MONGODB SORTING: SORT() METHOD & EXAMPLES

In this article, I'll show you how to use the MongoDB **sort() method** in various scenarios, including how to use it alongside other methods to sort a given data set.

Using the sort() method will increase the readability of a query, which leads to a better understanding of a given dataset. Not only that, sorted data will be used by developers to write more complex algorithms.

(This article is part of our MongoDB Guide. Use the right-hand menu to navigate.)

What is database sorting?

Database sorting presents data in an ascending or descending order with relation to the data in a specified field. You can carry out sorting operations on various data types such as:

- Strings
- Integers
- Decimal
- Etc.

The main advantage of sorting is that it increases the readability and uniformity of the data, which helps the users grasp the meaning of the data set more effectively.

MongoDB sort()

In MongoDB, sorting is done by the **sort()** method. The sort() method consists of two basic building

blocks. These building blocks are fields to be sorted and the sort order.

The sorting order in MongoDB is defined by either a one (1) or a minus (-1). Here the positive one represents the ascending order, while the negative one represents the descending order.

Basic syntax of MongoDB sort()

```
db.collection_name.find().sort({field_name: sort order})
```

According to the official documentation, MongoDB uses the following order when comparing values of different BSON types from lowest to highest. (BSON stands for **Binary JSON** format.)

This is the serialization format used in MongoDB to store documents and make remote procedure calls. Any non-existent fields in a document are treated as Null objects.

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)

The following examples will demonstrate the differences between an unsorted and a sorted MongoDB search query. For this example, we will be using the “vehicleinformation” collection.

“vehicleinformation” collection data

```
mongos> db.vehicleinformation.find().pretty()
{
        "_id" : ObjectId("5faaf5541139223fe5c19a01"),
        "make" : "Nissan",
        "model" : "GTR",
        "year" : 2016,
        "type" : "Sports",
        "reg_no" : "EDS 5578"
}
{
        "_id" : ObjectId("5faaf7131139223fe5c19a02"),
        "make" : "BMW",
        "model" : "X5",
        "year" : 2020,
        "type" : "SUV",
        "reg_no" : "LLS 6899"
}
{
        "_id" : ObjectId("5faaf7221139223fe5c19a03"),
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
{
        "_id" : ObjectId("5faaf7291139223fe5c19a04"),
        "make" : "Audi",
        "model" : "RS3",
        "year" : 2018,
        "type" : "Sports",
        "reg_no" : "RFD 7866"
}
{
        "_id" : ObjectId("5faaf72f1139223fe5c19a05"),
        "make" : "Ford",
        "model" : "Transit",
        "year" : 2017,
        "type" : "Van",
        "reg_no" : "TST 9880"
}
{
        "_id" : ObjectId("5faaf7371139223fe5c19a06"),
        "make" : "Honda",
        "model" : "Gold Wing",
        "year" : 2018,
        "type" : "Bike",
        "reg_no" : "LKS 2477"
}
mongos>
```

Unsorted collection

When a search query is carried out with the find() method, the default behavior is to return the output unsorted. The **{_id:0}** operator is used to remove the document ID for a simpler output.

```
db.vehicleinformation.find({},{_id:0})
```

Results in:

```
mongos> db.vehicleinformation.find({},{_id:0})
{ "make" : "Nissan", "model" : "GTR", "year" : 2016, "type" : "Sports", "re
g_no" : "EDS 5578" }
{ "make" : "BMW", "model" : "X5", "year" : 2020, "type" : "SUV", "reg_no" :
 "LLS 6899" }
{ "make" : "Toyota", "model" : "Yaris", "year" : 2019, "type" : "Compact",
"reg_no" : "HXE 0153" }
{ "make" : "Audi", "model" : "RS3", "year" : 2018, "type" : "Sports", "reg_
no" : "RFD 7866" }
{ "make" : "Ford", "model" : "Transit", "year" : 2017, "type" : "Van", "reg
_no" : "TST 9880" }
{ "make" : "Honda", "model" : "Gold Wing", "year" : 2018, "type" : "Bike",
"reg_no" : "LKS 2477" }
mongos>
```

Sorted collection

To get a sorted result, we append the **sort()** method to the end of the search query (**find()** method). This allows the user to generate a sorted output.

