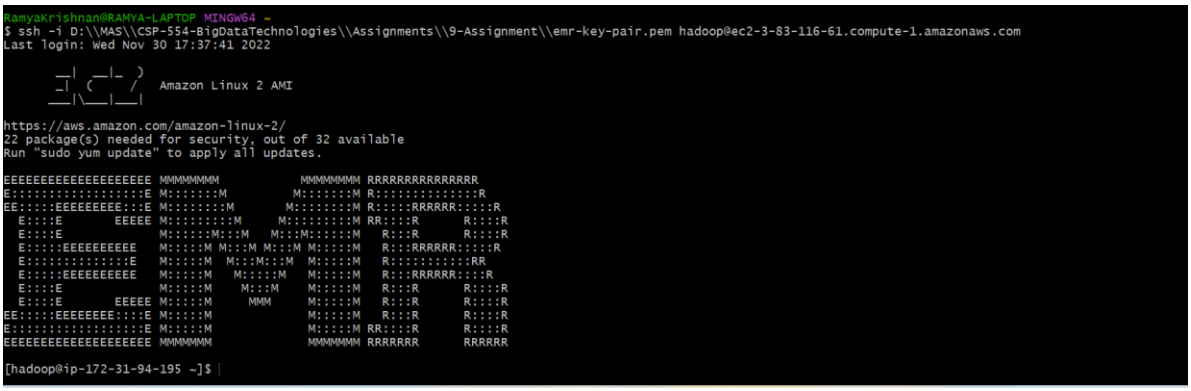
# CSP554—Big Data Technologies

## Assignment #9

## Worth: 15 points (1 point for each problem)

## HBase

Start up a Hadoop cluster as previously, but instead of choosing the "Core Hadoop" configuration chose the "HBase" configuration (see below), otherwise proceed as before:




## Exercises

Exercise 1) (1 point)

Create an HBase table with the following characteristics

Table Name: csp554Tbl

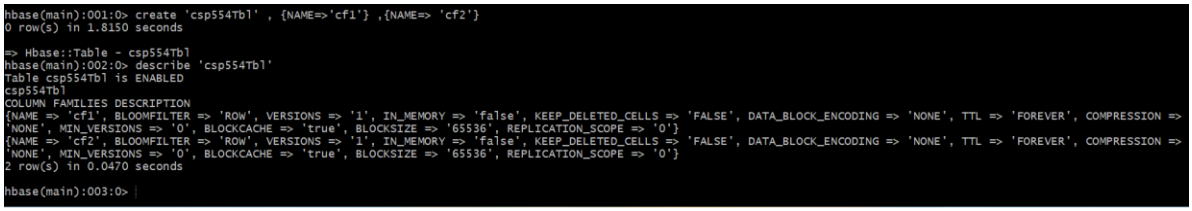
First column family: cf1

Second column family: cf2

Then execute the DESCRIBE command on the table and return command you wrote and the output as the results of this exercise.

**Command:**

```
Hbase shell
create 'csp554Tbl', {NAME=>'cf1'},{NAME=> 'cf2'}
describe 'csp554Tbl'
```



Exercise 2) (1 point)

Put the following data into the table created in exercise 1:

| Row Key | Column Family | Column (Qualifier) | Value |
|---|---|---|---|
| **Row1** | cf1 | name | Sam |
| **Row2** | cf1 | name | Ahmed |
| **Row1** | cf2 | job | Pilot |
| **Row2** | cf2 | job | Doctor |
| **Row1** | cf2 | level | LZ3 |
| **Row2** | cf2 | level | AR7 |

Execute the SCAN command on this table returning all rows, column families and columns. Provide the command and its result as the output of this exercise.

**Command:**

```
put 'csp554Tbl','Row1','cf1:name','Sam'
put 'csp554Tbl','Row2','cf1:name','Ahmed'
put 'csp554Tbl','Row1','cf2:job','Pilot'
put 'csp554Tbl','Row2','cf2:job','Doctor'
put 'csp554Tbl','Row1','cf2:level','LZ3'
put 'csp554Tbl','Row2','cf2:level','AR7'
scan 'csp554Tbl'
```

```
hbase(main):009:0> scan 'csp554Tbl'
ROW                                   COLUMN+CELL
 Row1                                 column=cf1:name, timestamp=1669830513131, value=Sam
 Row1                                 column=cf2:job, timestamp=1669830538538, value=Pilot
 Row1                                 column=cf2:level, timestamp=1669830555047, value=LZ3
 Row2                                 column=cf1:name, timestamp=1669830530721, value=Ahmed
 Row2                                 column=cf2:job, timestamp=1669830546275, value=Doctor
 Row2                                 column=cf2:level, timestamp=1669830563575, value=AR7
2 row(s) in 0.0360 seconds

hbase(main):010:0>
```

Exercise 3) (1 point)

Using the above table write a command that will get the value associated with row (Row1), column family (cf2) and column/qualifier (level). Provide the command and its result as the output of this exercise.

