INFO20003 Week 12 Lab

MongoDB: NoSQL in Practice

NOTE: the mongo syntax taught in this lab is NOT directly assessable. The concept of NoSql and differences with relational databases are however, and this lab can help you understand how the two compare.

(MongoDB is also used widely in industry (especially for web development), and so having a basic understanding of its working is always a good thing!)

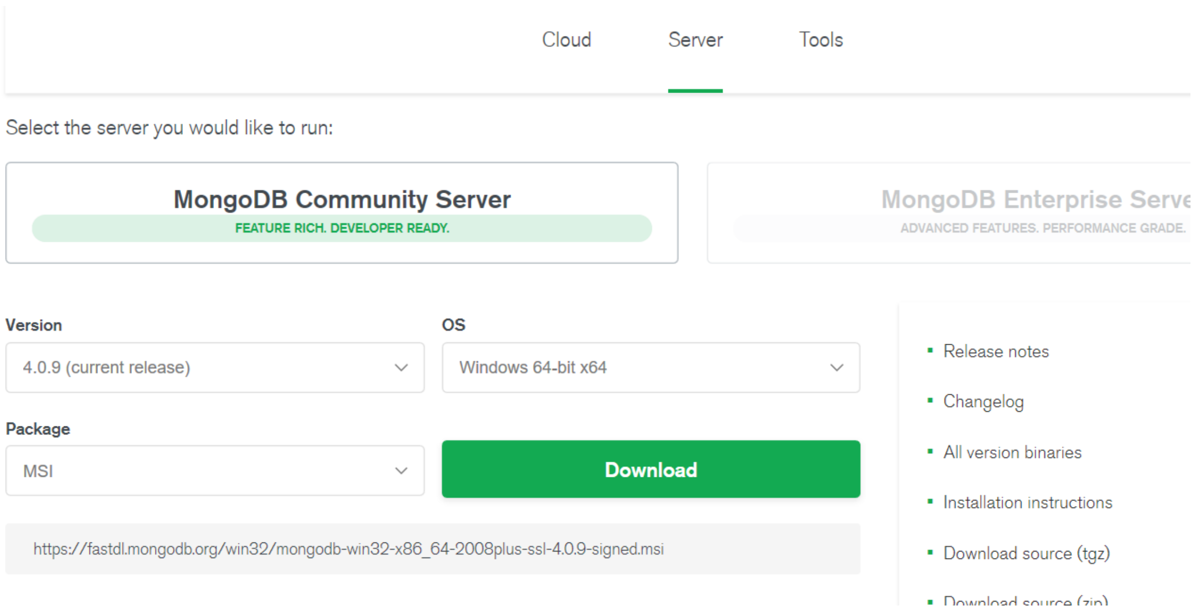
Objectives:

In this lab, you will:

1. explore the difference between an SQL and NoSQL database
2. load data into the MongoDB instance

Section 1: Install MongoDB

◆ **Task 1.1** Visit https://www.mongodb.com/download-center/community and download the current release of MongoDB Community Server for your operating system. Install using a 'complete' installation if asked during setup.



Select the server you would like to run:

Version: 4.0.9 (current release)
OS: Windows 64-bit x64
Package: MSI

https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-ssl-4.0.9-signed.msi

Section 2: Basic MongoDB

Similar but different concepts

The terminology of MongoDB is different to what you are familiar with in relational databases. Here are some important concepts translated from SQL to MongoDB:

Relational database concept	**MongoDB concept**
Database	Database
Table	Collection
Row	Document
Column	Field
Primary key	The `_id` field (the primary key always has this name)
Index	Index
Joins between tables	The `$lookup` function or embedded documents
Aggregation (GROUP BY)	Aggregation pipeline

About JSON

MongoDB documents are written in the JavaScript Object Notation (JSON) language. A JSON document contains a dictionary of fields and their values.

Here is an example of a JSON document:

```
{
  _id: ObjectId("509a8fb2f3f4948bd2f983a0"),
  userID: 119,
  name: "Lucy Adams",
  age: 27,
  emailConfirmed: true,
  hobbies: ["Carpentry", "Gardening", "Non-relational Databases"]
}
```

This document has 6 fields, including the special `_id` field which is the primary key.

Look at the `hobbies` field. Instantly you can see a difference compared to MySQL – we can store arrays in a MongoDB document!

In a JSON document, the value of a field can be one of the following types:

- Number
- String
- Boolean (`true` or `false`)
- Array (enclosed in square brackets)
- Object (another JSON document – an “embedded document” – enclosed in curly brackets)
- `null` (same as SQL `NULL` but written in lowercase)

Creating and manipulating documents

In this subsection we will set up a simple MongoDB collection (“table”) listing some people.

