Starting Mongo DB:

**Chapter 1:**

**Basic Facts**:

1. MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++.
2. MongoDB works on concept of collection and document.

**Definitions**:

1. **Database**

* Database is a physical container for collections.
* Each database gets its own set of files on the file system.
* A single MongoDB server typically has multiple databases.

## Collection

* Collection is a group of MongoDB documents (same as RDBMS table).
* A collection exists within a single database.

## Document

* A document is a set of key-value pairs.
* Documents have dynamic schema.
* Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Chapter 2: Basic Commands

1. The use Command

* MongoDB **use DATABASE\_NAME** is used to create database, If it doesn't exist, otherwise it will return the existing database.
* To check your currently selected database, use the command **db.**
* If you want to check your databases list, use the command **show dbs**.
* Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

2. The dropDatabase() Method

* MongoDB **db.dropDatabase()** command is used to drop a existing database.

Example:

use myDB

Db.dropDatabase();

3. The createCollection() Method

* MongoDB **db.createCollection(name, options)** is used to create collection.

use test

switched to db test

db.createCollection("mycollection");

* You can check the created collection by using the command show collections.

show collections

mycollection

system.indexes

1. The drop() Method

* MongoDB's **db.collection.drop()** is used to drop a collection from the database.

db.COLLECTION\_NAME.drop().

1. The insert() Method

* To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

db.mycol.insert({

\_id: ObjectId(7df78ad8902c),

title: 'MongoDB Overview'

})

* In the inserted document, if we don't specify the \_id parameter, then MongoDB assigns a unique ObjectId for this document.
* \_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

\_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer).

* To insert multiple documents in a single query, you can pass an array of documents in insert() command.

db.mycol.insert([{ ..},{… }]);

1. The find() Method

* To query data from MongoDB collection, you need to use MongoDB's **find().**

>db.COLLECTION\_NAME.find()

* To display the results in a formatted way, you can use **pretty()** method.

>db.mycol.find().pretty()

1. RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 in MongoDB

### Syntax

In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition. Following is the basic syntax of **AND** −

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

## OR in MongoDB

### Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

## Using AND and OR Together

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

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

## MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax: db.COLLECTION\_NAME.update(SELECTION\_CRITERIA, UPDATED\_DATA)

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

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

## MongoDB Save() Method

The **save()** method replaces the existing document with the new document passed in the save() method.

>db.mycol.save(

{

"\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic",

"by":"Tutorials Point"

}

)

## The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection.

>db.mycol.remove({'title':'MongoDB Overview'})

## Remove Only One

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

## Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

>db.mycol.remove()

>db.mycol.find()

MongoDB – Projection

when you execute **find()** method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

>db.mycol.find({},{"title":1,\_id:0})

This example will only display the title of the document not id.

# MongoDB - Limit Records

Following example will display only two documents while querying the document.

>db.mycol.find({},{"title":1,\_id:0}).limit(2)

**skip()** which also accepts number type argument and is used to skip the number of documents.

>db.mycol.find({},{"title":1,\_id:0}).limit(1).skip(1)

# MongoDB - Sort Records

To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

# MongoDB - Aggregation

For the aggregation in MongoDB, you should use **aggregate()** method.

Following is a list of available aggregation expressions.

|  |  |  |
| --- | --- | --- |
| **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"}}}])  It will provide the sum of “likes” column,and group by the column: ”by\_user” |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate([{$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"}}}]) |

## Pipeline Concept

MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework −

* **$project** − Used to select some specific fields from a collection.
* Consider a books collection with the following document:

{

"\_id" : 1,

title: "abc123",

isbn: "0001122223334",

author: { last: "zzz", first: "aaa" },

copies: 5

}

* The following $project stage includes only the \_id, title, and the author fields in its output documents:

db.books.aggregate( [ { $project : { title : 1 , author : 1 } } ] )

* The operation results in the following document:

{ "\_id" : 1, "title" : "abc123", "author" : { "last" : "zzz", "first" : "aaa" } }

* **$match** − This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.

