Mongo-in-Memory (MIM)

Ming provides an in-memory implementation of the PyMongo API called Mongo-in-Memory (MIM for short) that you can use for unit testing. This lesson will get you started.

MIM Motivation

Effective unit testing has many characteristics, but the two we'll focus on here are:

- unit tests run quickly
- unit tests are *independent* of one another

Unfortunately, these two characteristics are at odds with one another when testing MongoDB applications. MongoDB excels at rapidly reading and writing data to established databases. In order to make sure these operations stay fast, MongoDB allocates large contiguous data files on the disk. By default, even an *empty* database takes up 200MB on disk. While this is fine when you have large, long-lived databases, it becomes problematic in the case of unit tests.

MIM to the (unit testing) Rescue

Since MongoDB manages everything at the database level, the most effective way of isolating your unit tests from one another is to make sure it's been dropped before your unit test begins, allowing MongoDB to create it as-needed, and then to drop it again at the end.

In [1]:
```
import unittest

from ming import datastore

from lesson_2_0 import model as M

ds = datastore.DataStore(
    'mongodb://localhost:27017',
    database='unittest-database')

M.sess.bind = ds

class BaseDBTest(unittest.TestCase):
    def setUp(self):
        if M.sess.db.name in ds.conn.database_names():
            ds.conn.drop_database(M.sess.db.name)
        # Create any documents in the DB required for the test
    def tearDown(self):
        if M.sess.db.name in ds.conn.database_names():
            ds.conn.drop_database(M.sess.db.name)
```

Unfortunately, if you try to do this on each test, each test could take over a second just for setup and tear-down. MIM allows you to create an drop databases nearly instantly (at the expense of durability and scalability) explicitly for the purpose of unit tests. To update our 'base database test' above to use MIM, all we need to do is update our datastore:

In [2]:
```
ds = datastore.DataStore(
```

```
    'mim:///', database='unittest-database')

M.sess.bind = ds
```

Now we can create and drop databases to our heart's content.

Caveat Programmator

There are probably a few dark corners of the MongoDB query language not correctly supported by MIM. MIM is typically suited best for a unit test that you know works with a live server and wish to speed up substantially.

Exercises

I. Create a few unit tests that put data into the database using the model we developed in lesson 2.0. Use the live MongoDB server. Run the tests, timing them.
II. Update the unit tests to use MIM. Run the tests again, timing them.