In this instance, the data is sorted by the "year" field in ascending order.

```
db.vehicleinformation.find({},{_id:0}).sort({"year":1})
```

Results in:

```
mongos> db.vehicleinformation.find({},{_id:0}).sort({"year":1})
{ "make" : "Nissan", "model" : "GTR", "year" : 2016, "type" : "Sports", "re
g_no" : "EDS 5578" }
{ "make" : "Ford", "model" : "Transit", "year" : 2017, "type" : "Van", "reg
_no" : "TST 9880" }
{ "make" : "Audi", "model" : "RS3", "year" : 2018, "type" : "Sports", "reg_
no" : "RFD 7866" }
{ "make" : "Honda", "model" : "Gold Wing", "year" : 2018, "type" : "Bike",
"reg_no" : "LKS 2477" }
{ "make" : "Toyota", "model" : "Yaris", "year" : 2019, "type" : "Compact",
"reg_no" : "HXE 0153" }
{ "make" : "BMW", "model" : "X5", "year" : 2020, "type" : "SUV", "reg_no" :
 "LLS 6899" }
mongos>
```

If you don't give any arguments to the sort() method, the collection will not be sorted, and the resulting output will be in the default order—which is the order Mongo finds the results.

```
db.vehicleinformation.find({},{_id:0}).sort({})<.pre>
```

Result:

```
mongos> db.vehicleinformation.find({},{_id:0}).sort({})
{ "make" : "Nissan", "model" : "GTR", "year" : 2016, "type" : "Sports", "reg_no" :
 "EDS 5578" }
{ "make" : "BMW", "model" : "X5", "year" : 2020, "type" : "SUV", "reg_no" : "LLS 6
899" }
{ "make" : "Toyota", "model" : "Yaris", "year" : 2019, "type" : "Compact", "reg_no
" : "HXE 0153" }
{ "make" : "Audi", "model" : "RS3", "year" : 2018, "type" : "Sports", "reg_no" : "
RFD 7866" }
{ "make" : "Ford", "model" : "Transit", "year" : 2017, "type" : "Van", "reg_no" :
"TST 9880" }
{ "make" : "Honda", "model" : "Gold Wing", "year" : 2018, "type" : "Bike", "reg_no
" : "LKS 2477" }
mongos>
```

MongoDB sort() method usage

This section will cover how the sort() method can be used to carry out different sorting operations. Jump to the Sorting option you need:

- In ascending order
- In descending order
- With multiple fields
- With the limit() method
- With the skip() method
- Metadata
- With an index

Sorting in ascending order

In this example, I use the "make" text field to obtain the results in ascending order. The operator one ({"make":1}) is used to indicate the ascending order, and MongoDB projection is used to filter out all the other fields except the "make" field.

```
db.vehicleinformation.find({},{make:1,_id:0}).sort({"make":1})
```

Result:

```
mongos> db.vehicleinformation.find({},{make:1,_id:0}).sort({"make":1})
{ "make" : "Audi" }
{ "make" : "BMW" }
{ "make" : "Ford" }
{ "make" : "Honda" }
{ "make" : "Nissan" }
{ "make" : "Toyota" }
mongos>
```

Sorting in descending order

This example is the same as the above with one difference, which is using minus one ({"make":-1})

operator to indicate the descending order.

```
db.vehicleinformation.find({},{make:1,_id:0}).sort({"make":-1})
```

Result:

```
mongos> db.vehicleinformation.find({},{make:1,_id:0}).sort({"make":-1})
{ "make" : "Toyota" }
{ "make" : "Nissan" }
{ "make" : "Honda" }
{ "make" : "Ford" }
{ "make" : "BMW" }
{ "make" : "Audi" }
mongos>
```

Sorting using multiple fields

When sorting multiple fields, you should declare fields to be sorted within the sort() method. The query will be sorted according to the declaration position of the fields, where the sort order is evaluated from left to right. To demonstrate this, we will be using “vehiclesales” collection.

