Query Operators

In the previous lesson we saw how to do simple find and find_one queries. Now we will show how to query for specific documents and use some of MongoDB's advanced query operators from PyMongo.

Queries use a document-style (or python dict) syntax. This query says "find me all the documents that have a score of 90".

In [1]:
```
import pymongo
conn = pymongo.Connection()
db = conn.training
cursor = db.scores.find({'score': 90})
cursor.next()
```

Out[1]:
```
{u'_id': ObjectId('4c90f2543d937c033f424707'),
 u'name': u'quiz',
 u'score': 90.0,
 u'student': 2.0}
```

This time we will do the same query but add the count() method to only get the count of documents that match the query. We will use count() thoughout this lesson.

In [2]:
```
db.scores.find({'score': 90}).count()
```

Out[2]:
```
64
```

Queries can also use special query operators. These operators include `$gt`, `$gte`, `$lt`, `$lte`, `$ne`, `$nin`, `$regex`, `$exists`, `$not`, `$or`, and many more. The following queries demonstate the use of some of these operators.

In [3]:
```
db.scores.find({'score': {'$gt': 90}}).count()
```

Out[3]:
```
562
```

In [4]:
```
db.scores.find_one({'score': {'$lte': 60}})
```

Out[4]:
```
{u'_id': ObjectId('4c90f2543d937c033f424701'),
 u'name': u'quiz',
 u'score': 50.0,
 u'student': 0.0}
```

In [5]:
```
db.scores.find({'name': {'$in': ['quiz', 'exam']}}).count()
```

Out[5]:
```
2000
```

In [6]:
```
db.scores.find_one({'name': {'$nin': ['quiz', 'exam']}})
```

Out[6]:
```
{u'_id': ObjectId('4c90f2543d937c033f424702'),
 u'name': u'essay',
 u'score': 98.0,
 u'student': 0.0}
```

In [7]:
```
import re
rgx = re.compile('^qu')
db.scores.find_one({'name': rgx})
```

```
Out[7]: {u'_id': ObjectId('4c90f2543d937c033f424701'),
         u'name': u'quiz',
         u'score': 50.0,
         u'student': 0.0}
```

You can sort the query results on the server side using the sort method. Here's an exmaple of using a descending sort to get the highest score in our example collection.

```
In [8]: cursor = db.scores.find().sort([('score', pymongo.DESCENDING)])
        cursor.next()
```

```
Out[8]: {u'_id': ObjectId('4c90f2543d937c033f42471c'),
         u'name': u'quiz',
         u'score': 99.0,
         u'student': 9.0}
```

If we only want the server to return the fist two documents that match the query we can add a limit().

```
In [9]: cursor = db.scores.find().sort([('score', pymongo.DESCENDING)]).limit(2)
        for doc in cursor: print doc
```

```
{u'score': 99.0, u'_id': ObjectId('4c90f2543d937c033f42471c'), u'name': u'quiz',
u'student': 9.0}
{u'score': 99.0, u'_id': ObjectId('4c90f2543d937c033f424804'), u'name': u'essay',
u'student': 86.0}
```

We can also have the server skip a number of documents. Notice that methods of the cursor object can be chained together.

```
In [10]: cursor = db.scores.find().sort([('score', pymongo.DESCENDING)]).limit(2).skip(450)
         for doc in cursor: print doc
```

```
{u'score': 92.0, u'_id': ObjectId('4c90f2543d937c033f424c0f'), u'name': u'essay',
u'student': 431.0}
{u'score': 92.0, u'_id': ObjectId('4c90f2543d937c033f424c1a'), u'name': u'quiz',
u'student': 435.0}
```

We can also tell the server to only return certain fields from the matching documents.

```
In [11]: db.scores.find_one({'score': {'$gte': 65}}, {'score': 1, '_id': 0})
```

```
Out[11]: {u'score': 98.0}
```

You can also use distinct() to find all distinct values of a field (with an optional query).

```
In [12]: db.scores.distinct("name")
```

```
Out[12]: [u'quiz', u'essay', u'exam']
```

```
In [13]: db.scores.find({'score': {'$lte': 60}}).distinct('score')
```

```
Out[13]: [50.0, 56.0, 58.0, 53.0, 54.0, 51.0, 57.0, 60.0, 59.0, 52.0, 55.0]
```

Indexes

MongoDB supports single and compound key indexes. You can create indexes using PyMongo and get explain output for your queries.

```
In [14]: db.scores.find({'score': {'$lte': 75}}).explain()
```

```
Out[14]: {u'allPlans': [{u'cursor': u'BasicCursor', u'indexBounds': {}}],
          u'cursor': u'BasicCursor',
          u'indexBounds': {},
          u'indexOnly': False,
          u'isMultiKey': False,
          u'millis': 2,
          u'n': 1524,
          u'nChunkSkips': 0,
          u'nYields': 0,
          u'nscanned': 3000,
          u'nscannedObjects': 3000,
          u'oldPlan': {u'cursor': u'BasicCursor', u'indexBounds': {}}}
```

The explain output above says that the server had to scan all 3000 documents in the collection to find the 1524 that matched the query. Lets add an index to speed things up.

```
In [15]: db.scores.ensure_index('score')
```

```
Out[15]: u'score_1'
```

With the index added the server only has to scan the 1524 documents that actually match the query.

```
In [16]: db.scores.find({'score': {'$lte': 75}}).explain()
```

```
Out[16]: {u'allPlans': [{u'cursor': u'BtreeCursor score_1',
             u'indexBounds': {u'score': [[-1.7976931348623157e+308, 75]]}}],
          u'cursor': u'BtreeCursor score_1',
          u'indexBounds': {u'score': [[-1.7976931348623157e+308, 75]]},
          u'indexOnly': False,
          u'isMultiKey': False,
          u'millis': 1,
          u'n': 1524,
          u'nChunkSkips': 0,
          u'nYields': 0,
          u'nscanned': 1524,
          u'nscannedObjects': 1524}
```

You can also create compound key indexes from PyMongo.

```
In [17]: db.scores.ensure_index([('score', pymongo.ASCENDING), ('name', pymongo.DESCENDING)])
```

```
Out[17]: u'score_1_name_-1'
```

Exercises

I. In the training.scores collection, find all scores less than 65.
II. Find the highest score. Find the lowest score.
III. Create an index on the 'name' field. Do a regular expression query on that field, verifying it uses an index.