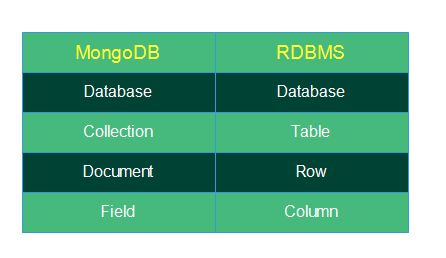
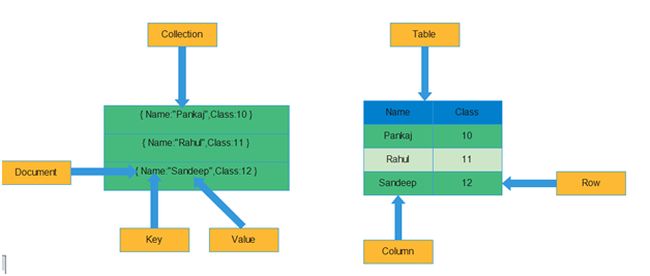
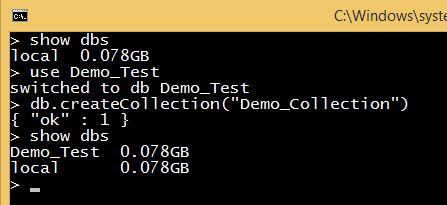
Mongodb documentation

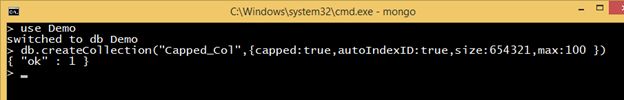
<https://docs.mongodb.com/manual/reference/mongodb-extended-json/>

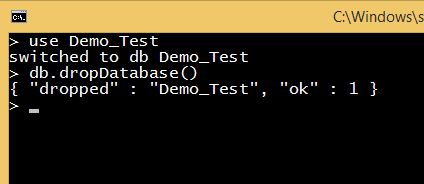


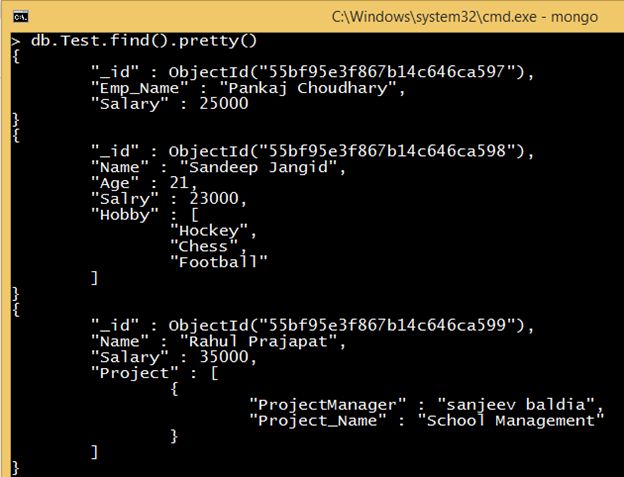




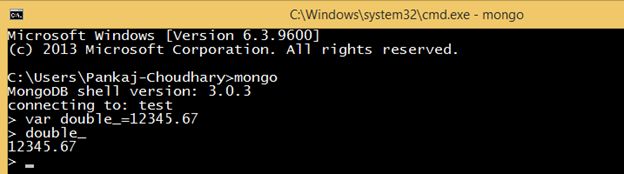
|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| Capped | Boolean | If the value of a Capped field is true then that means the collection is a capped type. The default value is false |
| autoIndexID | Boolean | If the value is true then an index on the \_id field is created automatically. The default value is false. |
| Size | Number | The maximum size of the collection is specified in bytes. If we set the Capped field to true then we need to specify this field also. |
| Max | Number | The maximum number of documents allowed in the collection |

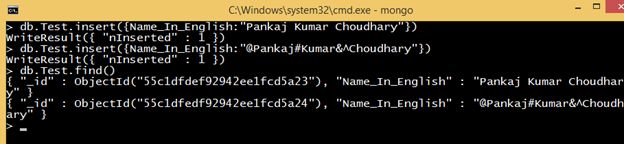






|  |  |
| --- | --- |
| **Data Type** | **Number** |
| Double | 1 |
| String | 2 |
| Object | 3 |
| Array | 4 |
| Binary Data | 5 |
| Undefined | 6 |
| Object Id | 7 |
| Boolean | 9 |
| Date | 10 |
| Null | 11 |
| Regular Expression | 12 |
| JavaScript | 13 |
| Symbol | 14 |
| JavaScript with scope | 15 |
| Integer | 16 and 18 |
| Timestamp | 10 |
| Min Key | 255 |
| Max Key | 12 |







