Lab 8.1: Introduction to MongoDB
IN705 Databases Three

Introduction

Relational databases are, and are likely to remain, the preferred way to handle object persistence in applications. Relational databases are especially good at providing transactional integrity and at processing ad-hoc queries.

For some applications, however, other databases may work well, particularly when there is a need for easy scalability or to avoid fixed schema. A variety of *NoSQL* databases have emerged recently that address these needs. One such database is MongoDB[1].

MongoDB is a *document database* that stores its data in *BSON*, or Binary JSON, format. JavaScript, rather than SQL, is used to interact with the database.

MongoDB is used in a number of notable large deployments for organisations like SAP, Codecademy, and CERN.

1 Install MongoDB

Although MongoDB has its own package repository, the standard Ubuntu package suits our needs. Install it on your EC2 server with the command

```
sudo apt-get install mongodb
```

2 Using the Mongo shell

Start MongoDB with the command

```
mongo lab8
```

MongoDB stores data in *collections* rather than tables. Unlike tables, collections have no fixed schemata and can be created by simply inserting a record into one, like this:

```
db.cities.insert( {
  name: "Dunedin",
  population: 126000,
  mayor: "Dave Cull",
  famous_for: ["scarfies", "shit weather"]
})
```

[1] See `http://www.mongodb.org`

Now you can see that the `cities` collection exists by entering the `show collections` command. You can view the items in the collection by using `db.cities.find()`.

The language of the MongoDB shell is Javascript. You can see the source of any function by typing its name without brackets, like this:

```
db.cities.insert
```

You can also write your own functions in JavaScript, like this:

```
function addCity(name, population, mayor, famous_for) {
    db.cities.insert({ name: name, population: population,
                       mayor: mayor, famous_for: famous_for});
}
```

Now we can add cities like this:

```
addCity("Christchurch", 366000, "Lianne Dalziel", ["earthquakes"])
```

Since MongoDB collections don't have to adhere to schemata, you can insert a city like this:

```
db.cities.insert({name: "Melbourne", country: "Australia"})
```

3 Querying

MongoDB provides decent support for ad hoc queries. For example, to find cities with populations under 200,000, we could enter

```
db.cities.find({population: {$lt: 200000}})
```

We can find Dunedin by entering

```
db.cities.find({name: "Dunedin"})
```

4 Updating

Updating is a bit tricky. For example

```
db.cities.update({name: "Dunedin"}, {population: 125000})
```

probably doesn't do what you want. Instead, use the `$set` operator like this:

```
db.cities.update({name: "Dunedin"}, {$set: {population: 125000}})
```

5 Deleting

You can remove records from a collection by passing in arguments that will find the correct records, like this:

```
db.cities.remove({name: "Christchurch"})
```

6 Homework

Explore more of the capabilities of MongoDB by consulting the relevant chapter in the *Seven Databases* book or online documents.

Since Mongo data records are scored in a JSON-like format, it should be easy to see how our splatter data could be transferred to it. Sketch out a plan to do this.