COSC2406/2407 Database Systems **RMIT University**

Week 3 Lab Tasks

Aims:
The aim of this laboratory is for you to familiarize yourself with the MongoDB Shell and to represent data using JSON.

Task 1: Unfinished work
Make sure you have completed the work from the previous lab sheet and have both MongoDB and Apache Derby installed on your AWS Linux instance.

Task 2: Connect to MongoDB on your AWS instance

Our aim by the end of this exercise is to have inserted the following data into M

Given_name	Family_name	Age	Student_number
Fred	Zhang	20	s1234
Kim	Jones	21	s4567
Ann	Na	24	s6789

Student		ter
s1234		
s1234		
s1234		
s4567	COSC2406	
s6789	COSC2406	
s6789	COSC2110	

(1) If you are accessing MongoDB from your own machine you may want to install Robomongo https://robomongo.org/ which is cross-platform management tool. Although it is installed on mydesktop.rmit.edu.au there is an issue with the security setup.

(2) You may either create enter this data via the console application "mongo" which is provided as part of mongodb on the server or use the MongoDB shell we have just set up to insert this data. Typing queries into the shell window in Robomongo has the same effect as typing a query into the mongo client on the server so it really is up to you which you are more comfortable with.

(3) Before you do this however, you will need to think about how to combine this data into a series of documents. In order to construct these queries we have provided you with a "cheat sheet" that you find useful which can be downloaded from: https://blog.codecentric.de/files/2012/12/MongoDB-CheatSheet-v1_0.pdf

(4) You need to combine the data in the provided tables – you could choose to make your main aggregate either students or results. In either case, create json "documents" to represent this data. I have given you an example of inserting data using the Robomongo gui shell below, with the output from running this query:

(5) Using `db.student.insert` in Mo r and insert the documents to represent the data in th
by firstly running mongod to run the server and then connect to the server in another window and call 'mongo' which will start your client. The client on your server will then execute commands just like the ones you have been executing in the mongodb web shell. You might find it more efficient to use a screen manager such as tmux or screen so you can easily switch from the client to the server and back again.

(6) Write a query to find the names of the students enrolled in COSC2110 in semester 2.

(7) The next thing we might want to do is test the performance of MongoDB in terms of the number of queries it can perform per second on our dataset. Our dataset is rather small and so a more realistic test might require a much larger dataset than this.

The built-in function we use for this is 'benchRun' which we can use to easily change the parameters of tests we run in mongodb.

We simply define an 'ops' variable in mongodb as follows – notice that we are providing all the data to allow the execution of three 'findOne' queries in a

json array:

```
ops = [ {op: "findOne", ns: "test.students", query:
{student_id: "s1234"}} , {op: "findOne", ns:
"test.students", query: {student_id: "s4567"}},
{op: "findOne", ns: "test.students", query:
{student_id: "s6789"}}]
```

Please note that you might have chosen another name for some of these fields and will need to adjust your query accordingly. We then run a for – loop such as the one below:

```
for(i=1; i<=128; i*=2) { res = benchRun( { parallel:
i, seconds: 5, ops: ops}); print("threads: " + i +
"\t queries/sec: " + res.query); }
```

This will run the queries we have specified in mongodb for five seconds and tell us how many times our set of queries were run per second according to the number of threads used.