UE 803 - Data Science

Session 4: Storing information (part 1)

Yannick Parmentier - Université de Lorraine / LORIA

Introduction

- So far:
  - (raw) data extracted from websites (either "manually" or via a Webservice/API)
  - (raw) data stored in text files using common formats (mainly CSV, JSON, XML)

Introduction

- So far:
  - (raw) data extracted from websites (either "manually" or via a Webservice/API)
  - (raw) data stored in text files using common formats (mainly CSV, JSON, XML)
- Main limitation:

Introduction

- So far:
  - (raw) data extracted from websites (either "manually" or via a Webservice/API)
  - (raw) data stored in text files using common formats (mainly CSV, JSON, XML)
- Main limitation: **how to query stored data ?**

Introduction

- So far:
  - (raw) data extracted from websites (either "manually" or via a Webservice/API)
  - (raw) data stored in text files using common formats (mainly CSV, JSON, XML)
- Main limitation: **how to query stored data ?**
  - *ad-hoc* query language + program ?

Introduction

- So far:
  - (raw) data extracted from websites (either "manually" or via a Webservice/API)
  - (raw) data stored in text files using common formats (mainly CSV, JSON, XML)
- Main limitation: **how to query stored data ?**
  - *ad-hoc* query language + program ?
  - *expressive and efficient* query language + program !

→ **databases**

Introduction (continued)

Database

Introduction (continued)

Database

> **Organized collection of data**, generally stored and accessed electronically from a computer system.
>
> The **database management system** (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data.

Source: Wikipedia

Introduction (continued)

Database

> **Organized collection of data**, generally stored and accessed electronically from a computer system.
>
> The **database management system** (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data.

Source: Wikipedia

Note that, depending on the vendor, the DBMS can either be a **standalone application** *or* rely on a **client-server architecture** (it is then made of two applications)

Introduction (continued)

- Querying data:

Introduction (continued)

- Querying data: **CRUD** (read/write) operations
  - Create
  - Read
  - Update
  - Delete

Introduction (continued)

- Querying data: **CRUD** (read/write) operations
  - Create
  - Read
  - Update
  - Delete
- Sequence of queries are often called **transactions**

Introduction (continued)

- Querying data: **CRUD** (read/write) operations
  - Create
  - Read
  - Update
  - Delete
- Sequence of queries are often called **transactions**
- (Ideally) expected properties are:

Introduction (continued)

- Querying data: **CRUD** (read/write) operations
  - Create
  - Read
  - Update
  - Delete
- Sequence of queries are often called **transactions**
- (Ideally) expected properties are: **ACID**
  - Atomicity
  - Coherence
  - Isolation
  - Durability

Introduction (continued)

- Securing fast reads: **indexing**
- Two main families of DBMS:
  - **relational** ones (based on *tables* and a high-level *Structured Query Language* [**SQL**]) → see next class
  - so-called **noSQL** ones (based on *flexible data structures*) → see today's class

Introduction (continued)

- Securing fast reads: **indexing**
- Two main families of DBMS:
  - **relational** ones (based on *tables* and a high-level *Structured Query Language* [**SQL**]) → see next class
  - so-called **noSQL** ones (based on *flexible data structures*) → see today's class

Note: we do not pay specific attention to data **partitioning / distribution** here!