**Command:**

```
get 'csp554Tbl','Row1','cf2:level'
```

```
hbase(main):010:0> get 'csp554Tbl','Row1','cf2:level'
COLUMN                                  CELL
 cf2:level                              timestamp=1669830555047, value=LZ3
1 row(s) in 0.0410 seconds

hbase(main):011:0>
```

Exercise 4) (1 point)

Using the above table write command that will get the value associated with row (Row2), column family (cf1) and column/qualifier (name). Provide the command and its result as the output of this exercise.

**Command:**

```
get 'csp554Tbl','Row2','cf1:name'
```

```
hbase(main):011:0> get 'csp554Tbl','Row2','cf1:name'
COLUMN                                  CELL
 cf1:name                               timestamp=1669830530721, value=Ahmed
1 row(s) in 0.0260 seconds

hbase(main):012:0>
```

Exercise 5) (1 point)

Using the above table write a SCAN command that will return information about only two rows using the LIMIT modifier. Provide the command and its result as the output of this exercise.

**Command:**

```
scan 'csp554Tbl',{LIMIT=>1}
```

```
hbase(main):012:0> scan 'csp554Tbl',{LIMIT=>1}
ROW                                     COLUMN+CELL
 Row1                                   column=cf1:name, timestamp=1669830513131, value=Sam
 Row1                                   column=cf2:job, timestamp=1669830538538, value=Pilot
 Row1                                   column=cf2:level, timestamp=1669830555047, value=LZ3
1 row(s) in 0.0380 seconds

hbase(main):013:0>
```

# Cassandra

## Exercises:

Exercise 1) (1 point)

Read the article "A Big Data Modeling Methodology for Apache Cassandra" and provide a ½ page summary including your comments and impressions.

Leading distributed database that is highly available, scalable, and transactional is Apache Cassandra. On clusters with thousands of nodes spread across numerous data centers, it is recognized to manage some of the largest databases in the world. Product catalogs and playlists, sensor data and the Internet of Things, messaging and social networking, recommendation, personalization, fraud detection, and several more time series data-related applications are examples of Cassandra data management use cases. The widespread use of Cassandra in big data applications is due, in part, to its scalable and fault-tolerant peer-to-peer architecture, adaptable and flexible data model that developed from the BigTable data model, declarative and user-friendly Cassandra Query Language (CQL), and extremely efficient write and read access paths that enable crucial big data applications to remain always available.

In this paper, a rigorous query-driven data modeling approach for Apache Cassandra is presented. In a variety of areas, including query-driven schema design, data nesting, and data duplication, the methodology was demonstrated to be significantly different from the conventional relational data modelling approach. In order to move from conceptual data models that are independent of technology to logical data models that are specific to Cassandra, this article describes mapping rules and mapping patterns and elaborates on the core data modeling concepts for Cassandra. Additionally, it clarifies the function of physical data modeling and suggests Chebotko Diagrams, a cutting-edge visualization method that may be utilized to record intricate logical and physical data models.

Finally, it introduces KDM, a potent data modeling tool that automates some of the most difficult, error-prone, and time-consuming data modelling processes, such as conceptual-to-logical mapping, logical-to-physical mapping, and CQL creation.

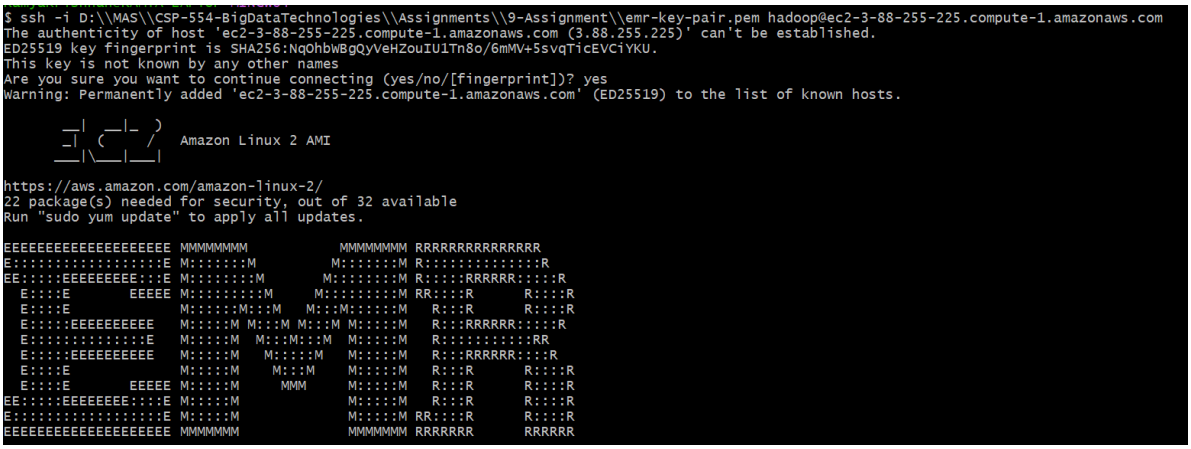
Exercise 2) (1 point)

