MongoDB

## Introduction

This document covers

Introduction

### General

|  |  |
| --- | --- |
|  | Column Header |
| Binaries | C:\Program Files\MongoDB\Server\4.4\bin |
| Default End Point | localhost:27017 |
| Dump | mongodump --uri="mongodb://localhost:27017" --archive=myarchive --gzip |
| Restore | mongorestore --uri="mongodb://localhost:27017" --drop --archive=myarchive --gzip |

### Basic Commands

|  |  |
| --- | --- |
|  | Column Header |
| Create database | use mydb; |
| Create collection | db.createCollection("c") |
| List collection names | show collections |
| List databases | show dbs |
| Insert one doc | db.c.insertOne( { nm: "Jo"} ); |
| Insert many docs | db.c.insertMany |
| Get all docs | db.c.find() |
| Drop collections | db.c.drop(); |
| Delete single document | db.c.deleteOne( {nm:”Joe”}) |

### Queries

### Embedded Documents

db.c.find({“person.sex”:"male", “person.first”:john}, {nm:1,\_id:0})

### And Filter

The following only return male whose name is don.

##### Query

db.c.find({sex:"male", nm:"don"}, {nm:1,\_id:0})

##### Result

{ "nm" : "don" }

### Conditionals

We can list all the people whose age is between 21 and 50 inclusive as follows.

##### Input collections

{ "nm" : "ken", "age" : 45 }

{ "nm" : "kim", "age" : 23 }

{ "nm" : "jon", "age" : 60 }

##### Query

db.c.find({ age: {"$gte" : 23, "$lte" : 50}}, {\_id:0})

##### Result

{ "nm" : "ken", "age" : 45 }

{ "nm" : "kim", "age" : 23 }

### In

We can list the 23- and 45-years old.

##### Input collections

{ "nm" : "ken", "age" : 45 }

{ "nm" : "kim", "age" : 23 }

{ "nm" : "jon", "age" : 60 }

##### Query

db.c.find({ age: {"$in" : [23,45]}}, {\_id:0})

##### Result

{ "nm" : "ken", "age" : 45 }

{ "nm" : "kim", "age" : 23 }

### Logical OR

We can list anyone who is 23 or whose name is jon as follows

##### Input Collection

{ "nm" : "kim", "age" : 23 }

{ "nm" : "jon", "age" : 60 }

##### Query

{ "nm" : "ken", "age" : 45 }

{ "nm" : "kim", "age" : 23 }

{ "nm" : "jon", "age" : 60 }

##### Result

db.c.find({ "$or" : [ {age:23}, {nm: "jon"}]}, {\_id:0})

### Null

Consider the following

##### Input Collection

{ "x" : 1, "nm" : null }

{ "x" : 2 }

{ "x" : 3, "nm" : "ken" }

Then checking for null returns the documents that have null for that field value or for which the field is not set at all

##### Query

db.c.find({nm:null}, {\_id:0})

##### Results

{ "x" : 1, "nm" : null }

{ "x" : 2 }

If we don’t want the documents which don’t even have that key we do this

##### Query

db.c.find({nm: {"$eq" :null, "$exists":true}}, {\_id:0})

##### Results

{ "x" : 1, "nm" : null }

#### Searching Arrays

##### Single element match

Image we have this

{ "fruit" : [ "apple", "orange", "pear" ] }

{ "fruit" : [ "apple", "banana" ] }

{ "fruit" : [ "banana", "kiwi" ] }

Then we can find all documents whose fruit array contains apple as follows

db.c1.find({fruit: "apple"},{\_id:0})

Which gives us

{ "fruit" : [ "apple", "orange", "pear" ] }

{ "fruit" : [ "apple", "banana" ] }

##### $all – multi element match

Image we have this

{ "fruit" : [ "apple", "orange", "pear" ] }

{ "fruit" : [ "apple", "banana" ] }

{ "fruit" : [ "banana", "kiwi" ] }

We can find all documents whose fruit field’s array contains both the values apple and pear

> db.c1.find({fruit: {"$all" : [ "apple", "pear"]}},{\_id:0})

