1 Project 1

Due Date: 9/17 by 11:59p

Important Reminder: As per the course *Academic Honesty Statement*, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

1.1 Aims of This Project

The aims of this project are as follows:

- To gain familiarity with the programming environment you will be using for this course.
- To provide experience with writing a non-trivial JavaScript program.
- To understand the use and limitations of using aynchronous callbacks.

1.2 Specifications

Write a command-line nodejs program which is invoked as:

```
$ db_ops.js DB_URL OP
```

`DB_URL` is the URL for a mongo database instance and `OP` is a JSON string `OP` specifies the CRUD operation which is being performed on the database specified by `DB_URL`. It must have the following top-level members:

op Must have a value which must be one of `"create"`, `"read"`, `"update"` or `"delete"`.

collection Must specify the name of the collection on which the operation specified by `op` is performed.

Additionally, `OP` may also have a top-level member `args` with specific form depending on the value of `op`. Specifically, `args` must be specified as follows:

When `op` is `create` `args` is mandatory and must be a non-empty list containing JSON objects to be inserted into the collection.

When `op` is `read`, `update` or `delete` `args` is optional and should default to `{}`. If given, it should specify a query for a mongo-db `find()` command.

When `op` is `update`, `OP` must contain an additional top-level member `fn`. It must specify a 2-element JSON list specifying a mapper function used for transforming the JSON elements selected by the `args` specifier before updating the collection. Specifically, the first element of the list must be a string containing a js identifier specifying the parameter name for the mapper function and the

second element of the list must be a string containing the body of the mapper function.

The action taken by the program depends on the `op` field of the `OP` argument to the program.

create The json objects specified by `args` are added to the specified collection in the database.

read The objects selected from the collection by the query specified by `args` are listed on standard output using `console.log()`.

update The objects selected from the collection by the query specified by `args` are updated in the collection after transformation by the mapper function specified by `fn`.

delete The objects selected from the collection by the query specified by `args` are deleted from the collection.

Note that all commands other than `read` return silently on success.

The program should attempt to do a reasonable job of error detection. When an error is detected, it may report the error either by a exception trace or by a suitable message printed on standard error (using `console.error()`).

1.3 A Sample Log

An annotated log of the program in operation is shown below. The log shows the manipulation of some data within the `test` collection in the `test_db` database of a mongo-db installation on the localhost running on the default port.

Since it is tedious to repeatedly type JSON at the command-line, some of the JSON is stored in files and spliced into the command-line using backquotes. Note also that special characters within the JSON must be protected from the shell by quoting the JSON either using single-quotes or double-quotes (the latter when splicing using backquotes).

```
#create env var for DB URL (use setenv URL mongodb... for csh variants).
$ export URL=mongodb://localhost:27017/test_db

#run mongo shell to show test collection in db is currently empty
$ mongo --eval 'db.test.find({})' test_db
MongoDB shell version: 3.2.11
connecting to: test_db

#list out file to be used for inserting entries
$ cat create.json
{
  "op": "create",
  "collection": "test",
  "args": [
```

```
    { "a": 1 },
    { "b": 2 },
    { "c": 3 }
  ]
}

#add entries to test
$ ./db_ops.js $URL "`cat create.json`"

#run mongo shell to show entries were inserted
$ mongo --eval 'db.test.find({})' test_db
MongoDB shell version: 3.2.11
connecting to: test_db
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff1"), "a" : 1 }
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff2"), "b" : 2 }
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff3"), "c" : 3 }

#use a read op to show entries inserted
$ ./db_ops.js $URL '{"op": "read", "collection": "test" }'
{ _id: 59ac8f45557cf41ebdacaff1, a: 1 }
{ _id: 59ac8f45557cf41ebdacaff2, b: 2 }
{ _id: 59ac8f45557cf41ebdacaff3, c: 3 }

#filter read to only select {a: 1} entry
 ./db_ops.js $URL \
   '{"op": "read", "collection": "test", "args": { "a": 1 } }'
{ _id: 59ac8f45557cf41ebdacaff1, a: 1 }

#no results if we filter using {a: 2}
$ ./db_ops.js $URL \
  '{"op": "read", "collection": "test", "args": { "a": 2 } }'

#delete {a: 1} entry
$ ./db_ops.js $URL \
  '{"op": "delete", "collection": "test", "args": { "a": 1 } }'

#use mongo shell to verify entry deleted
$ mongo --eval 'db.test.find({})' test_db
MongoDB shell version: 3.2.11
connecting to: test_db
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff2"), "b" : 2 }
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff3"), "c" : 3 }

#use read op to verify entry deleted
$ ./db_ops.js $URL '{"op": "read", "collection": "test" }'
{ _id: 59ac8f45557cf41ebdacaff2, b: 2 }
```

```
{ _id: 59ac8f45557cf41ebdacaff3, c: 3 }

#list out file used for updating entries
$ cat update.json
{
  "op": "update",
  "collection": "test",
  "args": { },
  "fn": [ "x",
       "{ return typeof(x['b']) !== 'undefined' ? { b: 42 } : x; }" ]
}

#update collection test as per update.json OP.
$ ./db_ops.js $URL "`cat update.json`"

#read test collection to verify update
$ ./db_ops.js $URL '{"op": "read", "collection": "test" }'
{ _id: 59ac8f45557cf41ebdacaff2, b: 42 }
{ _id: 59ac8f45557cf41ebdacaff3, c: 3 }

#use mongo shell to verify update
$ mongo --eval 'db.test.find({})' test_db
MongoDB shell version: 3.2.11
connecting to: test_db
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff2"), "b" : 42 }
{ "_id" : ObjectId("59ac8f45557cf41ebdacaff3"), "c" : 3 }

#delete all entries in test collection
$ ./db_ops.js $URL '{"op": "delete", "collection": "test" }'

#read test collection to verify delete
$ ./db_ops.js $URL '{"op": "read", "collection": "test" }'
$
```