Using noSQL - setting up a DBMS

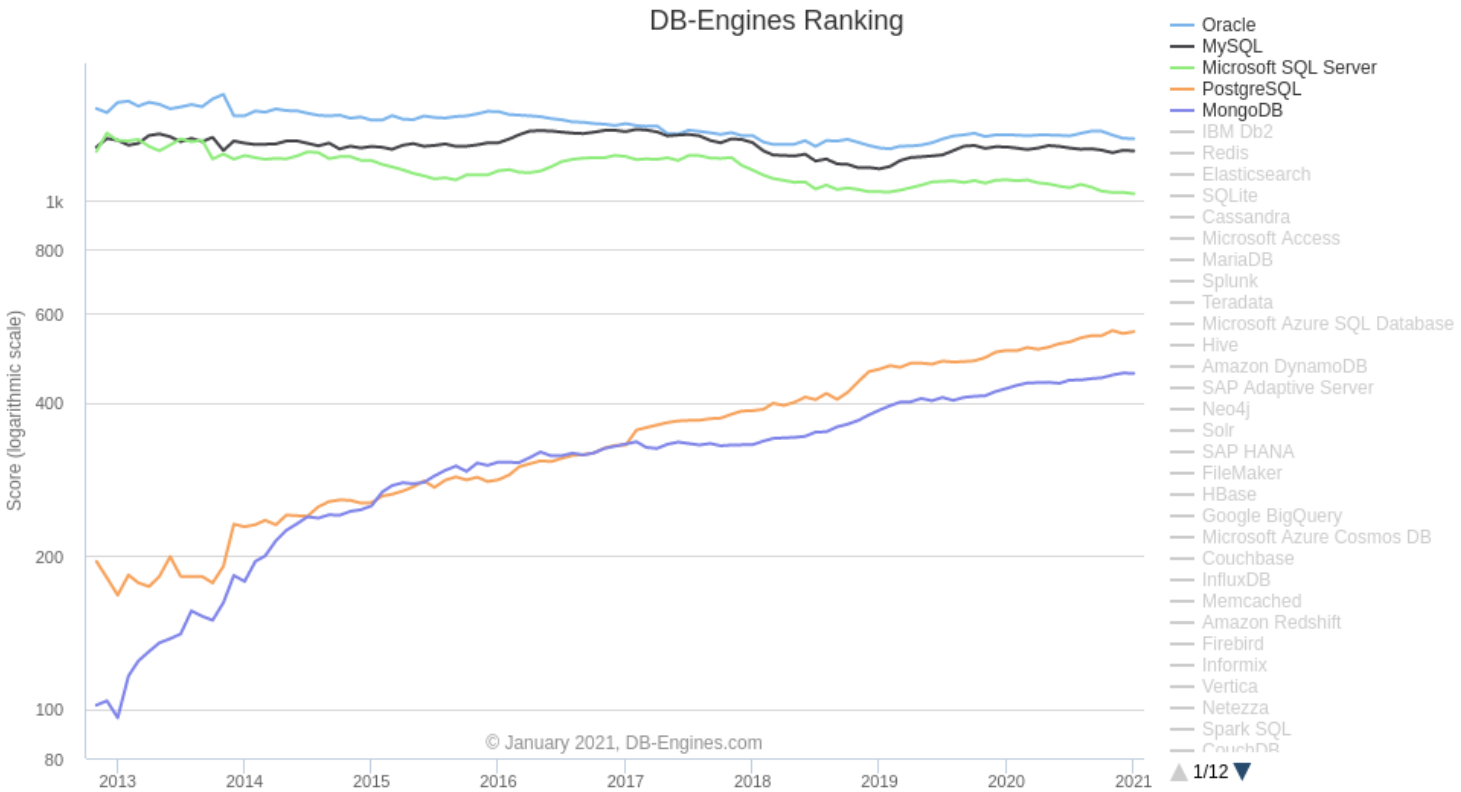
Introducing MongoDB

MongoDB

- Open-source DBMS relying on a Client/Server architecture (Server Side Public License)
- Large community of users, many **API wrappers** (aka *DB connectors*) for common programming languages
- **Efficient** and **flexible** (distributed) DBMS:
  - Data (conceptually) stored in **documents**
  - Data (technically) stored in **binary JSON files**

```
{
_id: ObjectId("509a8fb2f3f4948bd2f983a0"),
user_id: "abc123",
age: 55,
status: 'A'
}
```

MongoDB



Source: db-engines.com

MongoDB: local installation

1. **Get the server's sources** (community edition) at www.mongodb.com/download-center
   Default installation directory is:

   - Windows: `C:\Program Files\MongoDB\Server\`
   - Linux: `/usr/bin/`
   - Mac: `/Applications`

2. (optional) **Create** a default directory to host databases:

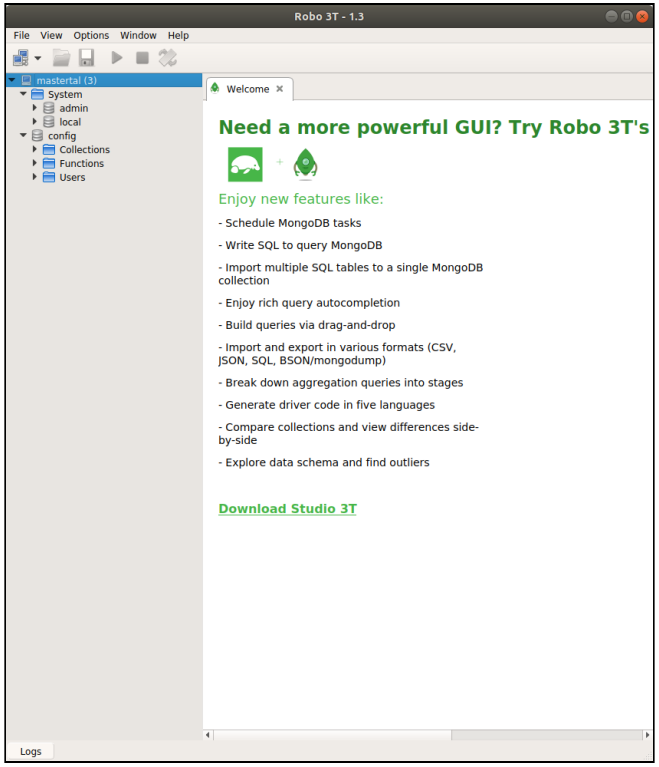
   - Windows: `C:\data\db`
   - Linux: `/data/db`
   - Mac: `/data/db`

3. **Run the MongoDB server**, e.g. `sudo /usr/bin/mongod &`

MongoDB: local installation (continued)

- (optional) **Get the graphical MongoDB client** (Robo3T) at https://robomongo.org/download
  - **Create** a new connection to localhost (port 27017)
  - **NB**: Make sure you launched the server first ...

... and that either `/data/db` exists (and you can write in it), or else that you invoked `mongod --dbpath <dir>`



MongoDB: local installation (continued)

- To **create a database**:
  - either use Robo3T (right-click on your connection)
  - or use a terminal (after installing the CLI-based MongoDB client [tools] available on mongodb.com)

```
$ mongosh
...
> db
test
> show databases
admin      41 kB
config   94.2 kB
local      41 kB
> use new_york
switched to db new_york
```

MongoDB: local installation (continued)

- To **create an empty collection** (say *restaurants*):
  - either using Robo3T (right-click on your database)
  - or in a terminal:

```
> use new_york
switched to db new_york
> db.restaurants
restaurants
```

