https://naiveskill.com/mongodb-with-python/ .whichCollection.find() • db.coll.find({ attr: { $operator: value} }) • db.coll.find({ attr: { operator: value} [, { attr: { $operator: value}] })
• db.coll.update({attr:value},{$set:{attr:value}})
• db.coll.updateMany({ attr: {$op: val} } , { $inc: { attr: val} }
• db.coll.updateMany({ attr: {$op: val} } , { $inc: { attr: val} }
• aggreg=db.coll.aggregate([
{ '$group': { '_id': attr, 'total': { '$op': "$attr" } } }, { '$sort': { 'total': -1 } }])

SOME OPERATORS:
Name Description

eq Matches values that are equal to a specified value.

*gtMatchesvaluesthataregreaterthanaspecifiedvalue.*gte Matches values that are greater than or equal to a specified value. *inMatchesanyofthevaluesspecifiedinanarray.*lt Matches values that are less than a specified value. *lteMatchesvaluesthatarelessthanorequaltoaspecifiedvalue.*ne Matches all values that are not equal to a specified value. *ninMatchesnoneofthevaluesspecifiedinanarray. LogicalNameDescription*and Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. *notInvertstheeffectofaqueryexpressionandreturnsdocumentsthatdonotmatchthequeryexpression.*nor Joins query clauses with a logical NOR returns all documents that fail to match both clauses. $or Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Install if needed

pip install pymongo

In [1]:
```
import pymongo
```

launch the daemon

inside the bin dir of mongo mkdir aulaMONGODB Macs-MBP-4:bin pedro$ mongod --dbpath aulaMONGODB

Connect the client

In [2]:
```
client = pymongo.MongoClient("mongodb://localhost:27017/")
client.list_database_names()
```

Out[2]:
```
['admin', 'config', 'local']
```

Create and use a simple DB

create a DB with patients health data

In [3]:
```
db = client["med_data"]
```

Add a patients data collection (like a table)

In [4]:
```
patient_data = db["patient_data"]
```

Insert data

name, age, biological sex and heart rate. blood pressure (systolic and diastolic pressure), millimetres of mercury (mmHg), for example 156/82.

In [5]:
```
patient_record = {
   "Name": "Maureen Skinner",
   "Age": 87,
   "Sex": "F",
   "Blood pressure": [{"sys": 156}, {"dia": 82}],
   "Heart rate": 82
}
```

This is called a document (equivalent to a row in RDBMS). You can add multiple documents using commas

Now insert and query it

hints: use insert_one TO INSERT

TO QUERY USE for item in ?.find(): print(item)

In [6]:
```
#ADD INSERT CODE HERE
patient_data.insert_one(patient_record)
```

Out[6]:
```
<pymongo.results.InsertOneResult at 0x1eb96e28e80>
```

In [7]:
```
#ADD QUERY CODE HERE
for item in patient_data.find():
    print(item)
```

```
{'_id': ObjectId('6274481ed0e69181b355e147'), 'Name': 'Maureen Skinner', 'Age': 87, 'Sex': 'F', 'Blood pr
essure': [{'sys': 156}, {'dia': 82}], 'Heart rate': 82}
```

Pretty print it...

In [8]:
```
from pprint import pprint

for item in patient_data.find():
    pprint(item)
```

```
{'Age': 87,
 'Blood pressure': [{'sys': 156}, {'dia': 82}],
 'Heart rate': 82,
 'Name': 'Maureen Skinner',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e147')}
```

ObjectId to uniquely identify each document. This is a 12-byte hexadecimal string consisting of a timestamp, randomly generated value and incrementing counter.