<u>Step A – Start an EMR cluster</u>

Start up an EMR/Hadoop cluster as previously, but instead of choosing the "Core Hadoop" configuration chose the "Spark" configuration (see below), otherwise proceed as before.

<u>Step B – Install the Cassandra database software and start it</u>

Open up a terminal connection to your EMR master node. Over the course of this exercise, you will need to open up three separate terminal connections to your EMR master node. This is the first, which we will call Cass-Term.

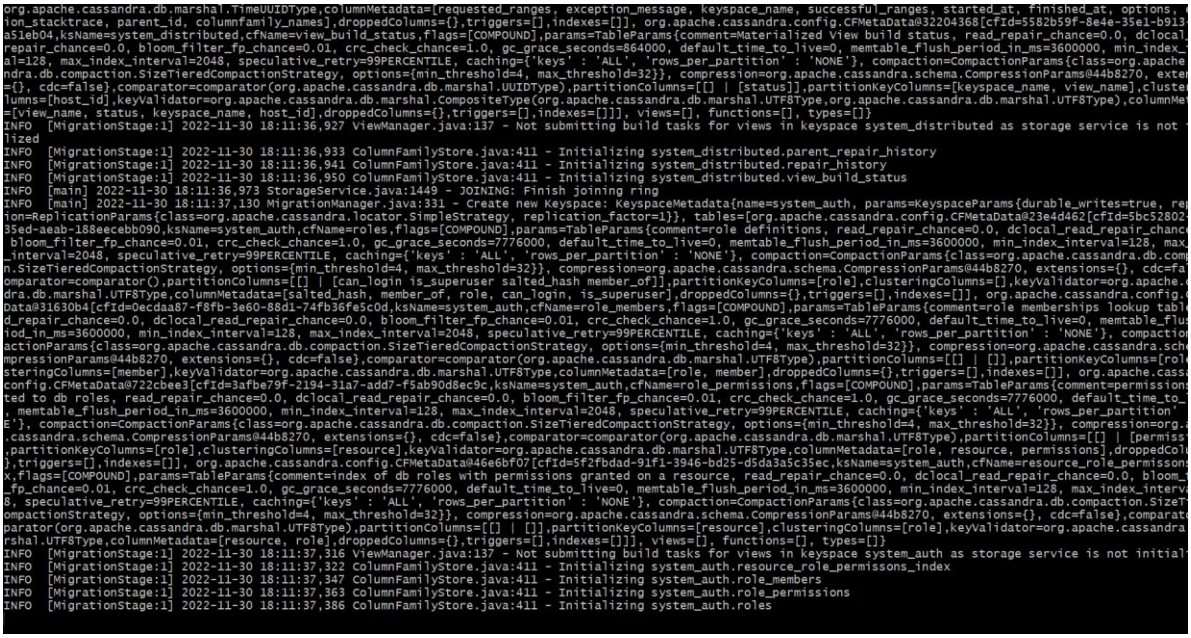



Enter the following two commands:

wget https://archive.apache.org/dist/cassandra/3.11.2/apache-cassandra-3.11.2-bin.tar.gz
tar -xzvf apache-cassandra-3.11.2-bin.tar.gz

Note, this will create a new directory (apache-cassandra-3.11.2) holding the Cassandra software release.
Then enter this command to start Cassandra (lots of diagnostic messages will appear):

apache-cassandra-3.11.2/bin/cassandra &




Step C – Run the Cassandra interactive command line interface

Open a second terminal connection to the EMR master node. Going forward we will call this terminal connection: Cqlsh-Term.
Enter the following into this terminal to start the command line interface csqlsh:

apache-cassandra-3.11.2/bin/cqlsh


Step D – Prepare to edit your Cassandra code

Open a third terminal connection to the EMR master node. Going forward we will call this terminal connection: Edit-Term.

