Introduction to MongoDB sort() method

To specify the order of the documents returned by a query, you use the sort() method:

cursor.sort({field1: order, field2: order, ...})

Code language: HTTP (http)

The sort() method allows you to sort the matching documents by one or more fields (field1, field2, …) in ascending or descending order.

The order takes two values: 1 and -1. If you specify { field: 1 }, the sort() will sort the matching documents by the field in ascending order:

cursor.sort({ field: 1 })

Code language: CSS (css)

If you specify { field: -1}, the sort() method will sort matching documents by the field in descending order:

cursor.sort({field: -1})

Code language: CSS (css)

The following sorts the returned documents by the field1 in ascending order and field2 in descending order:

cursor.sort({field1: 1, field2: -1});

Code language: HTTP (http)

It’s straightforward to compare values of the same type. However, it is not the case for comparing the values of different BSON types.

MongoDB uses the following comparison order from lowest to highest for comparing values of different BSON types:

1. MinKey (internal type)
2. Null
3. Numbers (ints, longs, doubles, decimals)
4. Symbol, String
5. Object
6. Array
7. BinData
8. ObjectId

9. Boolean
10. Date
11. Timestamp
12. Regular Expression
13. MaxKey (internal type)

For more information on comparison/sort order, check out this page.

MongoDB sort() method examples

We'll use the following products collection to illustrate how the sort() method works.

```
db.products.insertMany([
  { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate" : ISODate("2011-05-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 }, "color" : [ "white", "black" ], "storage" : [ 64, 128, 256 ] },
  { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate" : ISODate("2011-09-01T00:00:00Z"), "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("2015-01-14T00:00:00Z"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "blue" ], "storage" : [ 16, 64, 128 ] },
  { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate" : ISODate("2020-05-14T00:00:00Z"), "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("2022-09-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 9.7, "cpu" : 1.66 }, "color" : [ "white", "orange", "gold", "gray" ], "storage" : [ 128, 256 ] },
  { "_id" : 6, "name" : "xWidget", "spec" : { "ram" : 64, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "black" ], "storage" : [ 1024 ] },
  { "_id" : 7, "name" : "xReader", "price" : null, "spec" : { "ram" : 64, "screen" : 6.7, "cpu" : 3.66 }, "color" : [ "black", "white" ], "storage" : [ 128 ] }
])
```

Code language: JavaScript (javascript)

1) Sorting document by one field examples

The following query returns all documents from the products collection where the price field exists. For each document, it selects the _id, name, and price fields:

```
db.products.find({
    'price': {
        $exists: 1
    }
}, {
    name: 1,
    price: 1
})
```

Code language: PHP (php)

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 },
  { _id: 7, name: 'xReader', price: null }
]
```

Code language: JavaScript (javascript)

To sort the products by prices in ascending order, you use the sort() method like this:

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

```
    price: 1
}).sort({
    price: 1
})
```

Code language: PHP (php)

Output:

```
[
  { _id: 7, name: 'xReader', price: null },
  { _id: 5, name: 'SmartPhone', price: 599 },
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 1, name: 'xPhone', price: 799 },
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 }
]
```

Code language: JavaScript (javascript)

In this example, the sort() method places the document whose price is null first, and then the documents with the prices from lowest to highest.

To sort the documents in descending order, you change the value of the price field to -1 as shown in the following query:

```
db.products.find({
    'price': {
        $exists: 1
    }
}, {
    name: 1,
    price: 1
}).sort({
    price: -1
})
```

Code language: PHP (php)

Output:

```
[
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 },
  { _id: 1, name: 'xPhone', price: 799 },
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 5, name: 'SmartPhone', price: 599 },
  { _id: 7, name: 'xReader', price: null }
]
```

Code language: JavaScript (javascript)

In this example, the sort() method places the document with the highest price first and the one whose price is null last. (See the sort order above)

2) Sorting document by two or more fields example

The following example uses the sort() method to sort the products by name and price in ascending order. It selects only documents where the price field exists and includes the _id, name, and price fields in the matching documents.