◆ **Task 2.1** Create a collection called `people`.

```
db.createCollection("people")
```

The equivalent SQL would be: (This is for your information only – don’t try and run this!)

```
CREATE TABLE people (...);
```

Because MongoDB is “schema-on-read”, collections do not have a set schema (eg in this example, we haven’t told mongoDB what fields each ‘person’ will have, something that you *would* have to do for a relational database inside of the CREATE TABLE statement). Each document in the collection can have any number of fields.

◆ **Task 2.2** Insert a person into the collection.

```
db.people.insertOne({
  userID: 123,
  name: "Peter Sanchez",
  age: 45
});
```

Equivalent SQL:

```
INSERT INTO people (userID, name, age)
VALUES (123, 'Peter Sanchez', 45);
```

◆ **Task 2.3** Insert a second person with a different set of fields:

```
db.people.insertOne({
  userID: 123,
  name: "April Kenneally",
  gender: "F",
  address: { street: "2 View Avenue", city: "Pascoe Vale" }
});
```

Even within the same collection, the documents can have different fields.

Notice the “embedded document” – another JSON object, or document, has been included inside the overall document to describe the person’s address in a structured way.

◆ **Task 2.4** Try an update command

```
db.bonus.updateMany(
  { gender: "F" },
  { $set: { gender: "Female" } }
)
```

Note that this update command uses fields that aren’t in all of the data, but that’s OK!

Section 3: An No-sql data set

In this section we will load a collection into MongoDB and use Mongo command line client to inspect the collection.

◆ **Task 3.1** TASK 2.1 Download the **primer-dataset.json** file from the LMS

◆ **Task 3.2** TASK 2.2 Ensure that MongoDB is started

You should ensure that you have a minimized terminal window running the **mongod** (note the **d** at the end) daemon. Google for how to install mongodb if you get stuck.

◆ **Task 3.3** TASK 2.3 Import the JSON file into the database

Open a command prompt/Terminal window and change to the MongoDB server directory, then use mongoimport to import the JSON file into the database.

Windows:

```
cd \Program Files\MongoDB\Server\3.6\bin

mongoimport.exe --db test --collection restaurants --file
C:\Users\<username>\downloads\primer-dataset.json
```

Macintosh:

`cd ~/Downloads/mongodb-osx-x86_64-3.6.3/bin` (enter `cd`, followed by the location where you extracted MongoDB, followed by `/bin`)

```
mongoimport --db test --collection restaurants --file
~/Downloads/primer-dataset.json
```

Remember that the <username> will be your account on your own laptop

Each of the switches (with the -- prefix) indicate a part of the MongoDB structure where the db is called "test" and the collection is called "restaurants"

Data Structure

The restaurant collection is a data set that is representative of the way data is displayed to end users.

	Top Level Field					
Restaurant ID	30075445					
Name	Morris Park Bakery					
Address						
	Building	1007				
	Coordinates	[-73.856077, 40.848447]				
	Street	Morris Park Ave				
	ZipCode	10462				

Borough	Bronx					
Cuisine	Bakery					
Grades						
	Date 1	1393804800000	**Grade**	A	**Score**	2
	Date 2	1378857600000	**Grade**	A	**Score**	6
	Date 3	1358985600000	**Grade**	A	**Score**	10
	Date 4	1322006400000	**Grade**	A	**Score**	9
	Date 5	1299715200000	**Grade**	B	**Score**	14
	....	....				

Figure 1: Paper representation of the Restaurants collection

The way this data is represented in the JSON file is below

```
{
"address": {
  "building": "1007",
  "coord": [ -73.856077, 40.848447 ],
  "street": "Morris Park Ave",
  "zipcode": "10462"
},
"borough": "Bronx",
"cuisine": "Bakery",
"grades": [
  { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
  { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
  { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
  { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
  { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
],
"name": "Morris Park Bake Shop",
"restaurant_id": "30075445"
}
```

Figure 2: The JSON structure of the Morris Park Bake Shop record from the Restaurants collection

◆ **Task 3.4** Record your observations about the structure of the data in the JSON file. How would you restructure the Restaurants collection to make it suitable for a relational database management system?

Section 4: More Practice with mongo

This section will teach you more about mongo syntax and basic queries. Material in this section is not assessable and is for interest's sake (remember though that mongoDB is widely used, so getting a bit of hands on experience can't hurt!)

Mongo shell

Mongo shell is a command line interface (CLI) client that enables you to connect to the MongoDB server and interrogate the collection in the database.