You will use this terminal window to run the 'vi' editor to create your Cassandra code files. See the "Free Books and Chapters" section of our blackboard site for information on how to use the 'vi' editor.
As an alternative you could edit your Cassandra code files on your PC/MAC using a text editor like "notepad" or "textedit" "and then 'scp' them to the EMR mater node.

a) Create a file in your working (home) directory called init.cql using your Edit-term (or using your PC/MAC and then scp it to the EMR master node) and enter the following command. Use your IIT id as the name of your keyspace… For example, if your id is A1234567, then replace <IIT id> below with that value:

CREATE KEYSPACE <IIT id> WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

For example, you might write:

CREATE KEYSPACE A20506653 WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };


b) Then execute this file in the CQL shell using the Cqlsh-Term as follows…

source './init.cql';

c) To check if your script file has created a keyspace execute the following in the CQL shell:

describe keyspaces;

```
cqlsh> describe keyspaces;

system_schema  system_auth  a20506653  system  system_distributed  system_traces

cqlsh>
```

d) At this point you have created a keyspace unique to you. So, make that keyspace the default by entering the following into the CQL shell:

USE A20506653;

```
cqlsh> USE A20506653;
cqlsh:a20506653>
```

Now create a file in your working directory called ex2.cql using the Edit-Term (or PC/MAC and scp). In this file write the command to create a table named 'Music' with the following characteristics:

| Attribute Name | Attribute Type | Primary Key / Cluster Key |
|---|---|---|
| **artistName** | text | Primary Key |
| **albumName** | text | Cluster Key |
| **numberSold** | int | Non Key Column |
| **Cost** | int | Non Key Column |

Execute ex2.cql in the CQL shell. Then execute the shell command 'DESCRIBE TABLE Music;' and include the output as the result of this exercise.

```
[hadoop@ip-172-31-44-252 ~]$ cat ex2.cql
CREATE table a20506653.music(
artistname text,
albumname text,
numbersold int,
cost int,
PRIMARY KEY(artistname, albumname)
) WITH CLUSTERING ORDER BY (albumname DESC);
[hadoop@ip-172-31-44-252 ~]$
```

```
cqlsh:a20506653> describe table music;

CREATE TABLE a20506653.music (
    artistname text,
    albumname text,
    cost int,
    numbersold int,
    PRIMARY KEY (artistname, albumname)
) WITH CLUSTERING ORDER BY (albumname DESC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Exercise 3) (1 point)

Now create a file in your working directory called ex3.cql using the Edit-Term. In this file write the commands to insert the following records into table 'Music'…

| artistName | albumName | numberSold | cost |
|---|---|---|---|
| **Mozart** | Greatest Hits | 100000 | 10 |
| **Taylor Swift** | Fearless | 2300000 | 15 |
| **Black Sabbath** | Paranoid | 534000 | 12 |
| **Katy Perry** | Prism | 800000 | 16 |
| **Katy Perry** | Teenage Dream | 750000 | 14 |

a) Execute ex3.cql. Provide the content of this file as the result of this exercise.

```
[hadoop@ip-172-31-44-252 ~]$ cat ex3.cql
INSERT INTO MUSIC (artistname,albumname,numbersold,cost) values ('Mozart','Greatest Hits',100000,10);
INSERT INTO MUSIC (artistname,albumname,numbersold,cost) values ('Taylor Swift','Fearless',2300000,15);
INSERT INTO MUSIC (artistname,albumname,numbersold,cost) values ('Black Sabbath','Paranoid',534000,12);
INSERT INTO MUSIC (artistname,albumname,numbersold,cost) values ('Katy Perry','Prism',800000,16);
INSERT INTO MUSIC (artistname,albumname,numbersold,cost) values ('Katy Perry','Teenage Dream',750000,14);[hadoop@ip-172-31-44-252 ~]$
```

b) Execute the command 'SELECT * FROM Music;' and provide the output of this command as another result of the exercise.

```
cqlsh:a20506653> SELECT * FROM Music;

 artistname    | albumname     | cost | numbersold
---------------+---------------+------+------------
        Mozart | Greatest Hits |   10 |     100000
 Black Sabbath |      Paranoid |   12 |     534000
  Taylor Swift |      Fearless |   15 |    2300000
    Katy Perry | Teenage Dream |   14 |     750000
    Katy Perry |         Prism |   16 |     800000

(5 rows)
cqlsh:a20506653>
```

Exercise 4) (1 point)

Now create a file in your working directory called ex4.cql using the Edit-Term. In this file write the commands to query and output only Katy Perry songs. Execute ex4.cql. Provide the content of this file and output of executing this file as the result of this exercise.

