search for a specific document

The following example uses the find() method to search for the document with id 10:

```
db.books.find({_id: 10})
```

Code language: CSS (css)

The statement returns the document whose _id is 10. Since it doesn't have the projection argument, the returned document includes all available fields:

```
[
  {
    _id: 10,
    title: 'OSGi in Depth',
    isbn: '193518217X',
    categories: [ 'Java' ]
  }
]
```

Code language: JavaScript (javascript)

3) Using MongoDB find() method to return selected fields

The following example uses the find() method to search for documents whose category is Java. It returns the fields _id, title and isbn:

```
db.books.find({ categories: 'Java'}, { title: 1,isbn: 1})
```

Code language: CSS (css)

Output:

```
[
  {
    _id: 2,
    title: 'Android in Action, Second Edition',
    isbn: '1935182722'
```

```
  },
  { _id: 9, title: 'Griffon in Action', isbn: '1935182234' },
  { _id: 10, title: 'OSGi in Depth', isbn: '193518217X' },
  {
    _id: 21,
    title: '3D User Interfaces with Java 3D',
    isbn: '1884777902'
  },
  { _id: 22, title: 'Hibernate in Action', isbn: '193239415X' },
  { _id: 23, title: 'Hibernate in Action (Chinese Edition)' },
  {
    _id: 24,
    title: 'Java Persistence with Hibernate',
    isbn: '1932394885'
  },
  { _id: 28, title: 'Hibernate Search in Action', isbn: '1933988649' },
  {
    _id: 30,
    title: 'jQuery in Action, Second Edition',
    isbn: '1935182323'
  }
]
```

Code language: JavaScript (javascript)

To remove the _id field from the matching documents, you need to explicitly specify _id: 0 in the projection argument like this:

```
db.books.find({ categories: 'Java'}, { title: 1,isbn: 1,_id: 0})
```

Code language: CSS (css)

Output:

```
[
  { title: 'Android in Action, Second Edition', isbn: '1935182722' },
  { title: 'Griffon in Action', isbn: '1935182234' },
  { title: 'OSGi in Depth', isbn: '193518217X' },
  { title: '3D User Interfaces with Java 3D', isbn: '1884777902' },
  { title: 'Hibernate in Action', isbn: '193239415X' },
  { title: 'Hibernate in Action (Chinese Edition)' },
  { title: 'Java Persistence with Hibernate', isbn: '1932394885' },
  { title: 'Hibernate Search in Action', isbn: '1933988649' },
  { title: 'jQuery in Action, Second Edition', isbn: '1935182323' }
]
```

Code language: JavaScript (javascript)

Note that you'll learn how to construct more complex conditions using operators in the next tutorials.

Summary

- Use the find() method to select the documents from a collection and returns a cursor referencing the matching documents.

Introduction to the MongoDB projection

In MongoDB, projection simply means selecting fields to return from a query.

By default, the find() and findOne() methods return all fields in matching documents. Most of the time you don't need data from all the fields.

To select fields to return from a query, you can specify them in a document and pass the document to the find() and findOne() methods. This document is called a projection document.

To determine if a field is included in the returned documents, you use the following syntax:

```
{ <field>: value, ...}
```

Code language: HTML, XML (xml)

If the value is 1 or true, the <field> is included in the matching documents. In case the value is 0 or false, it is suppressed from the returned documents.

If the projection document is empty {}, all the available fields will be included in the returned documents.

To specify a field in an embedded document, you use the following dot notation:

{ "<embeddedDocument>.<field>": value, ... }

Code language: HTML, XML (xml)

Similarly, to include a <field> from an embedded document located in an array, you use the following dot notation syntax:

{ "<arrayField>.field": value, ...}

Code language: HTML, XML (xml)

MongoDB projection examples

We'll use the following products collection for the projection examples.

```
db.products.insertMany([

   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256],"inventory":[{ qty: 1200,"warehouse": "San Jose"}]},

   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 },"color":["white","black","purple"],"storage":[128,256,512],"inventory":[{ qty: 300,"warehouse": "San Francisco"}]},

   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128],"inventory":[{ qty: 400,"warehouse": "San Jose"},{ qty: 200,"warehouse": "San Francisco"}]},

   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256,1024],"inventory":[{ qty: 1200,"warehouse": "San Mateo"}]},

   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256]}

 ])
```

Code language: JavaScript (javascript)

1) Returning all fields in matching documents

If you don’t specify the projection document, the find() method will include all fields in the matching documents.