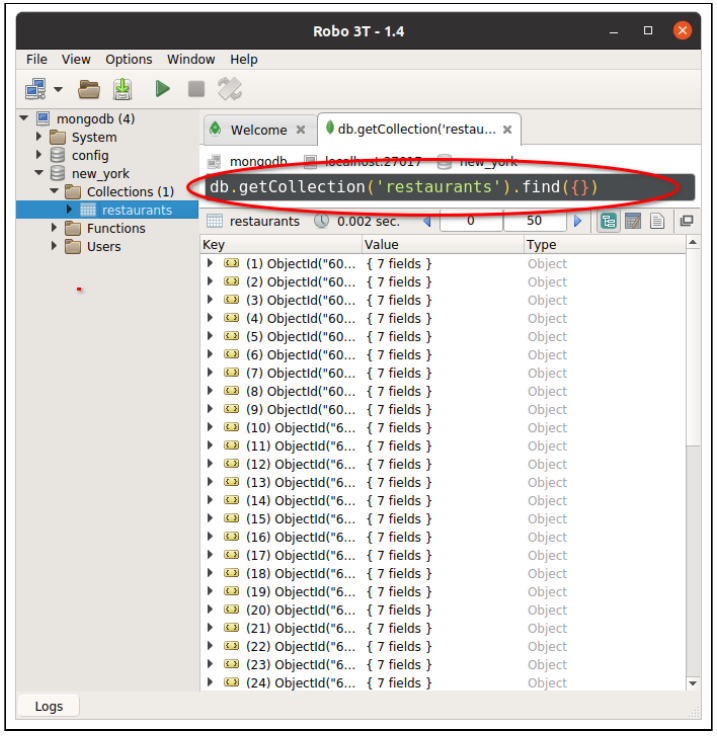
- To **import a collection** from a json file:

```
$ mongoimport  --db new_york  \
    --collection restaurants  \
    data.json

  2021-01-31T18:37:35.798+0100 connected to: mongodb://localhost
  2021-01-31T18:37:37.057+0100 25357 document(s) imported
  successfully. 0 document(s) failed to import.
```

MongoDB: local installation (continued)

- The collection can be displayed in Robo3T:
- Note you can **interact with the DBMS** in Robo3T through the connected database using the **console** at the top of the right panel!



NB: MongoDB instructions given in the next slides prefixed with `>` are run in the CLI, but using Robo3T would work as well.

Using noSQL - querying data

Introducing MongoDB instructions

Inspecting documents' structure

```
> db.restaurants.findOne()
```

```
{
  _id: ObjectId("6016eadf0fd4ccc335b932d5"),
  address: {
    building: '2780',
    coord: { type: 'Point', coordinates: [ -73.98241999999999,
                                            40.579505 ] },
    street: 'Stillwell Avenue',
    zipcode: '11224'
  },
  borough: 'Brooklyn',
  cuisine: 'American ',
  grades: [
    { date: 2014-06-10T00:00:00.000Z, grade: 'A', score: 5 },
    { date: 2013-06-05T00:00:00.000Z, grade: 'A', score: 7 },
    { date: 2012-04-13T00:00:00.000Z, grade: 'A', score: 12 },
    { date: 2011-10-12T00:00:00.000Z, grade: 'A', score: 12 }
  ],
  name: 'Riviera Caterer',
  restaurant_id: '40356018'
}
```

Adding documents

```
> db.restaurants.insertOne(
   {"name":"test",
    "borough":"tata"}
  );
```

```
{
  acknowledged: true,
  insertedId: ObjectId("60170b62b17dd489f2696053")
}
```

Adding documents

```
> db.restaurants.insertOne(
   {"name":"test",
    "borough":"tata"}
  );
```

```
{
  acknowledged: true,
  insertedId: ObjectId("60170b62b17dd489f2696053")
}
```

```
> db.restaurants.insertMany([{"borough":"toto"}, {"name":"tutu"}])
```

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6017134bb17dd489f2696054"),
    '1': ObjectId("6017134bb17dd489f2696055")
  }
}
```

Removing documents

```
> db.restaurants.deleteOne(
   {
     "_id":ObjectId("6017134bb17dd489f2696055")
   }
)
```

```
{ acknowledged: true, deletedCount: 1 }
```

Removing documents

```
> db.restaurants.deleteOne(
   {
     "_id":ObjectId("6017134bb17dd489f2696055")
   }
)
```

```
{ acknowledged: true, deletedCount: 1 }
```

```
> db.restaurants.deleteMany(
   { "name":/pizza/i }
)
```

```
{ acknowledged: true, deletedCount: 1261 }
```

Updating documents

```
> db.restaurants.updateOne(
   {"_id" :
      ObjectId("60170b62b17dd489f2696053")},
   {$set :
      {"comment" : "My new comment"}}
);
```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Filtering documents

```
> db.restaurants.find({
     "borough":"Brooklyn",
     "cuisine":"Italian",
     "address.street":"5 Avenue",
  }
)
```