## Examples

The examples use a collection named articles with the following documents:

{ "\_id" : ObjectId("512bc95fe835e68f199c8686"), "author" : "dave", "score" : 80, "views" : 100 }

{ "\_id" : ObjectId("512bc962e835e68f199c8687"), "author" : "dave", "score" : 85, "views" : 521 }

{ "\_id" : ObjectId("55f5a192d4bede9ac365b257"), "author" : "ahn", "score" : 60, "views" : 1000 }

{ "\_id" : ObjectId("55f5a192d4bede9ac365b258"), "author" : "li", "score" : 55, "views" : 5000 }

{ "\_id" : ObjectId("55f5a1d3d4bede9ac365b259"), "author" : "annT", "score" : 60, "views" : 50 }

{ "\_id" : ObjectId("55f5a1d3d4bede9ac365b25a"), "author" : "li", "score" : 94, "views" : 999 }

{ "\_id" : ObjectId("55f5a1d3d4bede9ac365b25b"), "author" : "ty", "score" : 95, "views" : 1000 }

* db.articles.aggregate(
* [ { $match : { author : "dave" } } ]
* );
* { "\_id" : ObjectId("512bc95fe835e68f199c8686"), "author" : "dave", "score" : 80, "views" : 100 }
* { "\_id" : ObjectId("512bc962e835e68f199c8687"), "author" : "dave", "score" : 85, "views" : 521 }
* **$group** − This does the actual aggregation as discussed above.
* **$sort** − Sorts the documents.

### Ascending/Descending Sort

For the field or fields to sort by, set the sort order to 1 or -1 to specify an ascending or descending sort respectively, as in the following example:

db.users.aggregate(

[

{ $sort : { age : -1, posts: 1 } }

]

)

* **$skip** − With this, it is possible to skip forward in the list of documents for a given amount of documents.
* **$limit** − This limits the amount of documents to look at, by the given number starting from the current positions.
* db.article.aggregate(
* { $limit : 5 }
* );
* **$unwind** − This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

### Unwind Array

Consider an inventory with the following document:

{ "\_id" : 1, "item" : "ABC1", sizes: [ "S", "M", "L"] }

The following aggregation uses the $unwind stage to output a document for each element in the sizes array:

db.inventory.aggregate( [ { $unwind : "$sizes" } ] )

The operation returns the following results:

{ "\_id" : 1, "item" : "ABC1", "sizes" : "S" }

{ "\_id" : 1, "item" : "ABC1", "sizes" : "M" }

{ "\_id" : 1, "item" : "ABC1", "sizes" : "L" }

* 1. Count

The simplest aggregation tool is count , which returns the number of documents in the

collection:

> db.foo.count()

2. distinct

The distinct command finds all of the distinct values for a given key. You must specify

a collection and key:

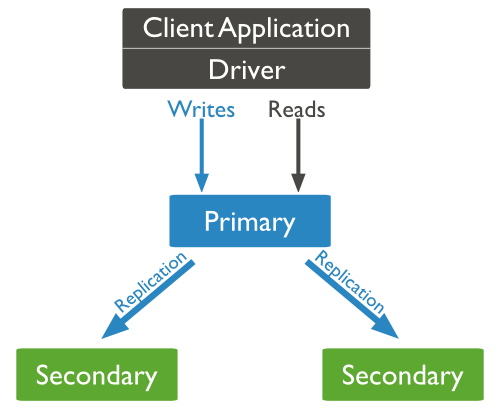
> db.runCommand({"distinct" : "people", "key" : "age"})

# MongoDB - Replication

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers.

MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set.

* Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
* In a replica set, one node is primary node and remaining nodes are secondary.
* All data replicates from primary to secondary node.
* At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
* After the recovery of failed node, it again join the replica set and works as a secondary node.



## Set Up a Replica Set

* Shutdown already running MongoDB server.
* Start the MongoDB server by specifying -- replSet option. Following is the basic syntax of --replSet −

mongod --port "PORT" --dbpath "YOUR\_DB\_DATA\_PATH" --replSet "REPLICA\_SET\_INSTANCE\_NAME"

### Example

mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0