**Why NoSQL**

* Offers rich query language
* Easy Saclability
* No Schema Required Non relational
* Highlty Distributable
* High performace with high availabiltiiy

**Why NoSQL**

* **Schema agnostic**
* A db schema is the description of all possible data and adata structures in a rellational database.
* Commodity hardware
* Highly Distributable
* MongoDB is a document oriented Database.

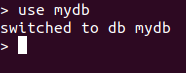
Commands in Mong Shell

**db** – To know which db is selected (By default you are connected to test database)



To create a new database:

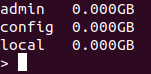
**use <database\_name>**

****

It will show that it switches to new database, but it will not create a database until you insert a document into it

To show all the databases:

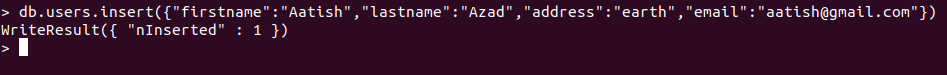
**show dbs**

****

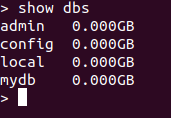
**Mongo stores data in collections of documents**

To Create a collection of data:

db.<collection\_name>.insert({json\_object})

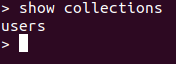
****

After inserting a document in a collectio by name users. Mongo will automatically create a database by name mydb

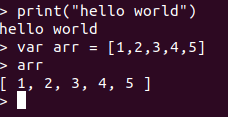


To show a list of collection in mongo db:

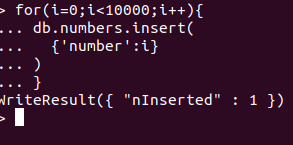
**show collections**

****

**Mongo comes wiht javascript interpreter:**

****

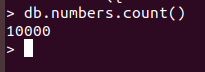
Mixing Javascript with Mongo, to show flexibility of MongoDB



In the above example we have inserted 10,000 nubers in a collection by name nubmer

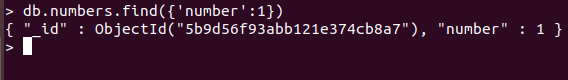
To show number of documents in collection:

**db.numbers.count()**



To find a document:

**db.numbers.find({‘number’:1})**

****

Let ask mongo to explain, how it find the numbers:

**db.numbers.find({'number':1}).explain()**



To get more details:

**db.numbers.find({'number':1}).explain('executionStats')**

****

It took 7 milli seconds to examine the 10000 docs

This is not good

Let’s add a simple index to change the outcome:

**db.numbers.find({'number':1}).explain('executionStats')**



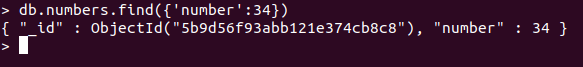
Importing a CSV file in mongo DB:

**mongoimport --db** <database\_name> **--collection** <collection\_name> **--type csv --headerline --file** <file\_name>

****

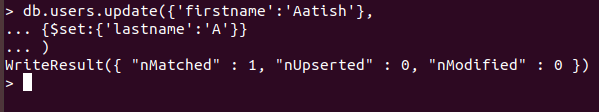
Find operation in Mongo:

**db.numbers.find({“number”:34})**

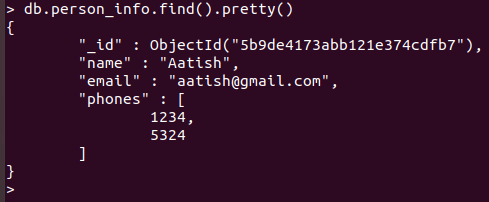
****

To Update a document in Mongo:

**db.users.update({'firstname':'Aatish'}, {$set:{'lastname':'A'}} )**

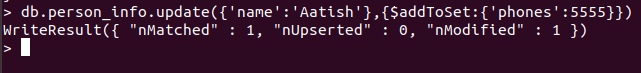
****

**Adding data to an array type field:**



To Update array field type:

**db.<collection\_name>.update({json\_object\_field},{$addToSet:{‘arrField’:’data’}})**

****

To Remove a data from the collection:

**db.<collection\_name>.remove({json\_obj})**

****

To Drop a collection:

**db.<collection\_name>.drop()**

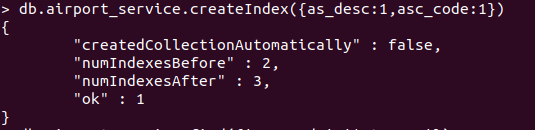
**Indexing in Mongo**

**db.<collection\_name>.createIndex({field:1})**

**eg: db.airport\_service.createIndex({as\_desc:1})**

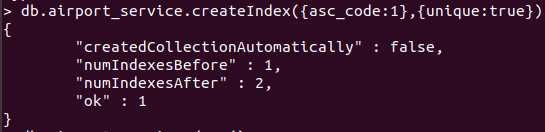
**Indexing is use in mongo db to enhance the search operation. By creating an index for a collection. Mongo Creates a lookup table for faster search operations on a given field.**