Add multiple documents to the collection

hint: use insert_many

In [9]:
```
patient_records = [
 {
   "Name": "Adam Blythe",
   "Age": 55,
   "Sex": "M",
   "Blood pressure": [{"sys": 132}, {"dia": 73}],
   "Heart rate": 73
 },
 {
   "Name": "Darren Sanders",
   "Age": 34,
   "Sex": "M",
   "Blood pressure": [{"sys": 120}, {"dia": 70}],
   "Heart rate": 67
 },
 {
   "Name": "Sally-Ann Joyce",
   "Age": 19,
   "Sex": "F",
   "Blood pressure": [{"sys": 121}, {"dia": 72}],
   "Heart rate": 67
 }
]
```

In [ ]:
```
patient_data.insert_many(patient_records)
```

```
---------------------------------------------------------------------------
BulkWriteError                            Traceback (most recent call last)
Input In [30], in <cell line: 1>()
----> 1 patient_data.insert_many(patient_records)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-package
s\Python310\site-packages\pymongo\collection.py:691, in Collection.insert_many(self, documents, ordered,
bypass_document_validation, session, comment)
    689 blk = _Bulk(self, ordered, bypass_document_validation, comment=comment)
    690 blk.ops = [doc for doc in gen()]
--> 691 blk.execute(write_concern, session=session)
    692 return InsertManyResult(inserted_ids, write_concern.acknowledged)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-package
s\Python310\site-packages\pymongo\bulk.py:512, in _Bulk.execute(self, write_concern, session)
    510         self.execute_no_results(sock_info, generator, write_concern)
    511 else:
--> 512     return self.execute_command(generator, write_concern, session)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-package
s\Python310\site-packages\pymongo\bulk.py:393, in _Bulk.execute_command(self, generator, write_concern, s
ession)
    390     client._retry_with_session(self.is_retryable, retryable_bulk, s, self)
    392 if full_result["writeErrors"] or full_result["writeConcernErrors"]:
--> 393     _raise_bulk_write_error(full_result)
    394 return full_result

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-package
s\Python310\site-packages\pymongo\bulk.py:136, in _raise_bulk_write_error(full_result)
    134 if full_result["writeErrors"]:
    135     full_result["writeErrors"].sort(key=lambda error: error["index"])
--> 136 raise BulkWriteError(full_result)

BulkWriteError: batch op errors occurred, full error: {'writeErrors': [{'index': 0, 'code': 11000, 'keyPa
ttern': {'_id': 1}, 'keyValue': {'_id': ObjectId('6274481fd0e69181b355e14e')}, 'errmsg': "E11000 duplicat
e key error collection: med_data.patient_data index: _id_ dup key: { _id: ObjectId('6274481fd0e69181b355e
14e') }", 'op': {'Hospital number': '9956734', 'Name': 'Adam Blythe', 'Age': 55, 'Sex': 'M', 'Prescribed
medications': [DBRef('medication_data', '?'), DBRef('medication_data', '?')], '_id': ObjectId('6274481fd0
e69181b355e14e')}}], 'writeConcernErrors': [], 'nInserted': 0, 'nUpserted': 0, 'nMatched': 0, 'nModified
': 0, 'nRemoved': 0, 'upserted': []}
```

In [10]:
```python
for item in patient_data.find():
    pprint(item)
```

```
{'Age': 87,
 'Blood pressure': [{'sys': 156}, {'dia': 82}],
 'Heart rate': 82,
 'Name': 'Maureen Skinner',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e147')}
{'Age': 55,
 'Blood pressure': [{'sys': 132}, {'dia': 73}],
 'Heart rate': 73,
 'Name': 'Adam Blythe',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e148')}
{'Age': 34,
 'Blood pressure': [{'sys': 120}, {'dia': 70}],
 'Heart rate': 67,
 'Name': 'Darren Sanders',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e149')}
{'Age': 19,
 'Blood pressure': [{'sys': 121}, {'dia': 72}],
 'Heart rate': 67,
 'Name': 'Sally-Ann Joyce',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e14a')}
```

UPDATE: Darren Sanders heart rate was supposed to be 88

hints: use update_one to choose Darren: {"Name": "Darren Sanders"} to change the heart rate: {"$set":{"Heart rate": 88}}

In [11]:
```python
#ADD QUERY CODE HERE
patient_data.update_one({'Name': 'Darren Sanders'}, {"$set":{"Heart rate": 88}})
```

Out[11]:
```
<pymongo.results.UpdateResult at 0x1eb96e2a980>
```

Linking (similar to foreign keys in RDBMS)

we want to store some other medical test results for a patient.