◆ **Task 4.1** Open a new command window and navigate to the MongoDB bin directory (MongoDB\Server\3.6\bin on Windows)

For macOS:

```
$ mongo
```

On Windows be sure to use

```
mongo.exe
```

The command prompt will change to ">" to indicate you are in the Mongo shell

```
>
```

The command `use` will set the correct database

```
> use test
```

To find all documents in a collection

```
> db.<collection>.find()
```

TASK 3.2 Find all documents in the restaurant collection

```
> db.restaurants.find()
```

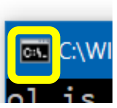
If you see the output `type "it" for more`*, this means there is more of the collection to be displayed. Type* `it` *and you will see more of the collection.*

```
> it
```

To exit the mongo shell simply type exit

```
> exit
```

Tip for Windows users
To paste commands into your mongo shell: (if shift-insert or right click isn't working)

1. Click the icon in the top-left of the Command Prompt  window
2. Go to the Edit menu
3. Click Paste

Query Methods

Try to start the mongo shell again if you have exited.

Query a top-level field

To find a restaurant with the name 'Da Vinci Pizza' we can query at the top level:

```
> db.restaurants.find( {"name": "Da Vinci Pizza"} )
```

◆ **Task 4.2** Find the information for a restaurant called Finnegan'S Wake (note the capital **S**)

```
> db.restaurants.find( {"name": "Finnegan'S Wake"} )
```

List the Borough, Zip Code, Cuisine and Street shown in the record

Query a field in an embedded document (the document inside a document)

To find other restaurants in the same zipcode as Da Vinci Pizza we would type

```
> db.restaurants.find( { "address.zipcode": "10004" } )
```

When there are many records, MongoDB will only show the first few. Type "it" (short for iteration) until all records are returned. You will know when you are at the end of the cursor because you will see the message "no cursor".

Query a field in an array

The grade section of the restaurant collection is an array of reviews containing a timestamp, a grade and a score .

◆ **Task 4.3** Find restaurants across all of New York's five boroughs that have received a "C" grade using:

```
> db.restaurants.find( { "grades.grade": "C" } )
```

Combining query conditions

The form to combine query conditions is

```
{ <field1>: <value1>, <field2>: <value2>, ... }
```

◆ **Task 4.4** find any restaurants with a C grade in zipcode 10004:

```
> db.restaurants.find( { "address.zipcode": "10004" ,
"grades.grade": "C"})
```

Operators & Conditions

Below is a table of the operators and conditions and an example of their use in MongoDB:

Operator	**Syntax**	**Example:**
Greater Than	`$gt`	`db.restaurants.find( { "grades.score": { $gt: 30 } } )`
Less Than	`$lt`	`db.restaurants.find( { "grades.score": { $lt: 10 } } )`
Greater Than Equal To	`$gte`	`db.restaurants.find( { "grades.score": { $gte: 30 } } )`
Less Than Equal To	`$lte`	`db.restaurants.find( { "grades.score": { $lte: 10 } } )`
AND		`db.restaurants.find( { "address.zipcode": "10004", "grades.grade": "C" } )`
OR	`$or`	`db.restaurants.find( { $or: [ { "cuisine": "Italian" }, { "address.zipcode": "10004" } ] } )`
Not equal	`$ne`	`db.restaurants.find( { "cuisine": { $ne: "Italian" } } )`
IN (array)	`$in`	`db.restaurants.find( { "cuisine": { $in: [ "Italian", "American" ] } } )`
NOT IN	`$nin`	`db.restaurants.find( { "cuisine": { $nin: [ "Italian", "American" ] } } )`

Table 4: Basic operators and conditions syntax in MongoDB

Insert, Update & Delete Methods

The Mongo shell provides the ability to add, update and delete records.

Insert

Consider the following record for a restaurant called Vella:

```
{
   "address" : {
      "street" : "2 Avenue",
      "zipcode" : "10075",
      "building" : "1480",
      "coord" : [ -73.9557413, 40.7720266 ]
   },
   "borough" : "Manhattan",
   "cuisine" : "Italian",
   "grades" : [
      {
         "date" : ISODate("2014-10-01T00:00:00Z"),
         "grade" : "A",
         "score" : 11
      },
      {
         "date" : ISODate("2014-01-16T00:00:00Z"),
         "grade" : "B",
         "score" : 17
      }
   ],
   "name" : "Vella",
   "restaurant_id" : "41704620"
}
```

To insert this information we would use the insert statement

```
db.restaurants.insert()
```

