mongodb:

nosql db, written in c++, cross-platform, document and collection oriented db

Supports only json data.

collection: group of documents. like rdbms table. A collection exist in a single db. They do not enforce a schema. documents in a collection can have diff fields.All documents in a collection have similar or related purpose.

**Run mongodb on your machine:**

1. <https://docs.mongodb.com/manual/installation/>
2. Mongo.exe: cli
3. Mongod.exe: mongo database
4. Mongodump: to dump binary
5. Mongoexport: to export to csv, json
6. Mongorestore: to restore exported
7. Mongoimport:
8. Mongostst: status og mongodb

Cmd: goto bin pf mongodb

Mongodb server

Database0 DatabaseN

Coll0 col1 ColN Col0 Col1 ColN (Collections)

1. Directory to store files: /data/db:

Mongod –dbpath C:/Temp : to change the location of directory.

x

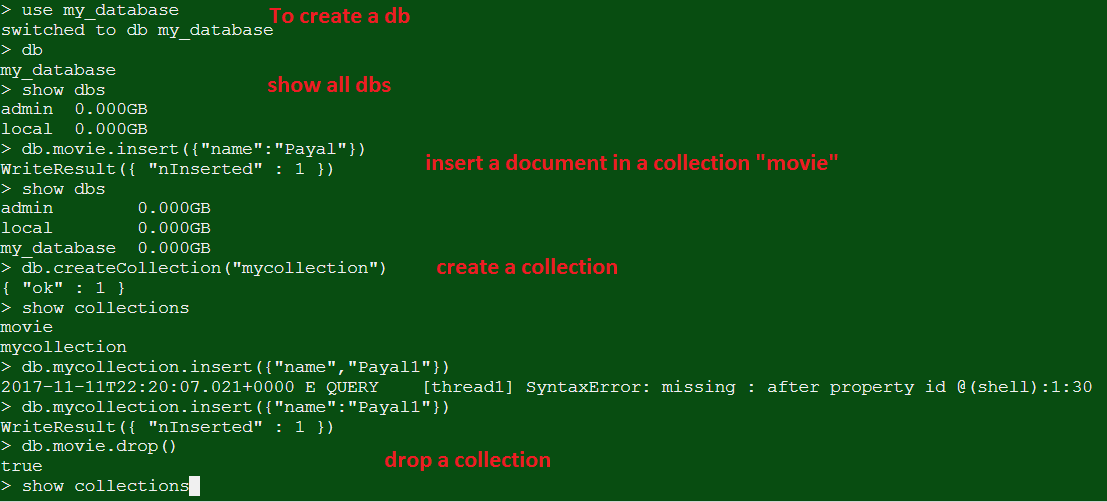
document: key value pair. documents have dynamic schema. like a row in a table.

field: like a column

primary key: default key\_id provided by mongodb. 12 bytes hexadecimal value. first 4 bytes: timestamp, next 3: machine id, next 2 : process id, last 3: incremental value.

No relationship concept. NO complex joins. Mapping of application object to db objects not needed.

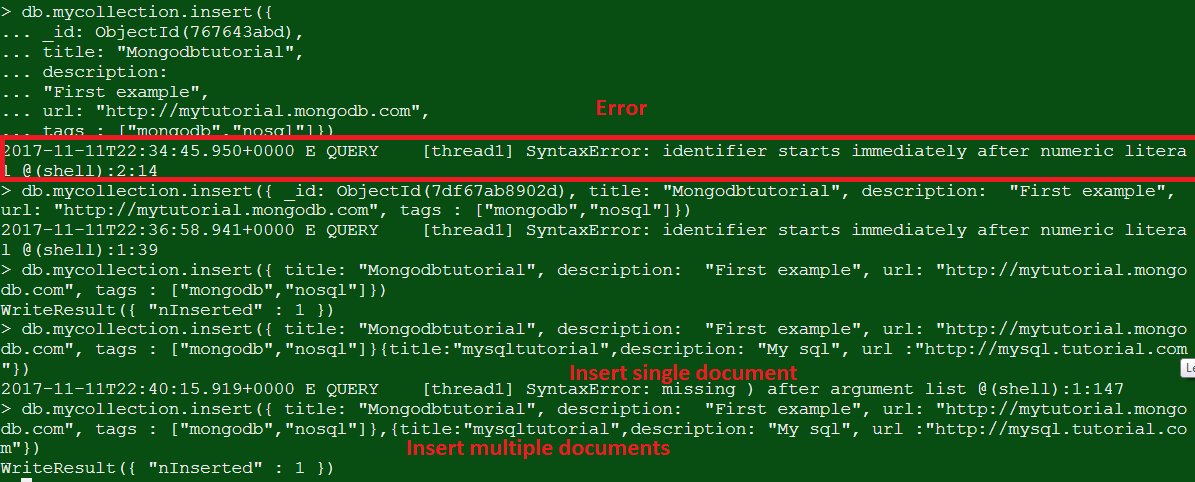
Data stored in form of json , index on any attribute.