1.4 Provided Files

The files directory contains the following files which provide the basic skeleton for your project:

db_ops.js The *main* program for your project. Does trivial processing of the command-line arguments. You should not need to change this file.

db_ops_lib.js The guts of the code for your project. You should write your code here.

README A README which minimally you must fill-in with your identification information by replacing the `XXXX` entries. You may also add any

other information which is relevant to your submission.

1.5 Hints

You may do the project on any machine which has compatible nodejs and mongodb installations. However, it is entirely your responsibility to make sure that the code works on your gcloud vm before submission.

The following steps are merely advisory:

1. Review material you will need for this project; the material for JSON, testing and mongo-db covered in class as well as online documentation. You should get into the habit of making extensive use of search engines to find necessary information.

   You can use the nodejs REPL and the *mongo shell* to play around with js and mongo-db respectively.

2. Get started by using `npm init` to create a skeleton project. Specifically, go to the directory where you want to keep your projects and create and initialize a sub-directory for this project:

```
$ mkdir db_ops
$ cd db_ops
$ mkdir test
$ npm init -y
...
{
  "name": "db_ops",
...
}


$ npm install mongodb --save
npm notice created a lockfile ...
...
#ensure that downloaded modules are never checked into git
$ echo node_modules > .gitignore
```

3. Set up your project to use mocha for testing and add `bin` and `files` top-level entries to the `package.json` generated during the previous step:

```
 "scripts": {
   "test": "mocha"
 },
 "bin" {
   "db_ops": "./db_ops.js"
 },
```

```
"files": [
  "db_ops.js", "db_ops_lib.js", "README"
],
```

4. Copy over the files from the files directory to your project directory. If you have completed the previous steps you should be able to run `./db_ops.js` and at least get back a usage message.

5. Decide how to structure your code. You need to handle each of the 4 different CRUD commands; it seems reasonable to do a dispatch on the `op` value of the `OP` specification to a separate js function dedicated to each command.

6. Implement the `create` command using the mongo docs. It is imperative that you ensure that the db connection is closed before exit. Since the insertion code will run asynchronously via a callback (or promise), it is **absolutely necessary** to ensure that the db close is done only at completion of the callback. Test interactively or by adding tests in the `test` directory (the latter is preferred).

7. Implement the `delete` command as in the previous step.

8. Implement the `read` command.

9. Implement the `update` command. This is the only command which is slightly non-trivial. You will need to create the mapper function specified by the `fn` specification in the `OP`. You can do so using the provided `newMapper()` function. If you call `newMapper()` giving it the 2 elements of the `fn` array, it will return to you a new function. If you store this function in a variable `mapper`, then you can map a data value `d` using `mapper.call(null, d)`.

   The operation will need to read the db entries specified by `args`; map each returned entry using the mapper function before writing out the mapped value to the db.

10. Test to ensure that you have met all the project specifications.

1.6 Debugging Your Code

You can of course simply use `console.log()` or `console.error()` to help debug your project. However, it may be worth your while to use a full-fledged debugger. Look around on the web, possibilities include using an IDE like eclipse or the debugger built into chrome. Note that you will not be able to use GUI applications running on your gcloud VM (unless you upgrade it; in that case you will need to turn it off when not in use to conserve your coupon balance).

1.7 Project Submission

Follow the the *gitlab instructions* in order to set up an account on gitlab.com.

- Submit your project files in a top-level `submit/db_ops` directory of your `cs480w` or `cs580w` project in gitlab.
- You should **not** submit executables or object files. You should submit all the files you used from the files directory, whether you modified them or not.
- Make sure that you edit the `README` to provide your ID information by replacing the `XXXX` entries.

If your project is incomplete on the due date, please add a file called `.LATE` in your directory so that it will not be graded. Once the project has been completed late, please remote that file and email the grader who has been assigned to your course: *CS 480W*, *CS 580W*.