MongoDb & Python Essentials

mongodb.md

##MONGODB & PYTHON

###Ubuntu Install

```
sudo apt-get install mongodb
pip install pymongo
```

Table - Collection
Column - Property
Row - Document
Node - Single Instance of the MongoDB daemon process

###Connecting to MongoDB

```
""" An example of how to connect to MongoDB """
import sys
from pymongo import Connection
from pymongo.errors import ConnectionFailure

def main():
    """ Connect to MongoDB """
    try:
        c = Connection(host="localhost", port=27017)
        print "Connected successfully"
        print c
    except ConnectionFailure, e:
        sys.stderr.write("Could not connect to MongoDB: %s" % e)
        sys.exit(1)
if __name__ == "__main__": main()
```

###Getting a Database Handle

```
""" An example of how to get a Python handle to a MongoDB database """
import sys
from pymongo import Connection
from pymongo.errors import ConnectionFailure
def main():
    """ Connect to MongoDB """
    try:
        c = Connection(host="localhost", port=27017)
    except ConnectionFailure, e:
        sys.stderr.write("Could not connect to MongoDB: %s" % e)
        sys.exit(1)
    # Get a Database handle to a database named "mydb"
    dbh = c["mydb"]
    # Demonstrate the db.connection property to retrieve a reference to the
    # Connection object should it go out of scope. In most cases, keeping a
    # reference to the Database object for the lifetime of your program should
    # be sufficient.
    assert dbh.connection == c
```

```
    print "Successfully set up a database handle"
if __name__ == "__main__": main()
```

###Insert a Document Into a Collection

```
""" An example of how to insert a document """
import sys
from datetime import datetime
from pymongo import Connection
from pymongo.errors import ConnectionFailure
def main():
    try:
        c = Connection(host="localhost", port=27017)
    except ConnectionFailure, e:
        sys.stderr.write("Could not connect to MongoDB: %s" % e)
        sys.exit(1)
    dbh = c["mydb"]
    assert dbh.connection == c
    user_doc = {
        "username" : "janedoe",
        "firstname" : "Jane",
        "surname" : "Doe",
        "dateofbirth" : datetime(1974, 4, 12),
        "email" : "janedoe74@example.com",
        "score" : 0
    }
    dbh.users.insert(user_doc, safe=True)
    print "Successfully inserted document: %s" % user_doc
if __name__ == "__main__": main()
```

safe=True ensures that your write
will succeed or an exception will be thrown
dbh.users.insert(user_doc, safe=True)

w=2 means the write will not succeed until it has
been written to at least 2 servers in a replica set.
dbh.users.insert(user_doc, w=2)

###Query Language #####Retrieve single document

```
user_doc = dbh.users.find_one({"username" : "janedoe"})
print user_doc
if not user_doc:
  print "no document found for username janedoe"
```

#####Retrieve all documents

```
users = dbh.users.find({"username" : "janedoe"})
if not users:
  print "no document found for username janedoe"
else:
  for user in users:
    print user.get("email")
```

#####Retrieve a subset of properties from each document

```
users = dbh.users.find({"firstname":"jane"}, {"email":1})
for user in users:
  print user.get("email")
```

#####Count

```
userscount = dbh.users.find().count()
print "There are %d documents in users collection" % userscount
/*--or--*/
db.users.count({'name.first':'John'});
```

#####Sort

```
import pymongo
###
users = dbh.users.find().sort("dateofbirth", pymongo.DESCENDING)
for user in users:
  print user.get("email")
```

#####Limit

```
users = dbh.users.find().sort("score", pymongo.DESCENDING).limit(10)
for user in users:
  print user.get("username"), user.get("score", 0)
```

#####Skip

```
users = dbh.users.find().sort("username",pymongo.DESCENDING).limit(2).skip(1)
for user in users:
  print user.get("username")
```

#####Get rid of duplicates

```
for user in dbh.users.find(snapshot=True):
```

###Updating Documents

```
import copy
# first query to get a copy of the current document
old_user_doc = dbh.users.find_one({"username":"janedoe"})
new_user_doc = copy.deepcopy(old_user_doc)
# modify the copy to change the email address
new_user_doc["email"] = "janedoe74@example3.com"
# run the update query
# replace the matched document with the contents of new_user_doc
dbh.users.update({"username":"janedoe"}, new_user_doc, safe=True)
```

