Basic Commands

* - show dbs --> To check list of all available databases
* - db --> To check currently selected db
* - show collections --> To list all collections in selected db
* - use <db name> (like use local) --> To select and use the db. If db doesn't exist MongoDB will create and use it
* - db.dropDatabse() --> To drop the current db
* - db.stats() --> To view the stats of current db

Cursor

* MongoDB doesn't return all the documents when we call find(), but instead it returns documents in patches and that patch is k/a cursor
* A cursor is pointer to a list of documents. It is an object
* In shell we can use "it" command to req more documents. But in PL (like Node, php) it doesn't work. Instead we've methods like "db.collection.find().toArray()" to return all documents when working with PL
* Another cursor method is "forEach()" which we can use to transform results like

db.customer.find().forEach(function(cust){print("Customer name: " + cust.name})

Working with collections

Queries

* db.createCollection("<collection name>") 🡪 To create collection only (i.e. db.createCollection("products") )
* db.createCollection("collection") also takes optional 2nd arg which is object that specify whether collection is capped or not if capped(property) is set to true and if it's set to true we have to specify size(prop in bytes) too. Also we can define max(prop) to specify how many documents are allowed in the collection (A capped collection is one that overwrites its prev entries when max size is reached using size prop). We can also specify autoIndexId(prop) which indexes the default \_id, like

db.createCollection("log", { capped : true, size : 5242880, max : 5000 } )

* db.collection.drop() --> to drop a collection (like db.customer.drop() )
* db.runCommand({"convertToCapped": "collection-name", size: <size>}) 🡺 To convert a non-capped collection into capped collection, like

db.runCommand({"convertToCapped": "mycoll", size: 100000});

* db.runCommand({"convertToCapped": "collection-name", size: <size>}) 🡺 To convert a non-capped collection into capped collection, like

db.runCommand({"convertToCapped": "mycoll", size: 100000});

* db.runCommand({"collMod": "collection-name", cappedSize: <size>, cappedMax: <max>}) 🡺 To change the size and max num of documents for a capped collection

db.runCommand({"convertToCapped": "mycoll", cappedSize: 200000, cappedMax: 500});

CRUD Operations

1. Creating documents

* db.collection.insertOne({}) --> To create document in collection in current db like db.customer.insertOne({ name: "John", age: 31, gender: "male" })
* db.collection.insertMany([{}, {}, {}]) --> To insert many documents
* db.collection.insert() --> Can be used as insertOne or insertMany (but deprecated)

1. Fetching Documents

* db.collection.find() --> To fetch all documents (like db.customer.find() )
* db.collection.findOne() --> To fetch first document (like db.customer.findOne() )
* db.collection.findOne({key: "value"}) --> To fetch first document from collection where key matches value like

db.customer.findOne({name: "Asad"})

* db.collection.find({key: "value"}) --> To fetch all document from collection where condition satisfies like

db.customer.find({gender: "male"}) equivalent to

* db.collection.find({key: "value"}, {key: 0}) --> to return all documents and removing key-values specified in 2nd object like

db.customer.find({gender: "male"}, {\_id: 0, age: 0}) ) --> to return all documents where gender is male and removing \_id, age from results

* db.collection.find().pretty() --> To print in formatted way ( But can only be used with find() not with findOne() )
* db.movies.find( { rated: { $in: [ "PG", "PG-13" ] } } ) equivalent to =>

SELECT \* FROM movies WHERE rated IN ("PG", "PG-13) )

db.collection.find({key: {$op: value}}) 🡺 where $op can be $lt for <, $lte for <=, $gt for >, $gte for >=

Document

* db.inventory.insertMany( [

{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },

{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },

{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },

{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }

]);

Examples

* The following example retrieves all documents in the inventory collection where the status equals "A" and qty is less than ($lt) 30:

db.inventory.find({status: "A", qty: {$lt: 30}})

* The following example retrieves all documents in the inventory collection where the qty is less than ($lt) 30:

db.inventory.find({qty: {$lt: 30})

* The following example retrieves all documents in the collection where the status equals "A" or qty is less than ($lt) 30:

db.inventory.find({$or: [{status: "A" }, {qty: {$lt: 30}}] })

* The following example selects all documents in the collection where the status equals "A" and either qty is less than ($lt) 30 or item starts with the character p:

db.inventory.find( {

status: "A",

$or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]

} )

Query on Embedded/Nested Documents

To specify a query condition on fields in an embedded/nested document, use dot notation ("field.nestedField").