Which gives us

{ "fruit" : [ "apple", "orange", "pear" ] }

##### Array equality

If we want to perform an exact match, we can use array equality. Imagine we have this

{ "fruit" : [ "apple", "orange", "pear" ] }

{ "fruit" : [ "apple", "banana" ] }

{ "fruit" : [ "banana", "kiwi" ] }

We can do an exact match as follows.

db.c1.find({fruit: ["banana", "kiwi"]},{\_id:0})

{ "fruit" : [ "banana", "kiwi" ] }

Which gives us

{ "fruit" : [ "apple", "pear" ] }

##### Query Size

We can query for all documents whose fields array value is of a certain length

db.fruit.find({fruit: {"$size":2}} , {\_id:0})

{ "fruit" : [ "apple", "pear" ] }

Imagine we have this

{ "fruit" : [ "apple", "orange", "pear" ] }

{ "fruit" : [ "apple", "banana" ] }

{ "fruit" : [ "banana", "kiwi" ] }

We can find all documents whose fruit field is an array of size 2

db.c1.find({fruit: {"$size":2} },{\_id:0})

Which gives us

{ "fruit" : [ "apple", "banana" ] }

{ "fruit" : [ "banana", "kiwi" ] }

##### Slice

A positive number returns first n elements so given

{ "x" : [ 10, 11, 12, 13, 14, 15, 15 ] }

We can take first 3 elements in the result

db.nums.find({}, {x:{"$slice":3}})

Giving

{ "\_id" : ObjectId("5ec25a6b9d4450d02edd4142"), "x" : [ 10, 11, 12 ] }

A negative number returns last three

db.nums.find({}, {x:{"$slice":-3}})

Giving

{ "\_id" : ObjectId("5ec25a6b9d4450d02edd4142"), "x" : [ 14, 15, 16 ] }

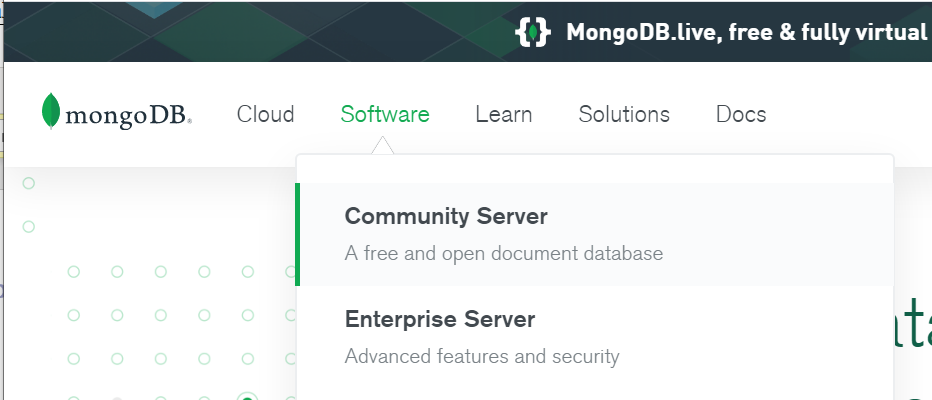
##### Indexing

We can index into arrays

db.nums.find({"x.1":11})

## Installation

Install the community server from mongodb.com.



The binaries are installed to the following location which we should add to our path.

C:\Program Files\MongoDB\Server\4.4\bin

I have chosen to install it into C:\Users\rps\Code\temp\mongo . I have also chosen custom and decided not to install it as a service as I am just playing around. Now we need to create a folder for data which I am going to call C:\Users\rps\Code\temp\mongo\Data Finally open a command prompt as an administrator and run the command

mongod --dbpath C:\Users\rps\Code\temp\mongo\Data

By default, MongoDB listens on port 27017

## The Shell

Assuming we have ran the server as per the previous section we can open the mongo shell. The shell is a complete JavaScript interpreter in addition to its mongo role. We can list the current database as follows.

db

>> test

To create a new database if it does not exist use the use command. If it does exist the use command will switch to that database. The database is not actually created until you first create a collection and insert a value into it. Before we enter the following command there is no database called mydatabase and no collection called acollect.

use mydatabase