#####Update Modifiers

```
dbh.users.update({"username":"janedoe"},
  {"$set":{"email":"janedoe74@example2.com"}}, safe=True)
# For multiple properties
dbh.users.update({"username":"janedoe"}, {"$set":{"email":"janedoe74@example2.com", "score":1}},
safe=True)
```

In order to have your update query write multiple documents, you must pass the “multi=True” parameter to the update method.

```
dbh.users.update({"score":0},{"$set":{"flagged":True}}, multi=True, safe=True)
```

###Deleting Documents

```
dbh.users.remove({"score":1}, safe=True)
# Delete all documents in user collection dbh.users.remove(None, safe=True)
```

###Common operations on sub-documents embedded in a list #####$pull

```
# Atomically remove an email address from a user document race-free using the # $pull update modifier
user_doc = {
  "username":"foouser",
  "emails":[
    {
      "email":"foouser1@example.com",
      "primary":True
    },{
      "email":"foouser2@example2.com",
      "primary":False
    },{
      "email":"foouser3@example3.com",
      "primary":False
    }
  ]
}
# Insert the user document
dbh.users.insert(user_doc, safe=True)
# Use $pull to atomically remove the "foouser2@example2.com" email sub-document
dbh.users.update({"username":"foouser"},
  {"$pull":{"emails":{"email":"foouser2@example2.com"}}}, safe=True)
```

#####$ne

```
# Use $pull to atomically remove all email sub-documents with primary not equal to True
dbh.users.update({"username":"foouser"},
  {"$pull":{"emails":{"primary":{"$ne":True}}}, safe=True)
```

#####$push

```
# Use $push to atomically append a new email sub-document to the user document
new_email = {"email":"fooemail4@exmaple4.com", "primary":False}
dbh.users.update({"username":"foouser"},
  {"$push":{"emails":new_email}}, safe=True)
```

#####Positional operator

```
# Demonstrate usage of the positional operator ($) to modify
# matched sub-documents in-place.
user_doc = {
  "username":"foouser",
  "emails":[
    {
      "email":"foouser1@example.com",
      "primary":True
    },{
      "email":"foouser2@example2.com",
      "primary":False
    },{
      "email":"foouser3@example3.com",
      "primary":False
    }
  ]
}
# Insert the user document
dbh.users.insert(user_doc, safe=True)

# Now make the "foouser2@example2.com" email address primrary
```

```
dbh.users.update({"emails.email":"foouser2@example2.com"},
  {"$set":{"emails.$.primary":True}}, safe=True)
# Now make the "foouser1@example.com" email address not primary
dbh.users.update({"emails.email":"foouser1@example.com"},
  {"$set":{"emails.$.primary":False}}, safe=True)
```

###MONGO EXPORT

```
mongoexport -d mydb -c users --out mydb.json
```

###BINARY BACKUP

```
mongodump
###
mongorestore -d mydb ./dump/mydb
###Convert to JSON
bsondump dump/mydb/users.bson > users.json
```

###MONGOSTAT

```
mongostat
```

##MONGO COMMANDS (JAVASCRIPT)

```
###SHOW AVAILABLE DATABASES
show dbs

###CONNECT TO A DATABASE OR CREATE
use databasename
db = databasename

###TO SHOW A COLLECTION
db.collectionName

###COUNT ELEMENTS ON A COLLECTION

db.collectionName.count()

###STORE DOCUMENTS ON A COLLECTION
db.collectionName.insert({title:"Document Title", url:"http://...", tags:["kapow","PWOF"], saved_on:new
Date()});

#OR

var doc = {};
doc.title = "NEW DOC TITLE";
doc.url = "http://...";
doc.tags =["KIAP","KABOOM"];
doc.saved_on = new Date();
doc.meta = {};
doc.meta.browser ="Chrome";
doc.meta.OS = "Mac OS";

db.links.save(doc); <----###if it already exists update(doc) else insert(doc)

###QUERYING
db.links.find();

###PRINT DOCUMENTS FORMATED
db.links.find().forEach(printjson);
```

###_id ObjectID( )

```
db.users.find()[1]._id.getTimestamp()

### SET NEW ID
function counter(name) {
  var ret = db.counters.findAndModify({query:{_id:name}, update:{$inc : {next:1}}, "new":true,
upsert:true});
  return ret.next;
}
db.products.insert({_id: counter("products"), name: "product 1"});
db.products.insert({_id: counter("products"), name: "product 2"});
```