Document

* db.inventory.insertMany( [

{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },

{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },

{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },

{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }

]);

Examples

* The following example selects all documents where the field uom nested in the size field equals "in":

db.inventory.find( { "size.uom": "in" } )

* The following query selects document where the field w embedded in the size field is less than 21:

db.inventory.find({"size.w": {$lt: 21}})

* The following query selects all documents where the nested field h is less than 15, the nested field uom equals "in", and the status field equals "D":

db.inventory.find({"size.h": {$lt: 15}, "size.uom": "in", status: "D"})

* The following query selects all documents where the field size equals the document { h: 14, w: 21, uom: "cm" }:

db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )

# Query an Array

Document

db.inventory.insertMany([

{ item: "journal", qty: 25, tags: ["blank", "red"], dim\_cm: [ 14, 21 ] },

{ item: "notebook", qty: 50, tags: ["red", "blank"], dim\_cm: [ 14, 21 ] },

{ item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim\_cm: [ 14, 21 ] },

{ item: "planner", qty: 75, tags: ["blank", "red"], dim\_cm: [ 22.85, 30 ] },

{ item: "postcard", qty: 45, tags: ["blue"], dim\_cm: [ 10, 15.25 ] }

]);

Examples

* The following example queries for all documents where the field tags value is an array with exactly two elements, "red" and "blank", in the specified order:

db.inventory.find({tags: ["red", "blank"]})

* If, instead, you wish to find an array that contains both the elements "red" and "blank", without regard to order or other elements in the array, use the $all operator:

db.inventory.find( { tags: { $all: ["red", "blank"] } } )

* The following example queries for all documents where tags is an array that contains the string "red" as one of its elements

db.inventory.find({tags: "red"})

* The following operation queries for all documents where the array dim\_cm contains at least one element whose value is greater than 25.

db.inventory.find( { dim\_cm: { $gt: 25 } } )

* The following query selects array where one element can satisfy the greater than 15 condition and another element can satisfy the less than 20 condition, or a single element can satisfy both:

db.inventory.find( { dim\_cm: { $gt: 15, $lt: 20 } } )

* $elemMatch operator: The following example queries for documents where the dim\_cm array contains at least one element that is both greater than ($gt) 22 and less than ($lt) 30:

db.inventory.find( { dim\_cm: { $elemMatch: { $gt: 22, $lt: 30 } } } )

* The following example queries for all documents where the second element in the array dim\_cm is greater than 25:

db.inventory.find({"dim\_cm.1": {$gt: 25}})

* Use the $size operator to query for arrays by number of elements.
* For example, the following selects documents where the array tags has 3 elements

db.inventory.find( {tags: {$size: 3} } )

# Query an Array of Embedded Documents

Document

db.inventory.insertMany( [

{ item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },

{ item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },

{ item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },

{ item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },

{ item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }

]);

Examples

* The following example selects all documents where an element in the instock array matches the specified document:

db.inventory.find( { "instock": { warehouse: "A", qty: 5 } } )

* The following example selects all documents where the instock array has at least one embedded document that contains the field qty whose value is less than or equal to 20:

db.inventory.find( { 'instock.qty': { $lte: 20 } } )

* The following example selects all documents where the instock array has as its first element a document that contains the field qty whose value is less than or equal to 20:

db.inventory.find( { 'instock.0.qty': { $lte: 20 } } )

* The following example queries for documents where the instock array has at least one embedded document that contains both the field qty equal to 5 and the field warehouse equal to A:

db.inventory.find( { "instock": { $elemMatch: { qty: 5, warehouse: "A" } } } )

# Project Fields to Return from Query

* By default, queries in MongoDB return all fields in matching documents. To limit the amount of data that MongoDB sends to applications, you can include a projection document to specify or restrict fields to return.

Document

db.inventory.insertMany( [

{ item: "journal", status: "A", size: { h: 14, w: 21, uom: "cm" }, instock: [ { warehouse: "A", qty: 5 } ] },

{ item: "notebook", status: "A", size: { h: 8.5, w: 11, uom: "in" }, instock: [ { warehouse: "C", qty: 5 } ] },

{ item: "paper", status: "D", size: { h: 8.5, w: 11, uom: "in" }, instock: [ { warehouse: "A", qty: 60 } ] },

{ item: "planner", status: "D", size: { h: 22.85, w: 30, uom: "cm" }, instock: [ { warehouse: "A", qty: 40 } ] },

{ item: "postcard", status: "A", size: { h: 10, w: 15.25, uom: "cm" }, instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }

]);

Examples

* The following operation returns all documents that match the query. In the result set, only the item, status and, by default, the \_id fields return in the matching documents.