```
[hadoop@ip-172-31-44-252 ~]$ cat ex4.cql
select * from music where artistname in ('Katy Perry');
```

```
cqlsh:a20506653> source './ex4.cql';

 artistname | albumname     | cost | numbersold
------------+---------------+------+------------
 Katy Perry | Teenage Dream |   14 |     750000
 Katy Perry |         Prism |   16 |     800000

(2 rows)
cqlsh:a20506653>
```

Exercise 5) (1 point)

Now create a file in your working directory called ex5.cql using the Edit-Term. In this file write the commands to query only albums that have sold 700000 copies or more. Execute ex5.cql. Provide the content of this file and the output of executing this file as the result of this exercise.

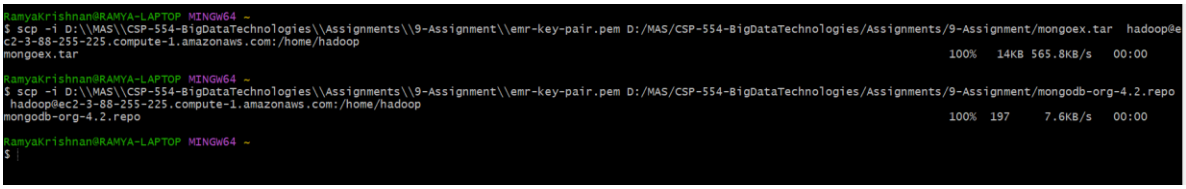
```
select * from music where numbersold >= 700000 ALLOW FILTERING;
```

```
cqlsh:a20506653> source './ex5.cql';

 artistname   | albumname     | cost | numbersold
--------------+---------------+------+------------
 Taylor Swift |      Fearless |   15 |    2300000
   Katy Perry | Teenage Dream |   14 |     750000
   Katy Perry |         Prism |   16 |     800000

(3 rows)
cqlsh:a20506653>
```

## MongoDB

Step A – Start an EMR cluster


Step B – Download the assignment software (mongoex.tar, mongodb-org-4.2.repo) to master node



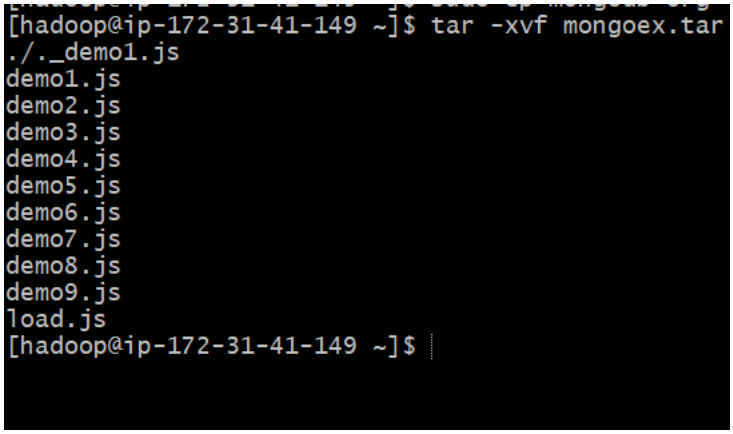

Step C – Install assignment software (mongoex.zip, mongodb-org-4.2.repo)

Enter the following into a terminal window which you have connected to the EMR master node. Going forward we will call this terminal connection Init-Term:

sudo cp mongodb-org-4.2.repo /etc/yum.repos.d

Then enter this into Init-Term to unzip mongoex.tar:

tar -xvf mongoex.tar



Step D – Install and start MongoDB

Enter the following into Init-Term to install MongoDB:

sudo yum install -y mongodb-org-4.2.15 mongodb-org-server-4.2.15 mongodb-org-shell-4.2.15 mongodb-org-mongos-4.2.15 mongodb-org-tools-4.2.15

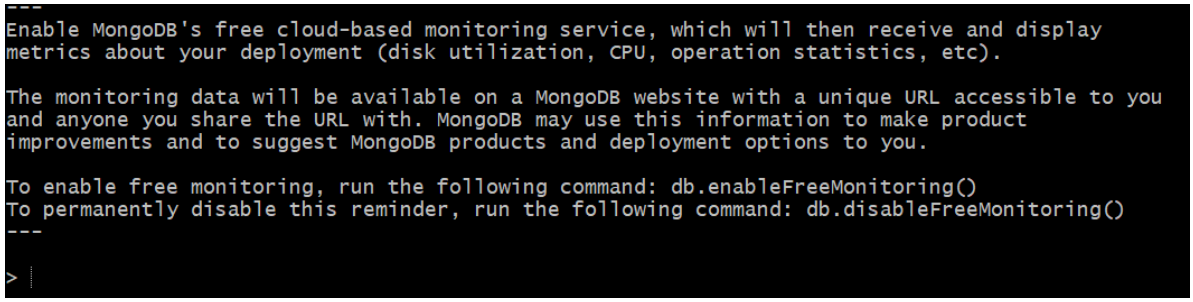

Now enter this into Init-Term to start mongodb:

sudo systemctl start mongod

Step E – Start the MongoDB Shell (Command Line Interpreter)

Open a second terminal connection to the EMR master node. Going forward we will call this terminal connection: CLI-Term.
You will use this terminal window to start and run the mongodb shell as follows:

mongo




Step F – Edit mongo query language files

Open a third terminal connection to the EMR master node. Going forward we will call this terminal connection: CLI-Term. You will use this terminal window to run the 'vi' editor to create your Mongo code files.

As an alternative you could edit your MongoDB code files on your PC/MAC and then 'scp' them to the EMR mater node.

Step G – Setting up the assignment database

Now, in the MongoDB shell, using the CLI-Term, create a database called "assignment" by entering the following into the MongoDB shell:

use assignment;