DataTypes:

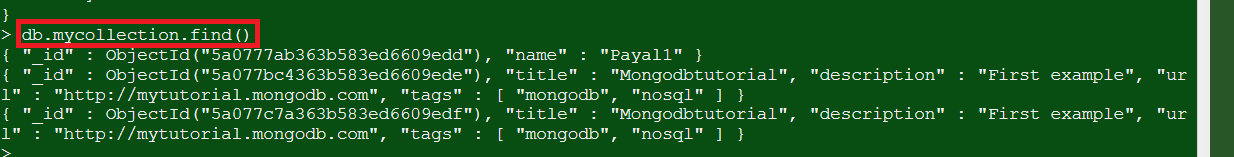
1. String
2. Integer (32 or 64 bit, depends on server)
3. Boolean
4. Double (decimal value)
5. Min/Max keys (to compare value against lowest and highest bson elements :binary json
6. Arrays: 1 key multiple values.
7. Timestamp
8. Object: for embedded documents
9. Null
10. Symbol
11. Date
12. Object ID
13. Binary data
14. Code: to store javascript code
15. Regular expression
16. Insert() | save() method: db.collection\_name.insert(document);

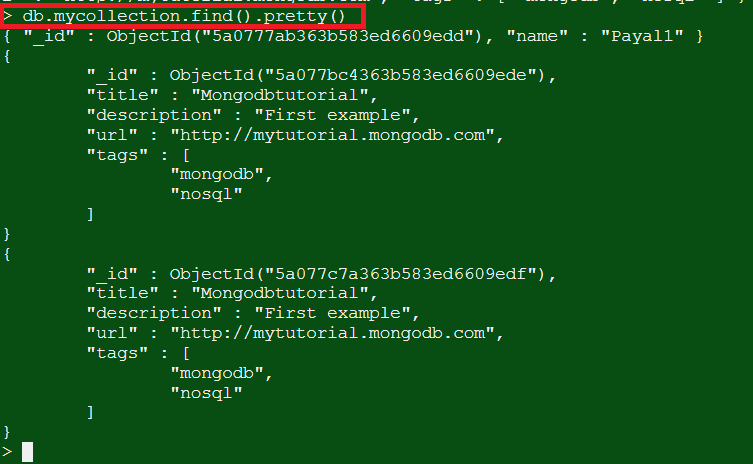
Save can be used to update a document passing in \_id. Else same as insert if no id provided.



1. Find(): to get data from collection.

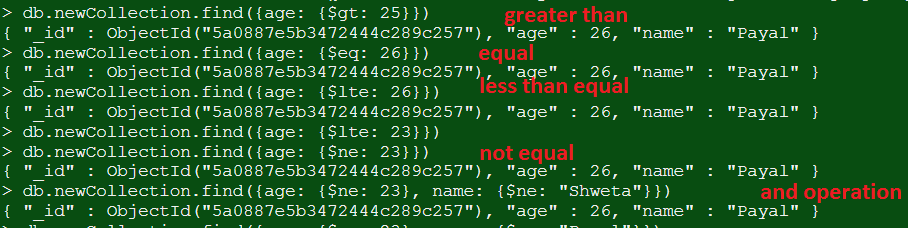
Pretty(): to display data in formatted way

