1. **Team Name**

   Jamba

2. **Project Name**

   Book Reviews Search Tool

3. **Project Description**

   Our application is made to help people decide on which books they are interested in by giving them an ability to search through reviews fluidly with specific queries tailored to their needs. We have millions of reviews stored across our database that will be stored at different shards to ensure that parts of the database will always be available. Users are able to search up different reviews based on their price, title, and book ID. We've created an environment where users who are looking to find a book or looking to learn more about a book can discover more information about other people's thoughts on the book. Since the data does not need to be structured with relationships, it is much easier to have large amounts of data distributed across multiple servers. Ultimately, making the queries much quicker and more efficient.

4. **Project Successes**

   - Deploying sharded cluster with using AWS
   - Connecting Python application to a AWS instance
   - Creating 15 function in the application to query the database
   - Successfully converting data from .csv format to .json
   - Good communication with team members and even distribution of the tasks

5. **Unexpected Events**

   - Leaving of one of the team members
     - Increased the workload on the other members
   - Shards stopped working several times (probably because of lack of memory/space)
     - Started using large AWS instances with increased storage capacity (20GB)
   - Incorrect work of the application
     - Changed data types and the logic of the application

6. **Lessons Learned**

   - Time management
   - Better to create instances with bigger disk and RAM capacity
   - Easy to make a mistake while setting up the database

- The wrong choice of the dataset might lead to more work depending on the type of queries you are trying to make (no array in ours for elemmatch)
- Recommendations:
    - Start early, have a plan written
    - Having notes with all the steps for setting up the database helps to save time
    - Be careful when choosing the dataset

7. **DataSet**

https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews

A large set of book reviews with the reviews and summary, book title, book_id, etc. Size - 3.32 GB.

Using Google collab, We downloaded the csv file and analyzed which data was null or not and then removed all the values with null “Title” values. We also decided to put a default price of 0.0 for the null values and “Unknown” for any other blank fields. We had to **remove half** of the values because the dataset was too large. **The final size of the dataset used for the app is 1.66GB.** Afterwards, we transferred the fixed dataframe with replaced null values into a Json file.

https://colab.research.google.com/drive/1CTT-HJkSnQkhkz3gB93L2Nzh2sL6LVS1?usp=sharing

8. **Database Schema description**

Database name - books

Collection - reviews

Each review includes the Id of the book, just like the Id of the user who left this review, the title of the book, and its price. Besides the score, the dataset also has the helpfulness score (how helpful the review is). In addition to it, each user has his own profile name which is also a part of the review. Reviews also have quick summary, and the main text with the review. Another column is the time that was spent on the review.

Indexes:

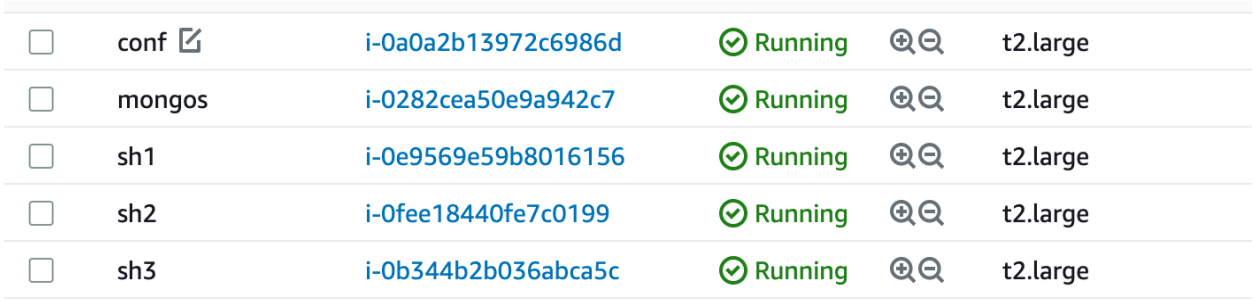
```
{ v: 2, key: { _id: 1 }, name: '_id_' },
 { v: 2, key: { Id: 'hashed' }, name: 'Id_hashed' }
```

Also created an index on the summary so that users can search for keywords within the summary:

db.reviews.createIndex({'review/summary':"text", 'review/text': "text"})

9. **NoSQL configuration**

Our team decided to use 5 nodes for this project. 1 for mongos, 1 for config server, and 3 for shards.



☐	conf	i-0a0a2b13972c6986d	Running		t2.large
☐	mongos	i-0282cea50e9a942c7	Running		t2.large
☐	sh1	i-0e9569e59b8016156	Running		t2.large
☐	sh2	i-0fee18440fe7c0199	Running		t2.large
☐	sh3	i-0b344b2b036abca5c	Running		t2.large