```
> use assignment;
switched to db assignment
>
```

This will set the shell variable ‘db’ to this new database.

Load a collection called ‘unicorns’ with sample data by executing the script load.js in the MongoDB shell as follows (don’t cut and paste this, type it in manually):

load(‘./load.js’);

```
> load('./load.js');
true
>
```

Note, look at the content of the script file (via the other terminal window you have opened to the EC2 instance) to see how each unicorn is described.

Confirm this has all worked by executing the following command in the MongoDB shell:

db.unicorns.find();


Note, the files named “demo*.js” (also included in the mongoex.tar file) provide examples of how to operate in the unicorn collection. These are a VERY good idea to review and understand and will present you with information helpful in completing the assignment. Also, try them out by typing something like

load(./demo1.js’);

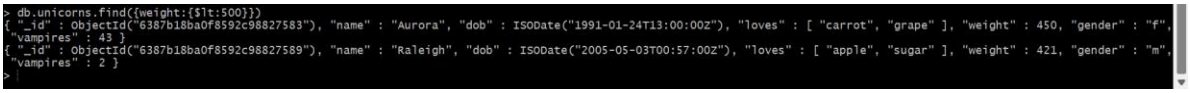
```
> load('./demo1.js');
true
>
```

## Exercises:

Exercise 1) (1 point)

Write a command that finds all unicorns having weight less than 500 pounds. Include the code you executed and some sample output as the result of this exercise. Recall you can place the command, if you choose, into a file, say 'ex1.js' and execute it with the load command as above and similarly for the following exercises.

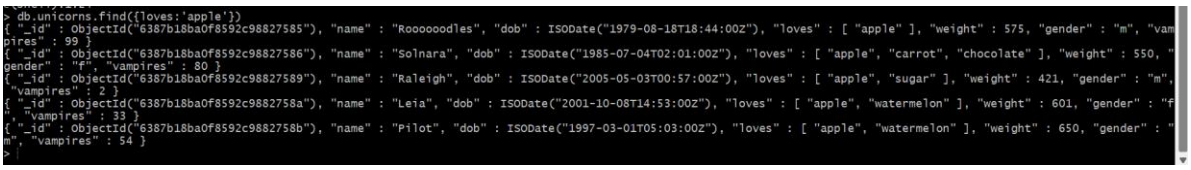
```
db.unicorns.find({weight:{$lt:500}})
```




Exercise 2) (1 point)

Write a command that finds all unicorns who love apples. Hint, search for "apple". Include the code you executed and some sample output as the result of this exercise.

```
db.unicorns.find({loves:'apple'})
```




Exercise 3) (1 point)

Write a command that adds a unicorn with the following attributes to the collection. Note dob means "Date of Birth."

| Attribute | Value(s) |
|---|---|
| **name** | Malini |
| **dob** | 11/03/2008 |
| **loves** | pears, grapes |
| **weight** | 450 |
| **gender** | F |
| **vampires** | 23 |
| **horns** | 1 |

Include the code you executed to insert this unicorn into the collection along with the output of a find command showing it is in the collection.