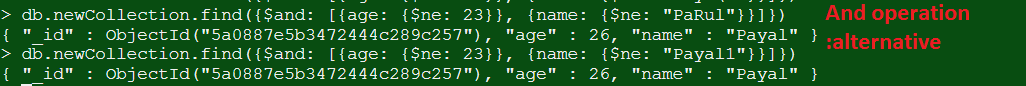




1. Query document on the basis of where clause:



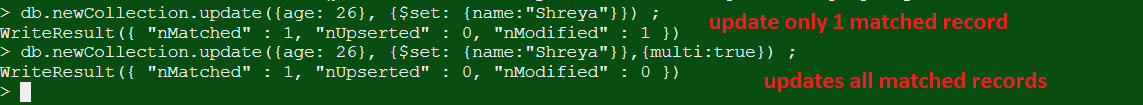




**'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

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

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



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

>db.mycol.find()

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

The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

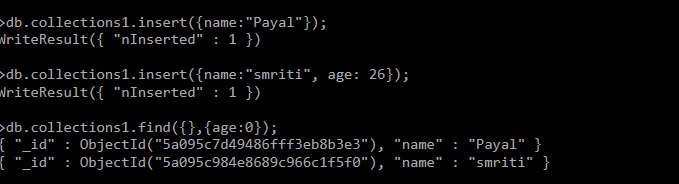
* **deletion criteria** − (Optional) deletion criteria according to documents will be removed.
* **justOne** − (Optional) if set to true or 1, then remove only one document.

If does not pass any value in the second argument, will by-default delete all arguments that matches criteria. Else if we pass 1, will delete just one. If we pass no parameters, will truncate the data in the collection. Will remove all documents.

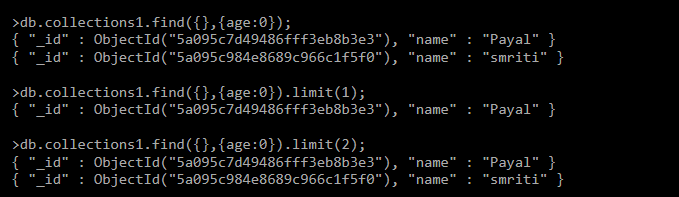
>db.mycol.remove()

>db.mycol.find()

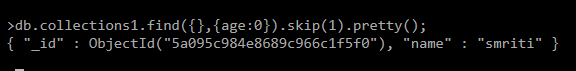
**Projection**: selecting only some fields. In second parameter for find, add 1 or 0 along with the key to specify which field to be selected as 1 and which should be removed as 0



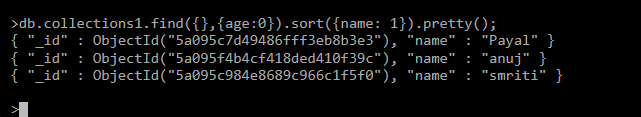
**Limit:** Takes 1 argument with number of documents to be displayed.

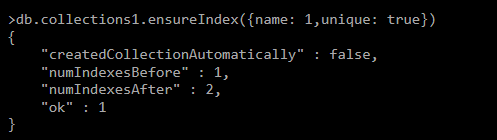


**Skip:** Takes 1 argument with number of documents to be skipped. Default value 0 and if skip has argument as1, will skip the first document.

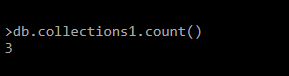


**Sort:** If 1 in ascending order, if -1 in descending order.



**Indexing:** To create an index use ensureIndex() method. 1 means in ascending order, -1 in descending order. 

**Aggregation Operations:**  aggregate () method. The **aggregate** function accepts an array of data transformations which are applied to the data in the order they're defined. This makes aggregation a lot like other data flow pipelines: the transformations that are defined first will be executed first and the result will be used by the next transformation in the sequence. The first stage of the pipeline is matching, and that allows us to filter out documents so that we're only manipulating the documents we care about. The matching expression looks and acts much like the MongoDB find function or a SQL **WHERE** clause



findOne method which finds the first document in the collection.