###Relations

```
db.users.insert({ name: "Andrew"});
var a = db.users.findOne({name:"Andrew"});
db.links.insert({title:"JEWTUBE",url:"http://www.youtube.com", userId: a._id});
```

###Query Syntax & Operators #####SELECT

```
db.users.drop();

db.users.find();  <== returns cursorObject

db.users.findOne({'firstname':'John'}); <==returns a single Document

db.users.findOne({'name':'John'}).name;
```

#####LIMIT

```
db.links.find({favourites:100},{ title: 1, url: true}); <== select fields to retrieve

db.links.find({favourites:100},{ title: 0, url: false}); <== exclude fields

db.links.find({favourites:100},{title:1,url:1, _id:0});
```

#####NESTING

```
db.users.find({'name.first':'John'});

db.users.findOne({'name.first':'John'},{'name.last':1});
```

#####GREATER THAN Operator

```
db.links.find({favourites:{$gt:50}});
```

#####LESS THAN Operator

```
db.links.find({favourites:{$lt:50}});
```

#####LESS THAN OR EQUAL TO Operator

```
db.links.find({favourites:{$lte:50}});
```

#####GREATER THAN OR EQUAL TO Operator

```
db.links.find({favourites:{$gte:50}});
```

OPERATORS WORKING SIMULTANEUSLY

```
db.links.find({favourites:{$gt:50, $lt:300}});
```

#####NOT EQUAL Operator

```
db.links.find({'name':{$ne:'John'}});
```

#####OR Operator

```
db.links.find({$or: [{'name.first':'John'},{'name.last':'Wilson'}]});
```

#####NOR (NOT OR) Operator

```
db.links.find({$nor: [{'name.first':'John'},{'name.last':'Wilson'}]});
```

#####AND Operator

```
db.users.find({ $and:[{'name.first':'John'},{'name.last':'Jones'}]});
```

#####EXISTS Operator

```
db.users.find({email: {$exists:true}});
```

#####MOD Operator

```
db.links.find({favourites: {$mod:[5,0]}});
```

#####NOT Operator

```
db.links.find({favourites: {$not: { $mod: [5,0] }}});
```

#####ELEMENT MATCH OPERATOR
Searches inside Arrays

```
db.users.find({logins:{ $elemMatch: {minutes:20 }}});
```

#####WHERE Operator

```
db.users.find({$where: 'this.name.first === "John"',age:30});
```

#####MERGE DUPLICATES

```
db.link.distinct('favourites');
```

#####GROUP Documents

```
db.links.group({
  key: { userID:true },
  initial: { favCount: 0 },
  reduce: function (doc, o){
    o.favCount += doc.favourites;
  },
  finalize: function (o){
    o.name = db.users.findOne({_id: o.userId}).name;
  }
});
```

#####REGEX INSIDE QUERIES

```
db.links.find({title: /tuts\+$/});
```

#####REGEX Operator
Allows you to group with other operators

```
db.links.find({ title:{ $regex: /tuts\+$/, $ne:'Mobiletuts'} });
```

#####Sort

```
db.links.find({title:1,_id:0}).sort({title: 1}); /*ASCENDING ORDER*/

db.links.find({title:1,_id:0}).sort({title: -1}); /*DESCENDING ORDER*/

db.links.find({},{title:1,favourites: 1, _id:0}).sort({favourites:-1, title:1});
```

#####Limit

```
db.links.find({},title:1,favourites:1,_id:0).sort({favourites:1}).limit(1);
```

#####Skip

```
db.links.find().skip(0*3).limit(3); /*GETS ELEMENTS 1,2 & 3*/
db.links.find().skip(1*3).limit(3); /*GETS ELEMENTS 4,5 & 6*/
```

###Updating Documents ####Update Method #####Update By Replacement
Replace whatever record it finds that matches the first object. All other fields are removed and replaced with the second object values.

```
db.users.update({'name.first':'John'},{'job':'developer'});

/*UPSERT*/

/*Use this if you want to create a document if it doesn't find the one you passed to update*/

db.users.update({name: 'Kate Wills'},{name: 'Kate Wills', job:'LISP Developer}, true);
```