```
[
  {
    _id: ObjectId("6016eadf0fd4ccc335b93c58"),
    address: {
      building: '248',
      coord: { type: 'Point', coordinates: [ -73.9814311,
                                              40.6753101 ] },
      street: '5 Avenue',
      zipcode: '11215'
    },
    borough: 'Brooklyn',
    cuisine: 'Italian',
  ...
```

Filtering documents (continued)

Projection

```
> db.restaurants.find(
    {"borough":"Brooklyn",
     "cuisine":"Italian",
     "address.street" : "5 Avenue"
    },
    {"name":1}
  )
```

```
[
  { _id: ObjectId("6016eadf0fd4ccc335b93c58"), name: 'Al Di La' },
  {
    _id: ObjectId("6016eae00fd4ccc335b93ee8"),
    name: 'Convivium Osteria'
  },
  { _id: ObjectId("6016eae00fd4ccc335b9432c"), name: 'La Villa' },
  { _id: ObjectId("6016eae00fd4ccc335b946d5"), name: 'Peperoncino'
  { _id: ObjectId("6016eae00fd4ccc335b949b4"), name: 'Mulino' },
  { _id: ObjectId("6016eae00fd4ccc335b951da"), name: "Peppino'S" },
  {
  ...
```

Fitering documents (continued)

Operators

```
> db.restaurants.find(
    {"borough":"Manhattan",
      "grades.score":{$lt : 10}
    },
    {"name":1,"grades.score":1, "_id":0}
  )
```

```
[
  {
    grades: [{ score:2 }, { score:11 }, { score:12 }, { score:12 }]
    name: 'Dj Reynolds Pub And Restaurant'
  },
  {
    grades: [{ score:3 }, { score:4 }, { score:6 }, { score:0 }],
    name: '1 East 66Th Street Kitchen'
  },
  ...
```

Note: the constraint is *existential* (is there a score < 10 ?)

Filtering documents (continued)

Combining operators

```
> db.getCollection('restaurants').find(
    {"borough":"Manhattan",
      "grades.score":{
          $lt:10,
          $not:{$gte:10}
      }
    },
    {"name":1,"grades.score":1, "_id":0}
  )
```

```
[
  {
    grades : [{ score:3 }, { score:4 }, { score:6 }, { score:0 }]
    name : "1 East 66Th Street Kitchen",
  },
  ...
```

Fitlering documents (continued)

Applying operators on the same element!

```
> db.restaurants.find(
    { "grades" : {
                    $elemMatch : {
                       "grade" : "C",
                       "score" : {$lt :40}
                     }
    }},
    {"grades.grade" : 1,"grades.score" : 1}
  )
```

```
[
  {
    _id : ObjectId("594b9172c96c61e672dcd6bc"),
    grades : [
           {grade : "A","score" : 9},
           {grade : "A","score" : 10},
           {grade : "A","score" : 9},
           {grade : "C","score" : 32}
    ]
  }
```

Available operators

`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

See full list 🔗

Aggregating data

```
> db.restaurants.aggregate( [
    { $match : {
        "grades.0.grade":"C"
    }},
    { $project : {
        "name":1, "borough":1, "_id":0
    }}
  ] )
```

```
[
  { borough: 'Queens', name: "Mcdonald'S" },
  { borough: 'Queens', name: 'Nueva Villa China Restaurant' },
  ...
```

equivalent to (javascript notation):

```
> varMatch   = { $match : { "grades.0.grade":"C"} };
> varProject = { $project : {"name":1, "borough":1, "_id":0}};
> db.restaurants.aggregate( [ varMatch, varProject ] );
```

Aggregating data (continued)

Sorting documents *(by ascending name)*

```
> varSort = { $sort : {"name":1} };
> db.restaurants.aggregate(
                  [ varMatch, varProject, varSort ] );
```