This could include some blood test results and an ECG/EKG image for some investigations for a heart attack and some blood tests, including:

Creatine Kinase (CK) Troponin I (TROP) Aspartate aminotransferase (AST)

patient_record = { "Hospital number": "3432543", "Name": "Karen Baker", "Age": 45, "Sex": "F", "Blood pressure": [{"sys": 126}, {"dia": 72}], "Heart rate": 78, "Test results" : [] }The test results top add are: "ECG": "\scans\ECGs\ecg00023.png" "BIOCHEM": [{"AST": 37}, {"CK": 180}, {"TROPT": 0.03}]

Add the patient document with those values, THEN query to see those values

hints: use insert_one use find

In [12]:
```
patient_record = {
  "Hospital number": "3432543",
  "Name": "Karen Baker",
  "Age": 45,
  "Sex": "F",
  "Blood pressure": [{"sys": 126}, {"dia": 72}],
  "Heart rate": 78,
  "Test results" : [
   {
     "ECG": "\scans\ECGs\ecg00023.png"
   },
   {
     "BIOCHEM": [{"AST": 37}, {"CK": 180}, {"TROPT": 0.03}]
   }
  ]
}
```

In [13]:
```
#ADD insert QUERY CODE HERE
patient_data.insert_one(patient_record)
```

Out[13]:
```
<pymongo.results.InsertOneResult at 0x1eb96e28d90>
```

In [14]:
```
#ADD find QUERY CODE HERE
for item in patient_data.find():
    pprint(item)
```

```
{'Age': 87,
 'Blood pressure': [{'sys': 156}, {'dia': 82}],
 'Heart rate': 82,
 'Name': 'Maureen Skinner',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e147')}
{'Age': 55,
 'Blood pressure': [{'sys': 132}, {'dia': 73}],
 'Heart rate': 73,
 'Name': 'Adam Blythe',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e148')}
{'Age': 34,
 'Blood pressure': [{'sys': 120}, {'dia': 70}],
 'Heart rate': 88,
 'Name': 'Darren Sanders',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e149')}
{'Age': 19,
 'Blood pressure': [{'sys': 121}, {'dia': 72}],
 'Heart rate': 67,
 'Name': 'Sally-Ann Joyce',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e14a')}
{'Age': 45,
 'Blood pressure': [{'sys': 126}, {'dia': 72}],
 'Heart rate': 78,
 'Hospital number': '3432543',
 'Name': 'Karen Baker',
 'Sex': 'F',
 'Test results': [{'ECG': '\\scans\\ECGs\\ecg00023.png'},
                  {'BIOCHEM': [{'AST': 37}, {'CK': 180}, {'TROPT': 0.03}]}],
 '_id': ObjectId('6274481fd0e69181b355e14b')}
```

Now we want to link to another collection representing medication data - first insert the data

hint: use insert_many

In [15]:
```
medication_data = db["medication_data"]
```

In [16]:
```
medication_record = [
 {
   "Drug name": "Omeprazole",
   "Type": "Proton pump inhibitor",
   "Oral dose": "20mg once daily",
   "IV dose": "40mg",
   "Net price (GBP)": 4.29
 },
 {
   "Drug name": "Amitriptyline",
   "Type": "Tricyclic antidepressant",
   "Oral dose": "30–75mg daily",
   "IV dose": "N/A",
   "Net price (GBP)": 1.32
 }
]
```

In [17]:
```
#ADD QUERY CODE HERE
medication_data.insert_many(medication_record)
```

Out[17]:
```
<pymongo.results.InsertManyResult at 0x1eb96e2a230>
```

In [18]:
```
for item in medication_data.find():
    pprint(item)
```

```
{'Drug name': 'Omeprazole',
 'IV dose': '40mg',
 'Net price (GBP)': 4.29,
 'Oral dose': '20mg once daily',
 'Type': 'Proton pump inhibitor',
 '_id': ObjectId('6274481fd0e69181b355e14c')}
{'Drug name': 'Amitriptyline',
 'IV dose': 'N/A',
 'Net price (GBP)': 1.32,
 'Oral dose': '30–75mg daily',
 'Type': 'Tricyclic antidepressant',
 '_id': ObjectId('6274481fd0e69181b355e14d')}
```

Now link that medication to patients

hint:complete the parts with a question mark...