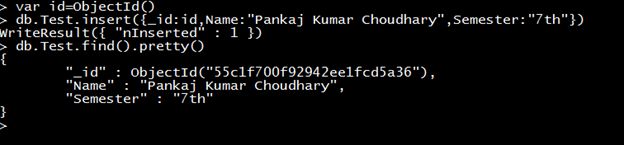


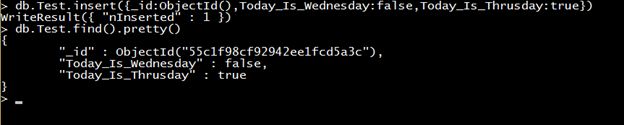


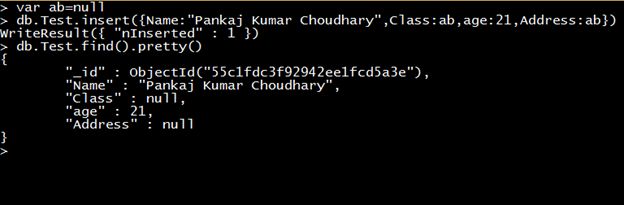
**ObjectId**  
The ObjectId data type stores the document's ID. ObjectId is: small, likely unique, fast to generate and ordered. The size of ObjectId is 12 bytes. These 12 bytes are divided into the following 4 parts.

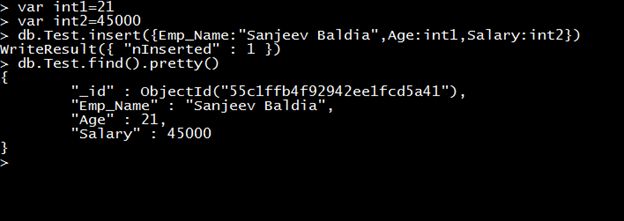
|  |  |
| --- | --- |
| **Part Name** | **Size(Bytes)** |
| Time Stamp | 4 |
| Machine Id | 3 |
| Process Id | 2 |
| Counter | 3 |

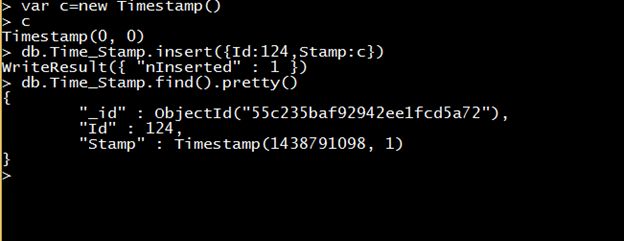
MongoDB uses an \_id field for each document to uniquely identiy them. This \_id field *s*a primary something. It can't be a duplicate. Data is stored in hexadecimal format in the \_id field.



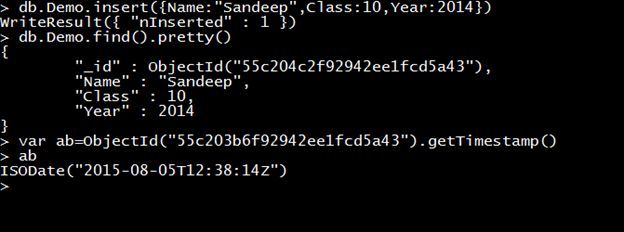


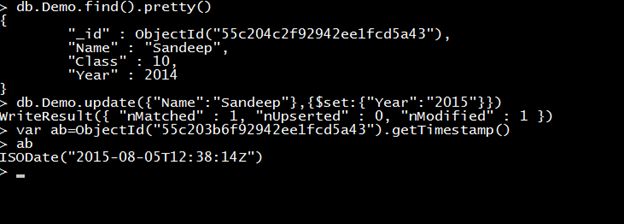






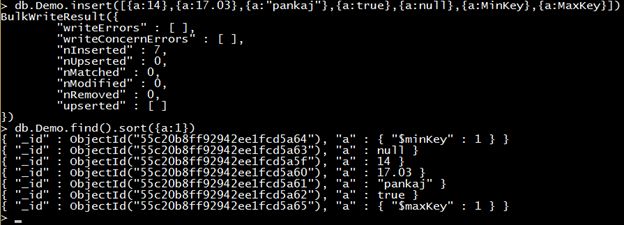
In the preceding example we can see that the starting value of the Timestamp is (0,0). After the insertion operation the value of the timestamp is (1438791098,1). The first value of the timestamp is the current timestamp and the second value is the order of operation.