```
[
  { borough: 'Manhattan', name: '121 Fulton Street' },
  { borough: 'Manhattan', name: '508 Restaurant And Bar' },
  { borough: 'Manhattan', name: '525 Lex Restaurant & Bar' },
  { borough: 'Manhattan', name: 'Abottega' },
  { borough: 'Queens', name: "Acey Ducey'S" },
  ...
```

Aggregating data (continued)

Grouping documents by key

```
> varGroup= { $group: {"_id": null, "total": {$sum: 1}}};

> db.restaurants.aggregate( [ varMatch, varGroup ] );
```

```
{"_id" : null, "total" : 220}
```

Aggregating data (continued)

Grouping documents by key

```
> varGroup= { $group: {"_id": null, "total": {$sum: 1}}};

> db.restaurants.aggregate( [ varMatch, varGroup ] );
```

```
{"_id" : null, "total" : 220}
```

Note that the grouping **value** does not matter:

```
> varGroup2 = { $group : {"_id" : "borough",
                          "total" : {$sum : 1} } };
> db.restaurants.aggregate( [ varMatch, varGroup2 ] );
```

```
{"_id" : "borough", "total" : 220}
```

Aggregating data (continued)

Grouping documents by key *value*

```
> varGroup3 = { $group : {"_id" : "$borough",
    "total" : {$sum : 1} } };
> db.restaurants.aggregate( [ varMatch, varGroup3 ] );
```

```
[
  {_id : "Bronx", total : 27}
  {_id : "Staten Island", total : 7}
  {_id : "Manhattan", total : 83}
  {_id : "Brooklyn", total : 56}
  {_id : "Queens", total : 47}
]
```

Aggregating data (continued)

Applying an operator on a list

```
> varUnwind = {$unwind : "$grades"}
> varGroup4 = { $group :
              {"_id" : "$borough",
               "moyenne" : {$avg : "$grades.score"}}};
> varSort2 = { $sort : { "moyenne" : -1 } }

> db.restaurants.aggregate(
              [ varUnwind, varGroup4, varSort2 ] );
```

```
[
  { _id: 'Queens', moyenne: 11.644317376783727 },
  { _id: 'Brooklyn', moyenne: 11.450004830451164 },
  { _id: 'Manhattan', moyenne: 11.413674200587268 },
  { _id: 'Staten Island', moyenne: 11.30376792264088 },
  { _id: 'Bronx', moyenne: 11.042237442922374 },
  { _id: 'Missing', moyenne: 9.64102564102564 }
]
```

Working with MongoDB and Python

Introducing pymongo

About Pymongo

- Python **driver** for the MongoDB DBMS
- Implements the **MongoDB API**
- Offers a **programmatic access to a MongoDB database**
- Can be installed using `conda` or `pip`:

```
$ conda install pymongo
```

```
$ pip3 install pymongo
```

Creating a collection

```
from pymongo import MongoClient

cars = [ {'name': 'Audi', 'price': 52642},
    {'name': 'Mercedes', 'price': 57127},
    {'name': 'Skoda', 'price': 9000},
    {'name': 'Volvo', 'price': 29000},
    {'name': 'Bentley', 'price': 350000},
    {'name': 'Citroen', 'price': 21000},
    {'name': 'Hummer', 'price': 41400},
    {'name': 'Volkswagen', 'price': 21600} ]

client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.testdb
    db.cars.insert_many(cars)
```

```
<pymongo.results.InsertManyResult object at 0x7f2a0bd8f5c0>
```

Listing available collections

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.testdb
    print(db.list_collection_names())
```

Listing available collections

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.testdb
    print(db.list_collection_names())
```

Removing a collection

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.testdb
    db.cars.drop()
```

Reading data

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find()

    for car in cars:
        print('{0} {1}'.format(car['name'],
                               car['price']))
```

Reading data

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find()

    for car in cars:
        print('{0} {1}'.format(car['name'],
                               car['price']))
```

```
Audi 52642
Mercedes 57127
Skoda 9000
Volvo 29000
...
```