```
db.products.find({
    'price': {
        $exists: 1
    }
}, {
    name: 1,
    price: 1
}).sort({
    price: 1,
    name: 1
});
```

Code language: PHP (php)

Output:

```
[
  { _id: 7, name: 'xReader', price: null },
  { _id: 5, name: 'SmartPhone', price: 599 },
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 1, name: 'xPhone', price: 799 },
  { _id: 3, name: 'SmartTablet', price: 899 },
  { _id: 2, name: 'xTablet', price: 899 }
]
```

Code language: JavaScript (javascript)

In this example, the sort() method sorts the products by prices first. Then it sorts the sorted result set by names.

If you look at the result set more closely, you'll see that the products with _id 3 and 2 have the same price 899. The sort() method places the SmartTablet before the xTablet based on the ascending order specified by the name field.

The following example sorts the products by prices in ascending order and sorts the sorted products by names in descending order:

```
db.products.find({
    'price': {
        $exists: 1
    }
}, {
    name: 1,
    price: 1
}).sort({
    price: 1,
    name: -1
})
```

Code language: PHP (php)

Output:

```
[
  { _id: 7, name: 'xReader', price: null },
  { _id: 5, name: 'SmartPhone', price: 599 },
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 1, name: 'xPhone', price: 799 },
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 }
]
```

Code language: JavaScript (javascript)

In this example, the sort() method sorts the products by prices in ascending order. However, it sorts sorted products by names in descending order.

Unlike the previous example, the sort() places the xTable before SmartTablet.

3) Sorting documetns by dates

The following example sorts the documents from the products collection by values in the releaseDate field. It selects only document whose releaseDate field exists and includes the _id, name, and releaseDate fields in the matching documents:

```
db.products.find({
    releaseDate: {
        $exists: 1
    }

}, {
    name: 1,
    releaseDate: 1
}).sort({
    releaseDate: 1
});
```

Code language: CSS (css)

Output:

```
[
  {
    _id: 1,
    name: 'xPhone',
    releaseDate: ISODate("2011-05-14T00:00:00.000Z")
  },
  {
    _id: 2,
    name: 'xTablet',
    releaseDate: ISODate("2011-09-01T00:00:00.000Z")
  },
  {
    _id: 3,
    name: 'SmartTablet',
    releaseDate: ISODate("2015-01-14T00:00:00.000Z")
  },
  {
    _id: 4,
    name: 'SmartPad',
    releaseDate: ISODate("2020-05-14T00:00:00.000Z")
  },
  {
    _id: 5,
    name: 'SmartPhone',
    releaseDate: ISODate("2022-09-14T00:00:00.000Z")
  }
]
```

Code language: JavaScript (javascript)

In this example, the sort() method places the documents with the releaseDate in ascending order.

The following query sorts the products by the values in the releaseDate field in descending order:

```
db.products.find({
   releaseDate: {
      $exists: 1
   }

}, {
   name: 1,
   releaseDate: 1
}).sort({
   releaseDate: -1
});
```

Code language: CSS (css)

Output:

```
[
 {
   _id: 5,
   name: 'SmartPhone',
   releaseDate: ISODate("2022-09-14T00:00:00.000Z")
 },
 {
   _id: 4,
   name: 'SmartPad',
   releaseDate: ISODate("2020-05-14T00:00:00.000Z")
 },
```

```
  {
    _id: 3,
    name: 'SmartTablet',
    releaseDate: ISODate("2015-01-14T00:00:00.000Z")
  },
  {
    _id: 2,
    name: 'xTablet',
    releaseDate: ISODate("2011-09-01T00:00:00.000Z")
  },
  {
    _id: 1,
    name: 'xPhone',
    releaseDate: ISODate("2011-05-14T00:00:00.000Z")
  }
]
```

Code language: JavaScript (javascript)

4) Sorting documents by fields in embedded documents

The following example sorts the products by the values in the ram field in the spec embedded documents. It includes the _id, name, and spec fields in the matching documents.

```
db.products.find({}, {
    name: 1,
    spec: 1
}).sort({
    "spec.ram": 1
});
```

Code language: JavaScript (javascript)

Output:

```
[
  { _id: 1, name: 'xPhone', spec: { ram: 4, screen: 6.5, cpu: 2.66 } },
  {
    _id: 5,
    name: 'SmartPhone',
    spec: { ram: 4, screen: 9.7, cpu: 1.66 }
  },
  {
    _id: 4,
    name: 'SmartPad',
    spec: { ram: 8, screen: 9.7, cpu: 1.66 }
  },
  {
    _id: 3,
    name: 'SmartTablet',
    spec: { ram: 12, screen: 9.7, cpu: 3.66 }
  },
  {
    _id: 2,
    name: 'xTablet',
    spec: { ram: 16, screen: 9.5, cpu: 3.66 }
  },
  {
    _id: 6,
    name: 'xWidget',
    spec: { ram: 64, screen: 9.7, cpu: 3.66 }
  },
  {
```