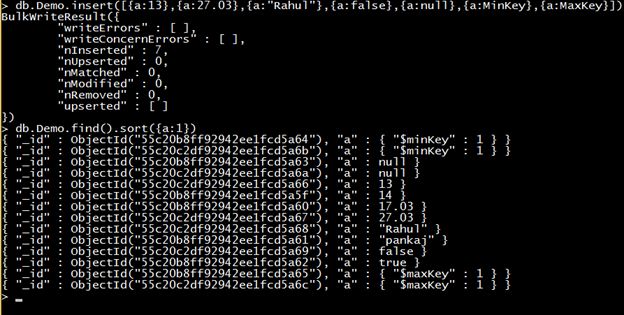




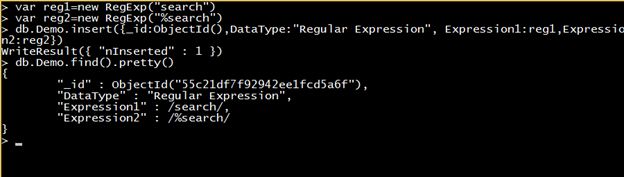
**Min/Max keys**Min/Max keys compare a value against the lowest and highest BSON elements. Min and Max keys both are Internal data types. When comparing values of different BSON types, MongoDB uses the following comparison order from lowest to highest:

1. MinKey
2. Null
3. Numbers (ints, longs, doubles)
4. Symbol, String
5. Object
6. Array
7. BinData
8. ObjectId
9. Boolean
10. Date
11. Timestamp
12. Regular Expression
13. MaxKey





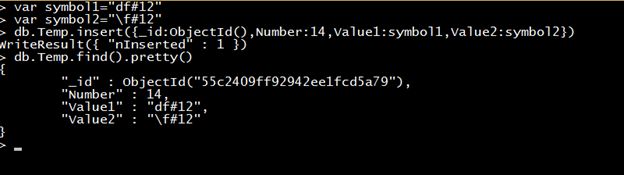
RE

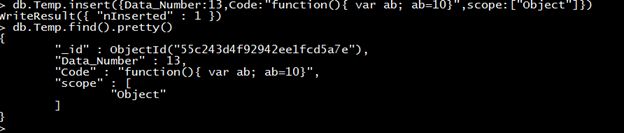


undefined



symbol





MongoDB sort() method

In MongoDB, sort() method is used to sort the documents in the collection. This method accepts a document containing list of fields along with their sorting order.

The sorting order is specified as 1 or -1.

* 1 is used for ascending order sorting.
* -1 is used for descending order sorting.

**Syntax:**

1. db.COLLECTION\_NAME.find().sort({**KEY**:1})

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS Equivalent** |
| Equality | {<key>:<value>} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

And

db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

OR

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

]

}

).pretty()

db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},

{"title": "MongoDB Overview"}]}).pretty()

Update

db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

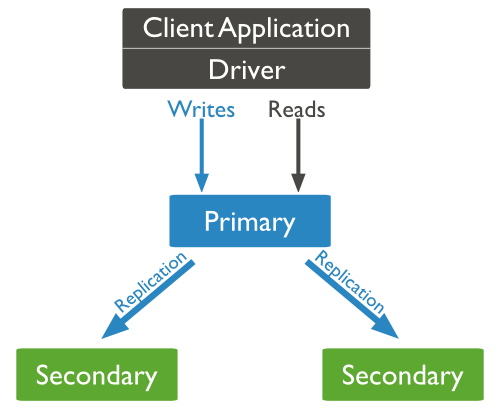
indexing

db.mycol.ensureIndex({"title":1})

If you want to display a list stating how many tutorials are written by each user, then you will use the following **aggregate()** method –

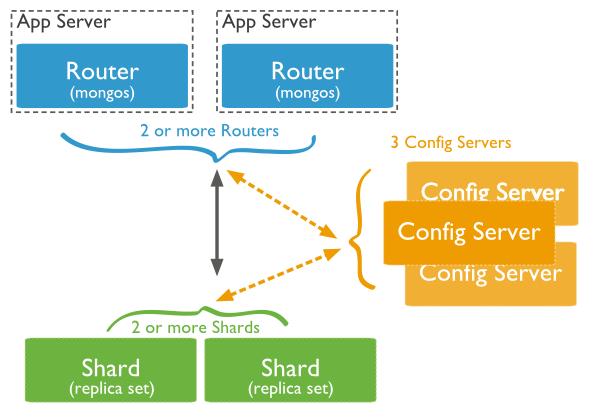
db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

|  |  |  |
| --- | --- | --- |
| **Expression** | **Description** | **Example** |
| $sum | Sums up the defined value from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | Gets the minimum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | Gets the maximum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | Inserts the value to an array in the resulting document. | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