#####Update By Modification

```
/*INCREMENT*/

var n = {title: 'Nettuts+'};
db.links.find(n,{title:1, favourites:1});
db.links.update(n,{$inc:{favourites: 5});
```

```
db.links.update(n,{$inc:{favourites: -5});

/*CHANGE VALUE*/

var q = {name: "Kate Wills};
db.users.update(q,{$set: {job: 'Wev Developer'}});

/*This also works with fields that doesn't exist yet, and to get rid of fields*/

db.users.update(q,{$unset: {job: 'Wev Developer'}});
```

#####MULTIPARAMETER Update Multiple Records
This just works with updates by modification.

```
/*First boolean value is used for UPSERTS, the second one for MULTIPARAMETER*/

db.users.update({'name.first':'Jane'}),{$set: {job:'developer'}},false,true);
```

####Save Method

```
var bob = db.users.findOne({'name.first':'Bob'});
bob.job = 'Server Admin';
db.users.save(bob);
```

####Find & Modify Method
This method takes a single object as its parameter. This object has several properties.

```
db.users.findAndModify({
  query:{ name:'Kate Wills' },
  update:{ $set: {age: 20} },
  new: true /*return the updated object, false is the default, and it returns the object before
updates*/
});


db.users.findAndModify({
  query:{ favourites: 110 },
  update:{ $inc: { favourites: 10 } },
  sort: { title: 1},
  new: true,
  fields: {title:1, favourites:1, _id:0}
});
```

####Modification Operators ####Array Operators #####PUSH Operator Push new items into an array

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $push:{ tags:'blog' } });
```

#####PUSH ALL Operator Push each item in an array to an array in the selected document.

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $pushAll:{ tags:['one','two'] } });
```

#####ADD TO SET Operator If you want your arrays to have unique values.

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $addToSet:{ tags:'code' } });
```

#####EACH Operator To uniquely add multiple values at once to an array

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $addToSet:{ tags:{ $each: ['one','two'] } } });
```

#####PULL Operator Remove elements inside an array.

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $pull:{ tags:'four' } });
```

#####PULL ALL Operator
Remove multiple values inside an array.

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $pullAll:{ tags: ['two',three'] } });
```

#####POP Operator Remove the first or last elements of an array

```
var n = {title: 'Nettuts+'};

db.links.update(n,{ $pop:{ tags: 1 } }); /*Pop off the end*/

db.links.update(n,{ $pop:{ tags: -1 } }); /*Pop off the begining*/
```

#####POSITIONAL Operator

```
db.users.update(
  {'logins.minutes':20},
  {$inc: {'logins.$.minutes':10} },
  false, true
);

db.users.update(
  {'logins.minutes':30},
  {$set: {'logins.$.location':'unknown'} },
  false, true
);
```

#####RENAME Operator Renames fields

```
db.users.update(
  {random:true},
  {$rename: {$rename: {'random':'new_attribute_name'} }},
  false,true
);
```

###Removing Documents ####REMOVE Operator

```
db.users.remove({'name.first':'John'});
```

####FIND AND MODIFY Operator

```
db.users.findAndModify({
  query:{'name.first':/B/},
  remove:true
});
```

####DELETE A COLLECTION

```
db.collection.drop();
```

####DELETE A WHOLE DATABASE

```
db.dropDatabase();
```

To delete specific fields within a document you have to use an update using the $unset operator.
###Indexes INDEX WHATEVER FIELDS YOU QUERY MOST OFTEN BY. ####EXPLAIN Mehtod

Get query specifications like objects found, kind of cursor, number of scanned objects, milliseconds it took to do the query.

```
db.links.find({title:'Nettuts+'}).explain();
```

####ENSURE INDEX Method To create your own indexes:

```
db.links.ensureIndex({ title: 1});
```

1 means to index in ascending order. To see if the index was created:

```
db.system.indexes.find();
```

To set unique indexes for every document:

```
db.links.ensureIndex({ title: 1},{ unique:true });
```

If there were multiple documents with the same value

```
db.links.ensureIndex({ title: 1},{ unique:true, dropDups:true});
```

It would only keep the first one and get rid to any subsequent documents with the same field value.

When documents don't have the selected field they would still get an index. To avoid this use "sparse":

```
db.links.ensureIndex({ title: 1},{ sparse:true });
```

####Compound Indexes

```
db.links.ensureIndex({ title:1, url: 1 });
```

MongoDB can just use a single index per query ###Get rid of indexes

```
db.links.dropIndex({title_1_url_1});
```

###Show indexes

```
db.system.indexes.find();
```