```
db.unicorns.insertOne({name:'Malini', dob:'2008-03-11',
loves:['pears','grapes'], weight:450, gender:'F', vampires:23, horns:1})
```

```
> db.unicorns.insertOne({name:'Malini', dob:'2008-03-11', loves:['pears','grapes'], weight:450, gender:'F', vampires:23, horns:1})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6387b3f2a0f8592c9882758f")
}
>
```

```
> db.unicorns.find();
{ "_id" : ObjectId("6387b18ba0f8592c98827582"), "name" : "Horny", "dob" : ISODate("1992-03-13T07:47:00Z"), "loves" : [ "carrot", "papaya" ], "weight" : 600, "gender" : "m",
 "vampires" : 63 }
{ "_id" : ObjectId("6387b18ba0f8592c98827583"), "name" : "Aurora", "dob" : ISODate("1991-01-24T13:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "gender" : "f",
 "vampires" : 43 }
{ "_id" : ObjectId("6387b18ba0f8592c98827584"), "name" : "Unicrom", "dob" : ISODate("1973-02-09T22:10:00Z"), "loves" : [ "energon", "redbull" ], "weight" : 984, "gender" :
"m", "vampires" : 182 }
{ "_id" : ObjectId("6387b18ba0f8592c98827585"), "name" : "Roooooodles", "dob" : ISODate("1979-08-18T18:44:00Z"), "loves" : [ "apple" ], "weight" : 575, "gender" : "m", "vam
pires" : 99 }
{ "_id" : ObjectId("6387b18ba0f8592c98827586"), "name" : "Solnara", "dob" : ISODate("1985-07-04T02:01:00Z"), "loves" : [ "apple", "carrot", "chocolate" ], "weight" : 550, "
gender" : "f", "vampires" : 80 }
{ "_id" : ObjectId("6387b18ba0f8592c98827587"), "name" : "Ayna", "dob" : ISODate("1998-03-07T08:30:00Z"), "loves" : [ "strawberry", "lemon" ], "weight" : 733, "gender" : "f
", "vampires" : 40 }
{ "_id" : ObjectId("6387b18ba0f8592c98827588"), "name" : "Kenny", "dob" : ISODate("1997-07-01T10:42:00Z"), "loves" : [ "grape", "lemon" ], "weight" : 690, "gender" : "m", "
vampires" : 39 }
{ "_id" : ObjectId("6387b18ba0f8592c98827589"), "name" : "Raleigh", "dob" : ISODate("2005-05-03T00:57:00Z"), "loves" : [ "apple", "sugar" ], "weight" : 421, "gender" : "m",
 "vampires" : 2 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758a"), "name" : "Leia", "dob" : ISODate("2001-10-08T14:53:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 601, "gender" : "f
", "vampires" : 33 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758b"), "name" : "Pilot", "dob" : ISODate("1997-03-01T05:03:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 650, "gender" : "
m", "vampires" : 54 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758c"), "name" : "Nimue", "dob" : ISODate("1999-12-20T16:15:00Z"), "loves" : [ "grape", "carrot" ], "weight" : 540, "gender" : "f" }
{ "_id" : ObjectId("6387b18ba0f8592c9882758d"), "name" : "Dunx", "dob" : ISODate("1976-07-18T18:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704, "gender" : "m
", "vampires" : 165 }
{ "_id" : ObjectId("6387b1f3a0f8592c9882758e"), "name" : "Leto", "gender" : "m", "home" : "Arrakeen", "worm" : false }
{ "_id" : ObjectId("6387b3f2a0f8592c9882758f"), "name" : "Malini", "dob" : "2008-03-11", "loves" : [ "pears", "grapes" ], "weight" : 450, "gender" : "F", "vampires" : 23, "
horns" : 1 }
```

Exercise 4) (1 point)

Write a command that updates the above record to add apricots to the list of things Malini loves. Include the code you executed and some sample output showing the addition.

```
db.unicorns.updateOne({loves:['pears','grapes']},{$set:{loves:['pears','grap
es','apricots']}})
```