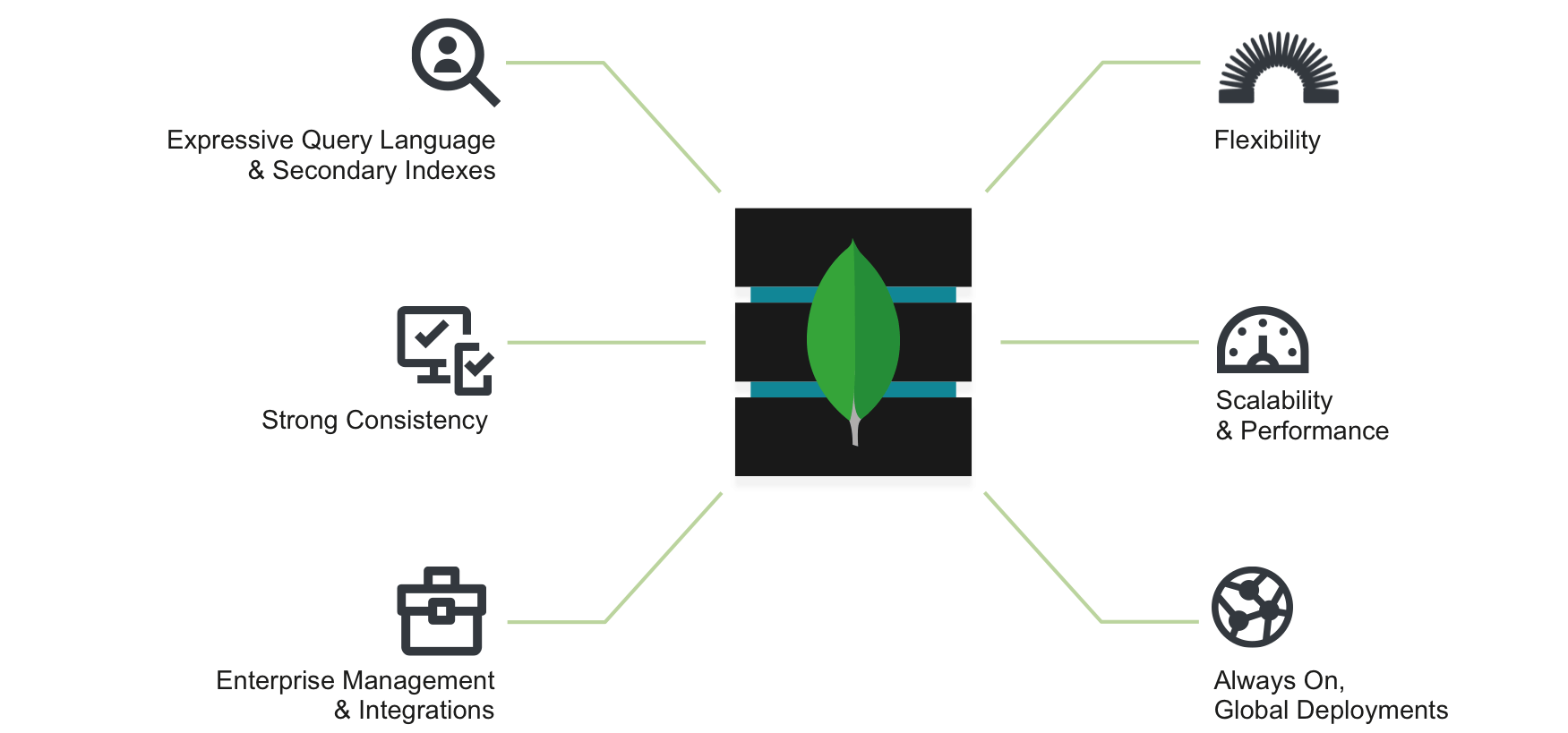


## **Why Sharding?**

* In replication, all writes go to master node
* Latency sensitive queries still go to master
* Single replica set has limitation of 12 nodes
* Memory can't be large enough when active dataset is big
* Local disk is not big enough
* Vertical scaling is too expensive







MongoDB supports many datatypes. Some of them are −

* **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
* **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
* **Boolean** − This type is used to store a boolean (true/ false) value.
* **Double** − This type is used to store floating point values.
* **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
* **Arrays** − This type is used to store arrays or list or multiple values into one key.
* **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
* **Object** − This datatype is used for embedded documents.
* **Null** − This type is used to store a Null value.
* **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
* **Date**− This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
* **Object ID** − This datatype is used to store the document’s ID.
* **Binary data** − This datatype is used to store binary data.
* **Code** − This datatype is used to store JavaScript code into the document.
* **Regular expression** − This datatype is used to store regular expression.



{

\_id: POST\_ID

title: TITLE\_OF\_POST,

description: POST\_DESCRIPTION,

by: POST\_BY,

url: URL\_OF\_POST,

tags: [TAG1, TAG2, TAG3],

likes: TOTAL\_LIKES,

comments: [

{

user: 'COMMENT\_BY',

message: TEXT,

datecreated: DATE\_TIME,

like: LIKES

},

{

user: 'COMMENT\_BY',

message: TEST,

dateCreated: DATE\_TIME,

like: LIKES

}}}