In [19]:
```
from bson.dbref import DBRef
patient_records = [
 {
   "Hospital number": "9956734",
   "Name": "Adam Blythe",
   "Age": 55,
   "Sex": "M",
   "Prescribed medications": [
     DBRef("medication_data", "?"),
     DBRef("medication_data", "?")
   ]
 },
 {
   "Hospital number": "4543673",
   "Name": "Darren Sanders",
   "Age": 34,
   "Sex": "M",
   "Prescribed medications": [
     DBRef("diagnosis_data", "?")
   ]
 }
]
```

In [20]:
```
patient_data.insert_many(patient_records)
```

Out[20]:
```
<pymongo.results.InsertManyResult at 0x1eb96e2ad40>
```

In [21]:
```
for item in patient_data.find():
    pprint(item)
```

```
{'Age': 87,
 'Blood pressure': [{'sys': 156}, {'dia': 82}],
 'Heart rate': 82,
 'Name': 'Maureen Skinner',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e147')}
{'Age': 55,
 'Blood pressure': [{'sys': 132}, {'dia': 73}],
 'Heart rate': 73,
 'Name': 'Adam Blythe',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e148')}
{'Age': 34,
 'Blood pressure': [{'sys': 120}, {'dia': 70}],
 'Heart rate': 88,
 'Name': 'Darren Sanders',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e149')}
{'Age': 19,
 'Blood pressure': [{'sys': 121}, {'dia': 72}],
 'Heart rate': 67,
 'Name': 'Sally-Ann Joyce',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e14a')}
{'Age': 45,
 'Blood pressure': [{'sys': 126}, {'dia': 72}],
 'Heart rate': 78,
 'Hospital number': '3432543',
 'Name': 'Karen Baker',
 'Sex': 'F',
 'Test results': [{'ECG': '\\scans\\ECGs\\ecg00023.png'},
                  {'BIOCHEM': [{'AST': 37}, {'CK': 180}, {'TROPT': 0.03}]}],
 '_id': ObjectId('6274481fd0e69181b355e14b')}
{'Age': 55,
 'Hospital number': '9956734',
 'Name': 'Adam Blythe',
 'Prescribed medications': [DBRef('medication_data', '?'),
                            DBRef('medication_data', '?')],
 'Sex': 'M',
 '_id': ObjectId('6274481fd0e69181b355e14e')}
{'Age': 34,
 'Hospital number': '4543673',
 'Name': 'Darren Sanders',
 'Prescribed medications': [DBRef('diagnosis_data', '?')],
 'Sex': 'M',
 '_id': ObjectId('6274481fd0e69181b355e14f')}
```

Querying data with conditions

collection.find({ }, { })

Find patient with the name "Darren Sanders"

hint: use find with the condition {"Name": "Darren Sanders"}

In [22]:
```
# pprint(ADD QUERY CODE HERE)[0])
pprint(patient_data.find({"Name":"Darren Sanders"})[0])
```

```
{'Age': 34,
 'Blood pressure': [{'sys': 120}, {'dia': 70}],
 'Heart rate': 88,
 'Name': 'Darren Sanders',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e149')}
```

But there are two Darrens, show both:

In [23]:
```
query = {"Name": "Darren Sanders"}
doc = patient_data.find(query)
for i in doc:
  pprint(i)
```

```
{'Age': 34,
 'Blood pressure': [{'sys': 120}, {'dia': 70}],
 'Heart rate': 88,
 'Name': 'Darren Sanders',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e149')}
{'Age': 34,
 'Hospital number': '4543673',
 'Name': 'Darren Sanders',
 'Prescribed medications': [DBRef('diagnosis_data', '?')],
 'Sex': 'M',
 '_id': ObjectId('6274481fd0e69181b355e14f')}
```