TASK 3.4 Add the restaurant Vella to the collection

```
> db.restaurants.insert({
      "address" : {
         "street" : "2 Avenue",
         "zipcode" : "10075",
         "building" : "1480",
         "coord" : [ -73.9557413, 40.7720266 ]
      },
      "borough" : "Manhattan",
      "cuisine" : "Italian",
      "grades" : [
         {
            "date" : ISODate("2014-10-01T00:00:00Z"),
            "grade" : "A",
            "score" : 11
         },
         {
            "date" : ISODate("2014-01-16T00:00:00Z"),
            "grade" : "B",
            "score" : 17
         }
      ],
      "name" : "Vella",
      "restaurant_id" : "41704620"
   })
```

If the row is successfully inserted, you will see the result

```
WriteResult({ "nInserted" : 1 })
```

TASK 3.5 Add the following restaurant, L'Atelier de Joel Robuchon, to the collection:

```
{
   "address" : {
      "street" : "10 Avenue",
      "zipcode" : "10011",
      "building" : "85",
   },
   "borough" : "Manhattan",
   "cuisine" : "French",
   "name" : "L'Atelier de Joel Robuchon"
}
```

Update

Updates in MongoDB depend on whether the field to be updated is the top level, as part of a document, or in an array.

To update top-level fields

```
db.restaurants.update(
    { "name" : "Juni" },
    {
      $set: { "cuisine": "American (New)" },
      $currentDate: { "lastModified": true }
    }
)
```

If successful will return the result

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

To update an embedded field

```
db.restaurants.update(
    { "restaurant_id" : "41156888" },
    { $set: { "address.street": "East 31st Street" } }
)
```

Result:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

To update more than one document:

By default the update method updates a single document. If you need to update more than one document you will need to use the `{multi: true}` option in the update method.

```
db.restaurants.update(
    { "borough" : "Missing" },
    { $set: { "borough" : "Missing data" } },
    {multi: true}
)
```

◆ **Task 4.5** Update the document to add the latitude -73.9563219 and longitude -40.7299648 (the coord field) for L'Atelier de Joel Robuchon (the restaurant you just inserted).

```
db.restaurants.update({"name":"L'Atelier de Joel Robuchon"}, {$set:
{"address.coord":"[-73.9563219, 40.7299648]"}})
```

Ensure you see 1 record matched and 1 record modified in the write result:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Delete

In MongoDB the remove keyword is used to remove records from the collection

```
db.restaurants.remove( ... )
```

To remove the record with the restaurant name Vella we would enter

```
db.restaurants.remove( {"name":"Vella"} )
```

This should remove all restaurants that have the name "Vella" in the collection.

◆ **Task 4.6** Remove the Manhattan restaurant Da Vinci Pizza

Confirm that the record exists

```
db.restaurants.find( {"name": "Da Vinci Pizza", "borough" :
"Manhattan"})
```

Now remove the record

```
db.restaurants.remove( {"name": "Da Vinci Pizza", "borough" :
"Manhattan"})
```

Aggregate Pipeline

Sometimes it is handy to count or use other aggregate functions. MongoDB can perform aggregation operations, such as grouping by a specified key and evaluating a total or a count for each distinct group.

The aggregate method performs stage-based aggregation. The aggregate() method accepts as its argument an array of stages, and each stage is processed sequentially.

```
db.collection.aggregate( [ <stage1>, <stage2>, ... ] )
```

Consider the following use of the aggregate method on the restaurants collection.

```
db.restaurants.aggregate(
   [
     { $group: { "_id": "$borough", "count": { $sum: 1 } } }
   ]
);
```

The first stage is creating a cursor of collection type borough (note the $ prefacing the borough type). The second is the type of aggregation totalling the values

The result set is as follows

```
{ "_id" : "Manhattan", "count" : 10259 }
{ "_id" : "Missing", "count" : 51 }
{ "_id" : "Queens", "count" : 5656 }
{ "_id" : "Bronx", "count" : 2338 }
{ "_id" : "Staten Island", "count" : 969 }
{ "_id" : "Brooklyn", "count" : 6086 }
```

Consider the following use of the aggregate method:

```
db.restaurants.aggregate(
   [
     { $match: { "borough": "Queens", "cuisine": "Brazilian" } },
     { $group: { "_id": "$address.zipcode" , "count": { $sum: 1 } } }
   ]
);
```

In this case we are matching the first set of conditions that the borough equals "Queens" and the cuisine equals "Brazilian". The second section is counting the number of Brazillian restaurants in each zipcode in Queens.

End of Week 12 Lab