db.inventory.find({status: "A"}, {item: 1, status: 1})

* You can remove the \_id field from the results by setting it to 0 in the projection, as in the following example:

db.inventory.find({status: "A"}, {item: 1, status: 1, \_id: 0})

* The following example which returns all fields except for the status and the instock fields in the matching documents:

db.inventory.find( { status: "A" }, { status: 0, instock: 0 } )

* The following example returns:

1. The \_id field (returned by default),
2. The item field,
3. The status field,
4. The uom field in the size document.

The uom field remains embedded in the size document.

db.inventory.find(

{ status: "A" },

{ item: 1, status: 1, "size.uom": 1 }

)

* For fields that contain arrays, MongoDB provides the following projection operators for manipulating arrays: $elemMatch, $slice, and $.
* The following example uses the $slice projection operator to return the last element in the instock array:

db.inventory.find( { status: "A" }, { item: 1, status: 1, instock: { $slice: -1 } } )

# Query for Null or Missing Fields

Document

db.inventory.insertMany([

{ \_id: 1, item: null },

{ \_id: 2 }

])

Examples

The { item : null } query matches documents that either contain the item field whose value is null or that do not contain the item field.

db.inventory.find({status: "A"}, {item: 1, status: 1})

1. Updating Documents

Document

db.inventory.insertMany( [

{ item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },

{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },

{ item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },

{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },

{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },

{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },

{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },

{ item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status: "A" }

] );

Examples

* The following example retrieves all documents in the inventory collection where the status equals "A" and qty is less than ($lt) 30:

db.inventory.updateOne(

{ item: "paper" },

{

$set: { "size.uom": "cm", status: "P" },

$currentDate: { lastModified: true }

}

)

* Above example uses the $currentDate operator to update the value of the lastModified field to the current date.

For multiple documents

db.inventory.updateMany( { "qty": { $lt: 50 } }, { $set: { "size.uom": "in", status: "P" }})

Replace a Document

* To replace the entire content of a document except for the \_id field, pass an entirely new document as the second argument to db.collection.replaceOne().
* The following example replaces the first document from the inventory collection where item: "paper":

db.inventory.replaceOne(

{ item: "paper" },

{ item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }

)

* $unset operator: Used to remove the fields from the document
* The following example removes the size and qty fields from all the documents in the inventory collection

db.inventory.updateMany({}, {$unset: {size: true, qty: true}})

* The following example removes the size and qty fields from all the documents in the inventory collection where qty is 25

db.inventory.updateMany({}, {$unset: {size: true, qty: 25}})

1. Deleting Documents

Queries

* db.collection.deleteOne({}) 🡺 To delete one document
* db.collection.deleteMany({}) 🡺 To delete all documents
* db.collection.deleteOne({key: “value”}) 🡺 To delete one document that matches the condition
* db.collection.deleteOne({key: “value”}) 🡺 To delete all documents that matches the specified condition

Ordered Insertion

* By default, the documents will be inserted in the order you specify and if the error occurs, althogh the document before the document that produced error will be inserted, the documents after it won't be inserted. Like

db.countries.insertMany([{\_id: "PK", name: "Pakistan"}, {\_id: "IN", name: "India"}, {\_id: "GR", name: "Germany"}])

* In above example the documents will be inserted in ordered (like 1st document with \_id "PK" will be inserted, then document with \_id: "IN" and so on)

db.countries.insertMany([{\_id: "TR", name: "Turkey"}, {\_id: "IN", name: "India"}, {\_id: "ENG", name: "England"}])

* In above example, althogh there is error due to same \_id "IN (which was provided earlier), yet document before it (with \_id: "TR") will be inserted but document after it (with \_id: "ENG") won't be inserted
* This is default and safe behavior, but we can overwrite it by specifying "{ordered: false}" to yet insert document even after insert by providing second document and setting ordered to false, like

db.countries.insertMany([{\_id: "TR", name: "Turkey"}, {\_id: "IN", name: "India"}, {\_id: "ENG", name: "England"}], {ordered: false})

writeConcern

* It simply indicates when to send acknowledgement request (like when writing data to only primary server (which is by default) OR only when data is written on all primary and secondary servers bought by us, like
* - db.users.insertOne({name: "Asad"}, {writeConcern: {w: 1}}) or may 2 or 3 depending on how many servers we've and when we wanna send acknowledgement requests