“vehiclessales” collection

```
db.vehiclesales.find({},{_id:0})
```

Result:

```
mongos> db.vehiclesales.find({},{_id:0})
{ "make" : "Audi", "model" : "RS3", "price" : 47000, "type" : "Sports" }
{ "make" : "BMW", "model" : "X3", "price" : 35000, "type" : "SUV" }
{ "make" : "Audi", "model" : "A1", "price" : 25000, "type" : "Compact" }
{ "make" : "Nissan", "model" : "GTR", "price" : 55000, "type" : "Sports" }
{ "make" : "Toyota", "model" : "Yaris", "price" : 21000, "type" : "Compact" }
{ "make" : "Audi", "model" : "RS5", "price" : 77000, "type" : "Sports" }
mongos>
```

The following example will show how to sort using the “make” and “price” fields. The data is first sorted by “make” as it’s the first argument, and then the data set will be further sorted by the “price” field.

```
db.vehiclesales.find({},{_id:0}).sort({"make":1,"price":1})
```

Result:

```
mongos> db.vehiclesales.find({},{_id:0}).sort({"make":1,"price":1})
{ "make" : "Audi", "model" : "A1", "price" : 25000, "type" : "Compact" }
{ "make" : "Audi", "model" : "RS3", "price" : 47000, "type" : "Sports" }
{ "make" : "Audi", "model" : "RS5", "price" : 77000, "type" : "Sports" }
{ "make" : "BMW", "model" : "X3", "price" : 35000, "type" : "SUV" }
{ "make" : "Nissan", "model" : "GTR", "price" : 55000, "type" : "Sports" }
{ "make" : "Toyota", "model" : "Yaris", "price" : 21000, "type" : "Compact" }
mongos>
```

As shown above, the data is first sorted by the make field. As there are multiple documents with the

same make “Audi,” the data gets sorted again by the price field in an ascending order resulting in the above output.

Sorting with the limit() method

The sort() method can be used along with the limit() method that limits the number of results in the search query. You should pass an integer to the limit() method, which then specifies the number of documents to which the result set should be limited.

The following examples use the "vehicleinformation" collection while the result is limited to two documents and sorted by both the ascending and descending order.

```
db.vehicleinformation.find({},{_id:0}).sort({"make":1,"year":1}).limit(2).pretty()
```

Result:

```
mongos> db.vehicleinformation.find({},{_id:0}).sort({"make":1,"year":1}).limit(2).pretty()
{
        "make" : "Audi",
        "model" : "RS3",
        "year" : 2018,
        "type" : "Sports",
        "reg_no" : "RFD 7866"
}
{
        "make" : "BMW",
        "model" : "X5",
        "year" : 2020,
        "type" : "SUV",
        "reg_no" : "LLS 6899"
}
mongos>
```

```
db.vehicleinformation.find({},{_id:0}).sort({"make":-1,"year":-1}).limit(2).pretty()
```

Results in:

```
mongos> db.vehicleinformation.find({},{_id:0}).sort({"make":-1,"
year":-1}).limit(2).pretty()
{
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
{
        "make" : "Nissan",
        "model" : "GTR",
        "year" : 2016,
        "type" : "Sports",
        "reg_no" : "EDS 5578"
}
mongos>
```

Sorting with the skip() method

You can also use the skip() method with the sort() method. The skip() method allows the user to skip a specified number of documents from the resulting dataset.

In the following example, You can see the first four documents are being skipped while being sorted by the year in ascending order.

```
db.vehicleinformation.find({},{_id:0}).sort({"year":1}).skip(4).pretty()
```

Result:

```
mongos> db.vehicleinformation.find({},{_id:0}).sort({"year":1}).
skip(4).pretty()
{
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
{
        "make" : "BMW",
        "model" : "X5",
        "year" : 2020,
        "type" : "SUV",
        "reg_no" : "LLS 6899"
}
mongos>
```

Metadata sorting

The sort() method can be used to sort the metadata values for a calculated metadata field.