```
> db.unicorns.updateOne({loves:['pears','grapes']},{$set:{loves:['pears','grapes','apricots']}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

```
> db.unicorns.find();
{ "_id" : ObjectId("6387b18ba0f8592c98827582"), "name" : "Horny", "dob" : ISODate("1992-03-13T07:47:00Z"), "loves" : [ "carrot", "papaya" ], "weight" : 600, "gender" : "m",
 "vampires" : 63 }
{ "_id" : ObjectId("6387b18ba0f8592c98827583"), "name" : "Aurora", "dob" : ISODate("1991-01-24T13:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "gender" : "f",
 "vampires" : 43 }
{ "_id" : ObjectId("6387b18ba0f8592c98827584"), "name" : "Unicrom", "dob" : ISODate("1973-02-09T22:10:00Z"), "loves" : [ "energon", "redbull" ], "weight" : 984, "gender" :
"m", "vampires" : 182 }
{ "_id" : ObjectId("6387b18ba0f8592c98827585"), "name" : "Roooooodles", "dob" : ISODate("1979-08-18T18:44:00Z"), "loves" : [ "apple" ], "weight" : 575, "gender" : "m", "vam
pires" : 99 }
{ "_id" : ObjectId("6387b18ba0f8592c98827586"), "name" : "Solnara", "dob" : ISODate("1985-07-04T02:01:00Z"), "loves" : [ "apple", "carrot", "chocolate" ], "weight" : 550, "
gender" : "f", "vampires" : 80 }
{ "_id" : ObjectId("6387b18ba0f8592c98827587"), "name" : "Ayna", "dob" : ISODate("1998-03-07T08:30:00Z"), "loves" : [ "strawberry", "lemon" ], "weight" : 733, "gender" : "f
", "vampires" : 40 }
{ "_id" : ObjectId("6387b18ba0f8592c98827588"), "name" : "Kenny", "dob" : ISODate("1997-07-01T10:42:00Z"), "loves" : [ "grape", "lemon" ], "weight" : 690, "gender" : "m", "
vampires" : 39 }
{ "_id" : ObjectId("6387b18ba0f8592c98827589"), "name" : "Raleigh", "dob" : ISODate("2005-05-03T00:57:00Z"), "loves" : [ "apple", "sugar" ], "weight" : 421, "gender" : "m",
 "vampires" : 2 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758a"), "name" : "Leia", "dob" : ISODate("2001-10-08T14:53:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 601, "gender" : "f
", "vampires" : 33 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758b"), "name" : "Pilot", "dob" : ISODate("1997-03-01T05:03:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 650, "gender" : "
m", "vampires" : 54 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758c"), "name" : "Nimue", "dob" : ISODate("1999-12-20T16:15:00Z"), "loves" : [ "grape", "carrot" ], "weight" : 540, "gender" : "f" }
{ "_id" : ObjectId("6387b18ba0f8592c9882758d"), "name" : "Dunx", "dob" : ISODate("1976-07-18T18:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704, "gender" : "m
", "vampires" : 165 }
{ "_id" : ObjectId("6387b1f3a0f8592c9882758e"), "name" : "Leto", "gender" : "m", "home" : "Arrakeen", "worm" : false }
{ "_id" : ObjectId("6387b3f2a0f8592c9882758f"), "name" : "Malini", "dob" : "2008-03-11", "loves" : [ "pears", "grapes", "apricots" ], "weight" : 450, "gender" : "F", "vampi
res" : 23, "horns" : 1 }
>
```

Exercise 5) (1 point)

Write a command that deletes all unicorns with weight more than 600 pounds. Include the code you executed and some sample output as the result of this exercise.

```
db.unicorns.deleteMany({weight:{$gt:600}})
```

```
> db.unicorns.deleteMany({weight:{$gt:600}})
{ "acknowledged" : true, "deletedCount" : 6 }
>
```


> db.unicorns.find();
{ "_id" : ObjectId("6387b18ba0f8592c98827582"), "name" : "Horny", "dob" : ISODate("1992-03-13T07:47:00Z"), "loves" : [ "carrot", "papaya" ], "weight" : 600, "gender" : "m",
"vampires" : 63 }
{ "_id" : ObjectId("6387b18ba0f8592c98827583"), "name" : "Aurora", "dob" : ISODate("1991-01-24T13:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "gender" : "f",
"vampires" : 43 }
{ "_id" : ObjectId("6387b18ba0f8592c98827585"), "name" : "Roooooodles", "dob" : ISODate("1979-08-18T18:44:00Z"), "loves" : [ "apple" ], "weight" : 575, "gender" : "m", "vam
pires" : 99 }
{ "_id" : ObjectId("6387b18ba0f8592c98827586"), "name" : "Solnara", "dob" : ISODate("1985-07-04T02:01:00Z"), "loves" : [ "apple", "carrot", "chocolate" ], "weight" : 550, "
gender" : "f", "vampires" : 80 }
{ "_id" : ObjectId("6387b18ba0f8592c98827589"), "name" : "Raleigh", "dob" : ISODate("2005-05-03T00:57:00Z"), "loves" : [ "apple", "sugar" ], "weight" : 421, "gender" : "m",
"vampires" : 2 }
{ "_id" : ObjectId("6387b18ba0f8592c9882758c"), "name" : "Nimue", "dob" : ISODate("1999-12-20T16:15:00Z"), "loves" : [ "grape", "carrot" ], "weight" : 540, "gender" : "f" }
{ "_id" : ObjectId("6387b1f3a0f8592c9882758e"), "name" : "Leto", "gender" : "m", "home" : "Arrakeen", "worm" : false }
{ "_id" : ObjectId("6387b3f2a0f8592c9882758f"), "name" : "Malini", "dob" : "2008-03-11", "loves" : [ "pears", "grapes", "apricots" ], "weight" : 450, "gender" : "F", "vampi
res" : 23, "horns" : 1 }
>