Journal

* When we insert (write a file) it first gets written into RAM and then disk (DB file) but say if the data is written to RAM but then the server went down and the data is not written in Disk, in that case journal has already the to-do list of what to do (in that case, to write data on Disk). The server (when it'll recover) will see in journal and then write data on Disk. This provides extra security from loss of data.

db.users.insertOne({name: "Asad"}, {writeConcern: {w: 1, j: true, wtimeout: 200}}) When set to true data will be written in journal (althogh now the insertion process will take more time but provides extra security) and wtimeout value specifies that it should cancel request of writing data after the time exceeds given time in wtimeout and this time should be more

Aggregation Stages

1. Sorting

The $sort stage has the following prototype form:

**Syntax:**

{ $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }

$sort

takes a document that specifies the field(s) to sort by and the respective sort order. <sort order> can have one of the following values:

|  |  |
| --- | --- |
| **Value** | **Description** |
| 1 | Sorts ascendingly |
| -1 | Sorts descending |

Document

db.restaurants.insertMany( [

{ "\_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan"},

{ "\_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens"},

{ "\_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn"},

{ "\_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan"},

{ "\_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn"},

] )

Examples

* The following command uses the $sort stage to sort on the borough field:
* db.restaurants.aggregate( [ {$sort: {borough: 1} }] )
* This operation sorts the documents in the users collection, in descending order according by the age field and then in ascending order according to the value in the posts field.
* db.restaurants.aggregate( [ {$sort: {borough: 1} }] )

1. $skip

Used to skip the first n documents

Document

db.inventory.insertMany([

{ item: 'canvas', status: 'A' },

{ item: 'journal', status: 'A' },

{ item: 'mat', status: 'A' },

{ item: 'mousepad', status: 'P' },

{ item: 'notebook', status: 'P' },

{ item: 'paper', status: 'D' },

{ item: 'planner', status: 'D' },

{ item: 'postcard', status: 'A' },

{ item: 'sketchbook', status: 'A' },

{ item: 'sketch pad', status: 'A' }

])

* The following query skips the first 7 documents

db.inventory.aggregate( [ {$skip: 7} ] )

Imp Aggregation Operators

Document

db.employees.insertMany([

{ "name": "Alice", "age": 25, "city": "New York", "salary": 50000, "department": "Sales"},

{ "name": "Bob", "age": 30, "city": "Los Angeles", "salary": 60000, "department": "Marketing" },

{ "name": "Charlie", "age": 28, "city": "San Francisco", "salary": 55000, "department": "Sales" },

{ "name": "David", "age": 35, "city": "Chicago", "salary": 70000, "department": "Engineering" },

{ "name": "Emma", "age": 27, "city": "Seattle", "salary": 52000, "department": "Marketing" },

{ "name": "Frank", "age": 32, "city": "Boston", "salary": 65000, "department": "Engineering" },

{ "name": "Grace", "age": 29, "city": "Austin", "salary": 53000, "department": "Sales" },

{ "name": "Hannah", "age": 26, "city": "Denver", "salary": 51000, "department": "Marketing" },

{ "name": "Ian", "age": 31, "city": "Miami", "salary": 67000, "department": "Engineering" },

{ "name": "Julia", "age": 33, "city": "Philadelphia", "salary": 68000, "department": "Engineering" },

{ "name": "Kevin", "age": 24, "city": "Phoenix", "salary": 48000, "department": "Sales" },

{ "name": "Linda", "age": 34, "city": "San Diego", "salary": 69000, "department": "Engineering" },

{ "name": "Mike", "age": 27, "city": "Portland", "salary": 54000, "department": "Marketing" },

{ "name": "Nancy", "age": 28, "city": "Houston", "salary": 56000, "department": "Sales" },

{ "name": "Oliver", "age": 29, "city": "Atlanta", "salary": 57000, "department": "Marketing" },

{ "name": "Pamela", "age": 30, "city": "Detroit", "salary": 58000, "department": "Sales" },

{ "name": "Quentin", "age": 31, "city": "Minneapolis", "salary": 59000, "department": "Engineering" },

{ "name": "Rachel", "age": 32, "city": "Las Vegas", "salary": 60000, "department": "Marketing" },

{ "name": "Samantha", "age": 33, "city": "Charlotte", "salary": 61000, "department": "Engineering" },

{ "name": "Tyler", "age": 34, "city": "Washington D.C.", "salary": 62000, "department": "Sales" }

])

1. $abs
2. $limit

db.employees.aggregate({$limit: 4})

db.employees.aggregate([{$skip: 4}, {$limit: 1}])

1. $skip
2. $add
3. $sum
4. $subtract
5. $min
6. $max
7. $sort