The following example used the "food" collection to demonstrate how documents can be sorted using the metadata "textScore." The field name in the sort() method can be arbitrary as the query system ignores the field name.

“Food” collection

```
db.food.find({},{_id:0})
```

Result:

```
mongos> db.food.find({},{_id:0})
{ "f_id" : 1, "food_desc" : "pizza" }
{ "f_id" : 2, "food_desc" : "burger" }
{ "f_id" : 3, "food_desc" : "pizza and coke" }
{ "f_id" : 4, "food_desc" : "a slice of pizza" }
{ "f_id" : 5, "food_desc" : "chicken wings" }
{ "f_id" : 6, "food_desc" : "large pizza" }
{ "f_id" : 7, "food_desc" : "pizza pizza" }
mongos>
```

```
db.food.find({$text:{$search: "pizza"}}, {score:{$meta: "textScore"}, _id:
0}).sort({sort_example:{$meta: "textScore"}})
```

Result:

```
mongos> db.food.find({$text:{$search: "pizza"}}, {score:{$meta: "text
Score"}, _id: 0}).sort({sort_example:{$meta: "textScore"}})
{ "f_id" : 7, "food_desc" : "pizza pizza", "score" : 1.5 }
{ "f_id" : 1, "food_desc" : "pizza", "score" : 1.1 }
{ "f_id" : 4, "food_desc" : "a slice of pizza", "score" : 0.75 }
{ "f_id" : 3, "food_desc" : "pizza and coke", "score" : 0.75 }
{ "f_id" : 6, "food_desc" : "large pizza", "score" : 0.75 }
mongos>
```

In this query, we have specified the sort field as “sort_example.” However, this is ignored as we are sorting metadata. Moreover, since we are sorting using “textScore” metadata, the resulting data set is sorted in descending order.

Sorting with an index

MongoDB can perform sort operations on a single-field index in ascending or descending order. In compound indexes, the sort order determines whether the index can be sorted. The sort keys must be listed in the same order as defined in the index.

For example, the compound index {make: 1, year: 1} can be sorted using “sort({make: 1, year: 1})” but not on “sort({year: 1, make: 1})”. Sorting using an index helps to reduce the resource requirements when performing the query.

Using the “vehicleslaes " collection, we define an index named "make_index"

```
db.vehiclesales.find({},{_id:0}).sort({make_index: 1})
```

Result:

```
mongos> db.vehiclesales.createIndex({make: 1},{name:"make_index"})
{
        "raw" : {
                "ShardReplSet/10.10.10.58:27018" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1605468911, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1605468911, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Here, the index “make_index” is used to sort the documents. To identify which index is used, we append the explain() method to the end of the query, which will result in the following output. From the output, we can identify that the “make_index” is used to fetch the relevant documents.

```
db.vehiclesales.find({},{_id:0}).sort({make: 1}).explain()
```

Result:

Finally, the query is run without the **explain()** method to obtain the output.

```
db.vehiclesales.find({},{_id:0}).sort({make: 1})
```

Result:

```
mongos> db.vehiclesales.find({},{_id:0}).sort({make: 1})
{ "make" : "Audi", "model" : "RS3", "price" : 47000, "type" : "Sports" }
{ "make" : "Audi", "model" : "A1", "price" : 25000, "type" : "Compact" }
{ "make" : "Audi", "model" : "RS5", "price" : 77000, "type" : "Sports" }
{ "make" : "BMW", "model" : "X3", "price" : 35000, "type" : "SUV" }
{ "make" : "Nissan", "model" : "GTR", "price" : 55000, "type" : "Sports" }
{ "make" : "Toyota", "model" : "Yaris", "price" : 21000, "type" : "Compact" }
mongos>
```

That's the end of our MongoDB sorting tutorial. Explore the right-hand menu for more MongoDB concepts and examples.

Related reading

- BMC Machine Learning & Big Data Blog
- MongoDB Guide, a series of articles and tutorials
- MongoDB: The Mongo Shell & Basic Commands
- Data Storage Explained: Data Lake vs Warehouse vs Database