Show the names of patients with heart rates higher than 70

hint: condition: {"Heart rate": {"$gt": 70}}, {"Name"}

In [24]:
```
for heart_rate in patient_data.find({"Heart rate":{"$gt":70}},{"Name"}):
    pprint(heart_rate)
```

```
{'Name': 'Maureen Skinner', '_id': ObjectId('6274481ed0e69181b355e147')}
{'Name': 'Adam Blythe', '_id': ObjectId('6274481ed0e69181b355e148')}
{'Name': 'Darren Sanders', '_id': ObjectId('6274481ed0e69181b355e149')}
{'Name': 'Karen Baker', '_id': ObjectId('6274481fd0e69181b355e14b')}
```

find patients with heart rate <= 70 and aged more than 18

hint: complete the text to work...

In [25]:
```
result = patient_data.find({
    "$and" : [
    {
        "Heart rate": {"$lte": 70}
    },
    {
        "Age": {"$gt": 18}
    }
    ]})
for pt in result:
    pprint(pt)
```

```
{'Age': 19,
 'Blood pressure': [{'sys': 121}, {'dia': 72}],
 'Heart rate': 67,
 'Name': 'Sally-Ann Joyce',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e14a')}
```

find the patients with a systolic (sys) blood pressure less than 140 mmHG (mm of mercury)

hints: {"Blood pressure.sys": {"$?": ?}}

use . to access the array elements

In [33]:
```
for normal in patient_data.find({'Blood pressure.sys': {'$lt': 140}}):
    pprint(normal)
```

```
{'Age': 55,
 'Blood pressure': [{'sys': 132}, {'dia': 73}],
 'Heart rate': 73,
 'Name': 'Adam Blythe',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e148')}
{'Age': 34,
 'Blood pressure': [{'sys': 120}, {'dia': 70}],
 'Heart rate': 88,
 'Name': 'Darren Sanders',
 'Sex': 'M',
 '_id': ObjectId('6274481ed0e69181b355e149')}
{'Age': 19,
 'Blood pressure': [{'sys': 121}, {'dia': 72}],
 'Heart rate': 67,
 'Name': 'Sally-Ann Joyce',
 'Sex': 'F',
 '_id': ObjectId('6274481ed0e69181b355e14a')}
{'Age': 45,
 'Blood pressure': [{'sys': 126}, {'dia': 72}],
 'Heart rate': 78,
 'Hospital number': '3432543',
 'Name': 'Karen Baker',
 'Sex': 'F',
 'Test results': [{'ECG': '\\scans\\ECGs\\ecg00023.png'},
                  {'BIOCHEM': [{'AST': 37}, {'CK': 180}, {'TROPT': 0.03}]}],
 '_id': ObjectId('6274481fd0e69181b355e14b')}
```

import restaurants and neighbourhoods JSON

wget https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json

In [27]:
```
!/Users/pedro/servers/MongoDB/mongodb-osx-x86_64-3.6.2/bin/mongoimport -d restaurants -c restaurants --fi
```

```
The system cannot find the path specified.
```

In [28]:
```
#NOTA: DE MODO GERAL, A IMPORTACAO TB deve FUNCIONAR COM:
import json
with open('restaurants.json') as f:
  file_data = json.load(f)

my_collection.insert_many(file_data)
```

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
Input In [28], in <cell line: 3>()
      2 import json
      3 with open('restaurants.json') as f:
----> 4   file_data = json.load(f)
      6 my_collection.insert_many(file_data)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.1264.0_x64__qbz5n2kfra8p0\li
b\json\__init__.py:293, in load(fp, cls, object_hook, parse_float, parse_int, parse_constant, object_pair
s_hook, **kw)
    274 def load(fp, *, cls=None, object_hook=None, parse_float=None,
    275         parse_int=None, parse_constant=None, object_pairs_hook=None, **kw):
    276     """Deserialize ``fp`` (a ``.read()``-supporting file-like object containing
    277     a JSON document) to a Python object.
    278
   (...)
    291     kwarg; otherwise ``JSONDecoder`` is used.
    292     """
--> 293     return loads(fp.read(),
    294         cls=cls, object_hook=object_hook,
    295         parse_float=parse_float, parse_int=parse_int,
    296         parse_constant=parse_constant, object_pairs_hook=object_pairs_hook, **kw)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.1264.0_x64__qbz5n2kfra8p0\li
b\json\__init__.py:346, in loads(s, cls, object_hook, parse_float, parse_int, parse_constant, object_pair
s_hook, **kw)
    341     s = s.decode(detect_encoding(s), 'surrogatepass')
    343 if (cls is None and object_hook is None and
    344         parse_int is None and parse_float is None and
    345         parse_constant is None and object_pairs_hook is None and not kw):
--> 346     return _default_decoder.decode(s)
    347 if cls is None:
    348     cls = JSONDecoder