Filtering data

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    expensive = db.cars.find({'price': {'$gt': 50000}})

    for ecar in expensive:
        print(ecar['name'])
```

Filtering data

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    expensive = db.cars.find({'price': {'$gt': 50000}})

    for ecar in expensive:
        print(ecar['name'])
```

```
Audi
Mercedes
Bentley
```

Counting data

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    n_cars = db.cars.count_documents({})

    print("There are {} cars".format(n_cars))
```

Counting data

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    n_cars = db.cars.count_documents({})

    print("There are {} cars".format(n_cars))
```

```
There are 8 cars
```

Projections

```python
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find({}, {'_id': 1, 'name':1})

    for car in cars:
        print(car)
```

Projections

```python
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find({}, {'_id': 1, 'name':1})

    for car in cars:
        print(car)
```

```
{'name': 'Audi', '_id': ObjectId('5b41eb21b9c5d915989d48a8')}
{'name': 'Mercedes', '_id': ObjectId('5b41eb21b9c5d915989d48a9')}
{'name': 'Skoda', '_id': ObjectId('5b41eb21b9c5d915989d48aa')}
{'name': 'Volvo', '_id': ObjectId('5b41eb21b9c5d915989d48ab')}
...
```

Sorting documents

```
from pymongo import MongoClient, DESCENDING

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find().sort('price', DESCENDING)

    for car in cars:
        print('{0} {1}'.format(car['name'],
                               car['price']))
```

Sorting documents

```
from pymongo import MongoClient, DESCENDING

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find().sort('price', DESCENDING)

    for car in cars:
        print('{0} {1}'.format(car['name'],
                               car['price']))
```

```
Bentley 350000
Mercedes 57127
Audi 52642
Hummer 41400
...
```

Aggregating documents

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    agr = [ {'$group':
                {'_id': 1,
                 'all':
                    { '$sum': '$price' } } } ]

    val = list(db.cars.aggregate(agr))

    print('Sum of prices is {}'.format(val[0]['all']))
```

Aggregating documents

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    agr = [ {'$group':
                {'_id': 1,
                 'all':
                    { '$sum': '$price' } } } ]

    val = list(db.cars.aggregate(agr))

    print('Sum of prices is {}'.format(val[0]['all']))
```

```
Sum of prices is 581769
```

Aggregating documents (continued)

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.testdb
    agr = [{ '$match':
                {'$or':
                    [ { 'name': 'Audi' },
                      { 'name': 'Volvo' }]
                }},
            { '$group':
                {'_id': 1,
                'audivolvo': { '$sum': '$price' } }}]

    val = list(db.cars.aggregate(agr))

    print('Sum is {}'.format(val[0]['audivolvo']))
```

Aggregating documents (continued)

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.testdb
    agr = [{ '$match':
                {'$or':
                    [ { 'name': 'Audi' },
                      { 'name': 'Volvo' }]
                }},
            { '$group':
                {'_id': 1,
                'audivolvo': { '$sum': '$price' } }}]

    val = list(db.cars.aggregate(agr))

    print('Sum is {}'.format(val[0]['audivolvo']))
```

```
Sum is 81642
```

Controling the extracted documents

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find().skip(2).limit(3)

    for car in cars:
        print('{0}: {1}'.format(car['name'],
                                car['price']))
```

Controling the extracted documents

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find().skip(2).limit(3)

    for car in cars:
        print('{0}: {1}'.format(car['name'],
                                car['price']))
```

```
Skoda: 9000
Volvo: 29000
Bentley: 350000
```

Running MongoDB commands

```
from pymongo import MongoClient
from pprint import pprint

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb
    print(db.list_collection_names())

    status = db.command("dbstats")

    pprint(status)
```

equivalent to:

```
> db.runCommand( {dbStats:1} )
```

or

```
> db.stats()
```

Exercise Sheet #5

Using MongoDB

References

- Openclassroom Course (in French)
- PyMongo tutorial

Thank you!

Slideshow created using remark.