Creating a compound index in mongo db



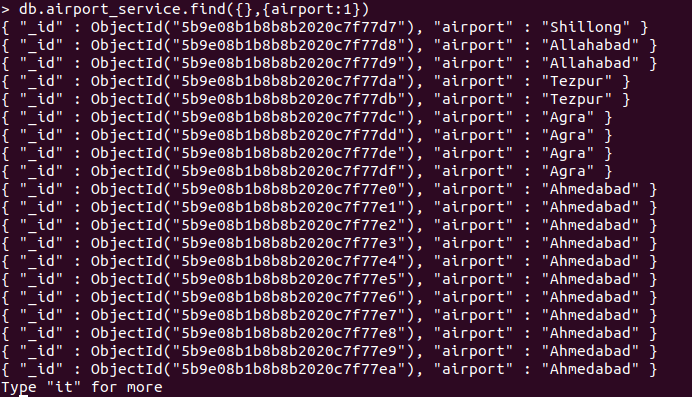
**Note:** In a collection there can be 64 indexes, but not more than that. But in real time you won’t need more than 3 to 4 Indexes.

When working with a database, frequently you need to be able to impose constraints on the data. The most common use case is requiring that a particular field be unique across the collection. The underscore ID field can be set when the document is inserted, but in many cases, you'll want to be able to impose this kind of restriction on other fields.

****

Creating unique indexes allows you to impose constraints on your data so that you can maintain consistency and work with the data confidently.

MongoDB supports projections, which allow you to specify which pieces of the documents you want to see. MongoDB supports projections, which allow you to specify which pieces of the documents you want to see.



It turns out that Mongo's always going to return this field, unless you specify that it should be excluded. So let's do that by extending the query.



let's grab a couple more fields



To make it more readable use the method: **pretty()**

if you look at the bottom of the response, you see Type it for more, which indicates that not all of the documents were returned. We're going to need to implement some kind of sorting.

//for acending

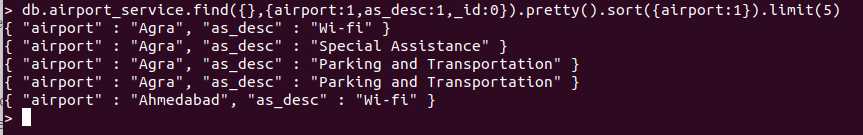
**db.airport\_service.find({},{airport:1,as\_desc:1,\_id:0}).pretty().sort({airport:1})**

//for decending

**db.airport\_service.find({},{airport:1,as\_desc:1,\_id:0}).pretty().sort({airport:-1})**

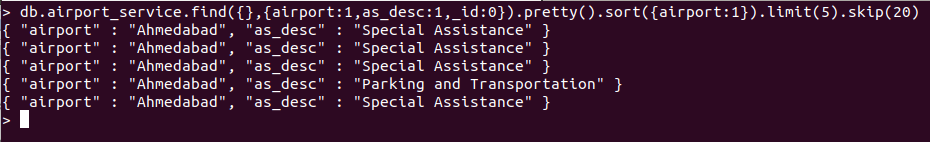
Mongo supports a Limit Operator as well and you can chain it right along with the other functions we have.

**db.airport\_service.find({},{airport:1,as\_desc:1,\_id:0}).pretty().sort({airport:1}).limit(5)**



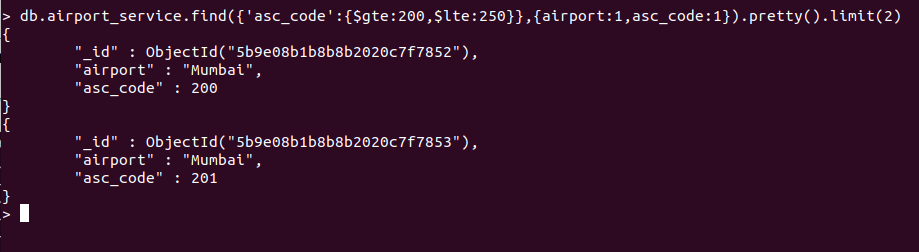
It's frequently necessary to implement paging, so you can add skip to your query to pick documents, other than the first ones returned. Again, we can just chain it on here, skip 20

**db.airport\_service.find({},{airport:1,as\_desc:1,\_id:0}).pretty().sort({airport:1}).limit(5).skip(20)**



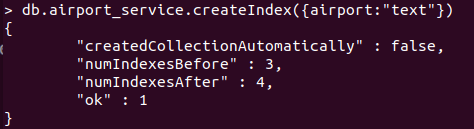
You can use these comparison operators to specify exactly the range of documents you want to see.

**db.airport\_service.find({'asc\_code':{$gte:200,$lte:250}},{airport:1,asc\_code:1}).pretty().limit(2)**

****

**Creating text indexes**

**db.airport\_service.createIndex({airport:”text”})**

****

Searching for text data:

**db.airport\_service.find({$text:{$search:'Bengaluru'}}).pretty()**

****

**Importing to Atlas**

mongoimport --uri "mongodb://root:root@atlas-host1:27017,atlas-host2:27017,atlas-host3:27017/mydb?ssl=true&replicaSet=myAtlasRS&authSource=admin" --collection air\_service --drop --type csv --headerline --file C:\Users\user\Downloads\air\_sewa\_airport\_services.csv