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.1264.0_x64__qbz5n2kfra8p0\li
b\json\decoder.py:340, in JSONDecoder.decode(self, s, _w)
    338 end = _w(s, end).end()
    339 if end != len(s):
--> 340     raise JSONDecodeError("Extra data", s, end)
    341 return obj

JSONDecodeError: Extra data: line 2 column 1 (char 544)
```

See databases

```
In [ ]: client.list_database_names()
```

Connect to restaurants, reference restaurants

```
In [ ]: db = client['restaurants']
```

```
In [ ]: client['restaurants'].list_collection_names()
```

```
In [ ]: restaurants=client['restaurants']['restaurants']
```

```
In [ ]: for item in restaurants.find().limit(5):
            print(item)
```

Compute the average scores of the restaurants.

We pass an array to the aggregate function. The $unwindparameterisusedtodeconstructthegradesarrayinordertooutputadocumentforeachelement. Nextweusethe$match parameter including everything (by using open and closing braces). We could filter further here by providing additional criteria. Next we use the $groupparametertogroupthedatathatwewanttoapplythecomputationto. Finallywecreatenewkeycalled“Avggrade”andapplythe$ avg (average) parameter to the grades scores of individual restaurants referencing grades followed by a dot and the score key. There are many other parameters that can be used for common computations such as sum,min, $max etc.

In [ ]:
```
result = restaurants.aggregate(
  [
     {"$unwind": "$grades"},
     {"$match": {}},
     {"$group": {"_id": "$name", "Avg grade": {"$avg": "$grades.score"}}}
  ]
)
```

In [ ]:
```
for item in result:
    print(item)
```

sort the returned in ascending or descending order. We could simply add another line with the sort parameter specifying which field to sort by. 1 (ascending) or -1 (descending). hint: add {"$sort": {"Avg grade": -1}}

In [ ]:
```
## your code here
```

In [ ]:
```
for item in result:
    print(item)
```

Now create a patient class in python that is a document

In [ ]:
```
pip install mongoengine
```

In [ ]:
```
from mongoengine import *
connect('odm_patients')
```

In [ ]:
```
class Patient(Document):
    patient_id = StringField(required=True)
    name = StringField()
    age = IntField()
    sex = StringField(max_length=1)
    heart_rate = IntField()
```

Add patients

# create instances of this class in the standard way in Python. Here we can create a couple of patients called Maxine and Hamza. Note that we add the save() function to the end of the line to write this data to the database.

In [ ]:
```
maxine_patient = Patient(patient_id = "342453", name = "Maxine Smith", age = 47, sex = "F", heart_rate =

hamza_patient = Patient(patient_id = "543243", name = "Hamza Khan", age = 22, sex = "M", heart_rate = 73)
```

View using python

In [ ]:
```
for patient in Patient.objects:
    print(patient.name, patient.patient_id, patient.age)
```

Now discover (find query) the patients in mongodb server directly

In [ ]:
```
client.list_database_names()
```

In [ ]:
```
## your code here
```

In [ ]:
```
for item in client['odm_patients'].patient.find():
    print(item)
```

Convert data from a Mongo database into tabular form as a Panda's dataframe object

In [ ]:
```
import pandas as pd

extracted_data = restaurants.find({},{"borough": "Bronx", "cuisine": "Bakery", "name": 1})
bronx_bakeries = list(extracted_data)

bakeries=pd.DataFrame.from_dict(bronx_bakeries);
```

In [ ]:
```
bakeries
```

The end !!!!

In [ ]:
```
```