For example, the following query returns all fields from all documents in the products collection where the price is 899:

```
db.products.find({price: 899});
```

Code language: CSS (css)

Output:

```
[
 {
  _id: 2,
  name: 'xTablet',
  price: 899,
  releaseDate: ISODate("2011-09-01T00:00:00.000Z"),
  spec: { ram: 16, screen: 9.5, cpu: 3.66 },
  color: [ 'white', 'black', 'purple' ],
  storage: [ 128, 256, 512 ],
  inventory: [ { qty: 300, warehouse: 'San Francisco' } ]
 },
 {
  _id: 3,
  name: 'SmartTablet',
  price: 899,
  releaseDate: ISODate("2015-01-14T00:00:00.000Z"),
  spec: { ram: 12, screen: 9.7, cpu: 3.66 },
  color: [ 'blue' ],
  storage: [ 16, 64, 128 ],
  inventory: [
   { qty: 400, warehouse: 'San Jose' },
```

```
    { qty: 200, warehouse: 'San Francisco' }
  ]
 }
]
```

Code language: JavaScript (javascript)

2) Returning specified fields including the _id field

If you specify the fields in the projection document, the find() method will return only these fields including the _id field by default.

The following example returns all documents from the products collection. However, its result includes only the name, price, and _id field in the matching documents:

```
db.products.find({}, {
  name: 1,
  price: 1
});
```

Code language: CSS (css)

Output:

```
[
 { _id: 1, name: 'xPhone', price: 799 },
 { _id: 2, name: 'xTablet', price: 899 },
 { _id: 3, name: 'SmartTablet', price: 899 },
 { _id: 4, name: 'SmartPad', price: 699 },
 { _id: 5, name: 'SmartPhone', price: 599 }
]
```

Code language: JavaScript (javascript)

To suppress the _id field, you need to explicitly specify it in the projection document like this:

```
db.products.find({}, {
  name: 1,
  price: 1
```

,

_id: 0

});

Code language: CSS (css)

Output:

[

{ name: 'xPhone', price: 799 },

{ name: 'xTablet', price: 899 },

{ name: 'SmartTablet', price: 899 },

{ name: 'SmartPad', price: 699 },

{ name: 'SmartPhone', price: 599 }

]

Code language: JavaScript (javascript)

3) Returning all fields except for some fields

If the number of fields to return is many, you can use the projection document to exclude other fields instead.

The following example returns all fields of the document _id 1 except for releaseDate, spec, and storage fields:

db.products.find({_id:1}, {

releaseDate: 0,

spec: 0,

storage: 0

})

Code language: CSS (css)

Output:

[

{

_id: 1,

name: 'xPhone',

```
    price: 799,
    color: [ 'white', 'black' ],
    inventory: [ { qty: 1200, warehouse: 'San Jose' } ]
  }
]
```

Code language: JavaScript (javascript)

4) Returning fields in embedded documents

The following example returns the name, price, and _id fields of document _id 1. It also returns the screen field on the spec embedded document:

```
db.products.find({_id:1}, {
    name: 1,
    price: 1,
    "spec.screen": 1
})
```

Code language: CSS (css)

Output:

```
[ { _id: 1, name: 'xPhone', price: 799, spec: { screen: 6.5 } } ]
```

Code language: CSS (css)

MongoDB 4.4 and later allows you to specify embedded fields using the nested form like this:

```
db.products.find({_id:1}, {
    name: 1,
    price: 1,
    spec : { screen: 1 }
})
```

Code language: CSS (css)

5) Projecting fields on embedded documents in an array

The following example specifies a projection that returns:

- the _id field (by default)

- The name field
- And qty field in the documents embedded in the inventory array.

```
db.products.find({}, {
   name: 1,
   "inventory.qty": 1
});
```

Code language: CSS (css)

Output:

```
[
 { _id: 1, name: 'xPhone', inventory: [ { qty: 1200 } ] },
 { _id: 2, name: 'xTablet', inventory: [ { qty: 300 } ] },
 {
   _id: 3, name: 'SmartTablet', inventory: [ { qty: 400 }, { qty: 200 } ]
 },
 { _id: 4, name: 'SmartPad', inventory: [ { qty: 1200 } ] },
 { _id: 5, name: 'SmartPhone' }
]
```

Code language: JavaScript (javascript)

Summary

- Use the {<field>: 1} to include the <field> in the matching documents and {<field>: 0} to exclude it.
- Use the { <embeddedDocument>.<field>: 1} to include the <field> from the <embeddedDocument> in the matching document and { <embeddedDocument>.<field>: 0} to suppress it.
- Use { <arrayField>.<field>: 1} to include the <field> from the embedded document in an array in the matching document and { <arrayField>.<field>: 0} to exclude it.