```
  _id: 7,
  name: 'xReader',
  spec: { ram: 64, screen: 6.7, cpu: 3.66 }
 }
]
```

Code language: JavaScript (javascript)

Summary

- Use the sort() method to sort the documents by one or more fields.
- Specify { field: 1 } to sort documents by the field in ascending order and { field: -1 } to sort documents by the field in descending order.
- Use the dot notation { "embeddedDoc.field" : 1 } to sort the documents by the field in the embedded documents (embeddedDoc).

Introduction to MongoDB limit() method

The find() method may return a lot of documents to the application. Typically, the application may not need that many documents.

To limit the number of returned documents, you use the limit() method:

```
db.collection.find(<query>).limit(<documentCount>)
```

Code language: CSS (css)

The <documentCount> is in the range of -2^{31} and 2^{31}. If you specify a value for the <documentCount> that is out of this range, the behavior of the limit() is unpredictable.

If the <documentCount> is negative, the limit() returns the same number of documents as if the <documentCount> is positive. In addition, it closes the cursor after returning a single batch of documents.

If the result set does not fit into a single batch, the number of returned documents will be less than the specified limit.

If the <documentCount> is zero, then is equivalent to setting no limit.

Note that the limit() is analogous to the LIMIT clause in SQL.

To get the predictable result set using the limit(), you need to sort the result set first before applying the method like this:

```
cursor
  .sort({...})
  .limit(<documentCount>)
```

Code language: HTML, XML (xml)

In practice, you often use the limit() with the skip() method to paginate a collection.

The skip() method specifies from where the query should start returning the documents:

```
cursor.skip(<offset>)
```

Code language: HTML, XML (xml)

The following shows the documents on the page pageNo with
the documentCount documents per page:

```
db.collection.find({...}
).sort({...}
).skip(pageNo > 0 ? ( ( pageNo - 1 ) * documentCount) : 0
).limit(documentCount);
```

The skip(<offset>) requires the MongoDB server to scan from the beginning of the result set before starting to return the documents. When the <offset> increases, the skip() will become slower.

Note that the limit() and skip() is analogous to the LIMIT OFFSET clause in SQL.

MongoDB limit() examples

We'll use the following products collection:

```
db.products.insertMany([
  { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate" : ISODate("2011-05-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 }, "color" : [ "white", "black" ], "storage" : [ 64, 128, 256 ] },
  { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate" : ISODate("2011-09-01T00:00:00Z"), "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("2015-01-14T00:00:00Z"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "blue" ], "storage" : [ 16, 64, 128 ] },
```

{ "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate" : ISODate("2020-05-14T00:00:00Z"), "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("2022-09-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 9.7, "cpu" : 1.66 }, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256] },

{ "_id" : 6, "name" : "xWidget", "spec" : { "ram" : 64, "screen" : 9.7, "cpu" : 3.66 }, "color" : ["black"], "storage" : [1024] },

{ "_id" : 7, "name" : "xReader", "price" : null, "spec" : { "ram" : 64, "screen" : 6.7, "cpu" : 3.66 }, "color" : ["black", "white"], "storage" : [128] }

])

Code language: JavaScript (javascript)

1) Using MongoDB limit() to get the most expensive product

The following example uses the limit() method to get the most expensive product in the products collection. It includes the _id, name, and price fields in the returned documents:

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

Code language: CSS (css)

Output:

```
[ { _id: 2, name: 'xTablet', price: 899 } ]
```

Code language: CSS (css)

In this example, we sort the products by prices in descending order and use limit() to select the first one.

The products collection has two products at the same price 899. The returned document depends on the order of documents stored on the disk.

To get the predictable result, the sort should be unique. For example:

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

Code language: CSS (css)

Output:

```
[ { _id: 3, name: 'SmartTablet', price: 899 } ]
```

Code language: CSS (css)

This example sorts the document by prices in descending order. And then it sorts the sorted result set by names in ascending order. The limit() returns the first document in the final result set.

2) Using MongoDB limit() and skip() to get the paginated result

Suppose you want to divide the products collection into pages, each has 2 products.

The following query uses the skip() and limit() to get the documents on the second page:

```
db.products.find({}, {
    name: 1,
    price: 1
}).sort({
    price: -1,
    name: 1
}).skip(2).limit(2);
```

Code language: CSS (css)

Output:

```
[
  { _id: 1, name: 'xPhone', price: 799 },
  { _id: 4, name: 'SmartPad', price: 699 }
```

]

Code language: JavaScript (javascript)

Summary

- Use limit() to specify the number of returned documents for a query.
- Use sort() and limit() to select top / bottom N documents.
- Use skip() and limit() to paginate the documents in a collection.