CONF 34.227.221.99 ec2-34-227-221-99.compute-1.amazonaws.com

MONGOS 34.229.203.66 ec2-34-229-203-66.compute-1.amazonaws.com

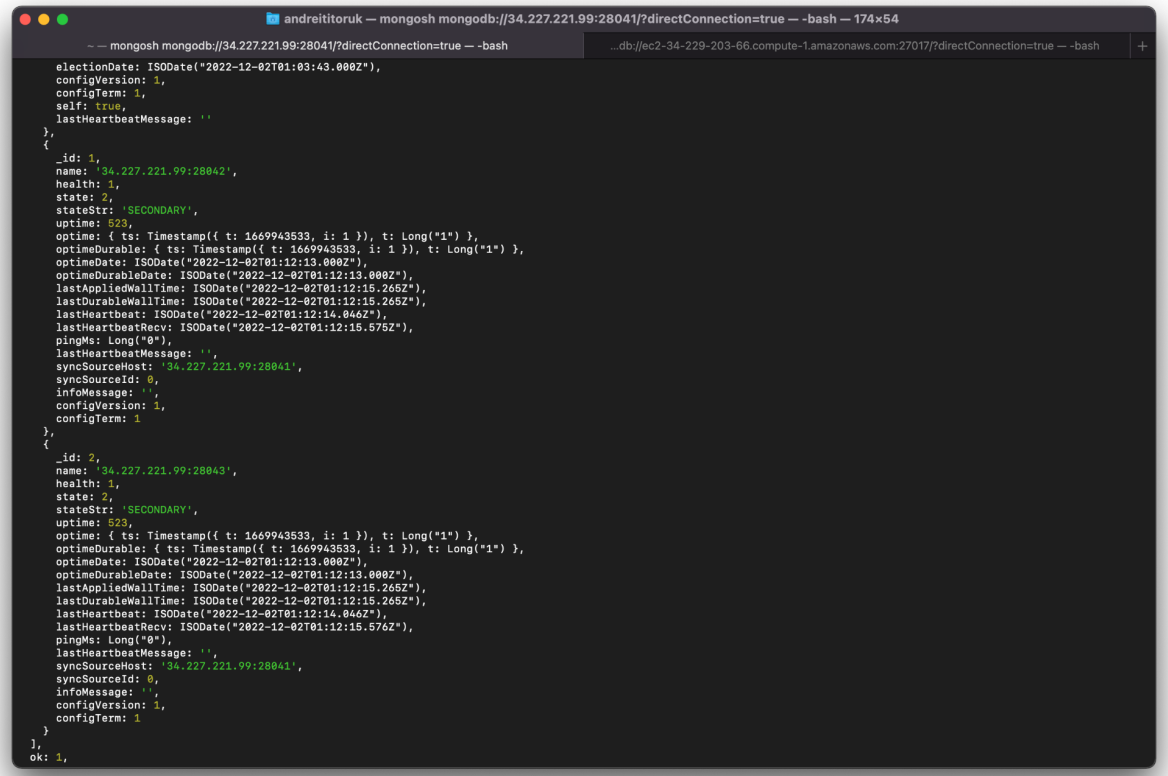
SH1. 54.196.153.249 ec2-54-196-153-249.compute-1.amazonaws.com

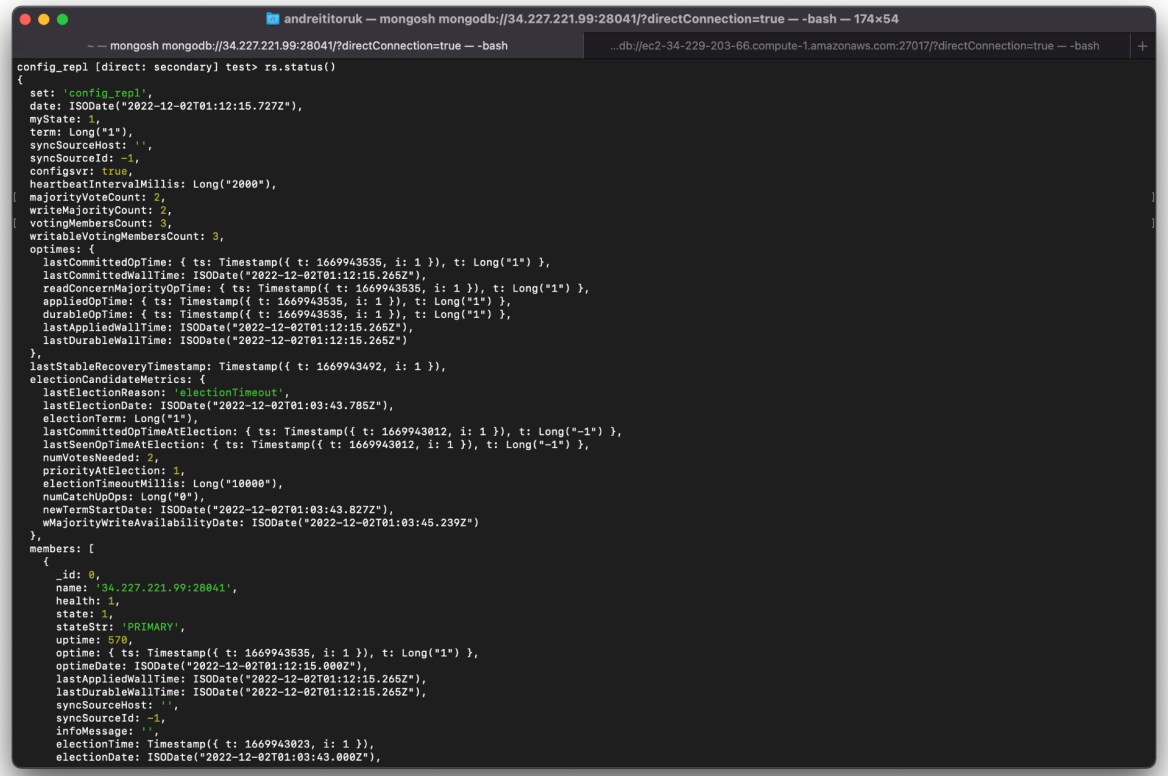
SH2 54.172.50.63 ec2-54-172-50-63.compute-1.amazonaws.com

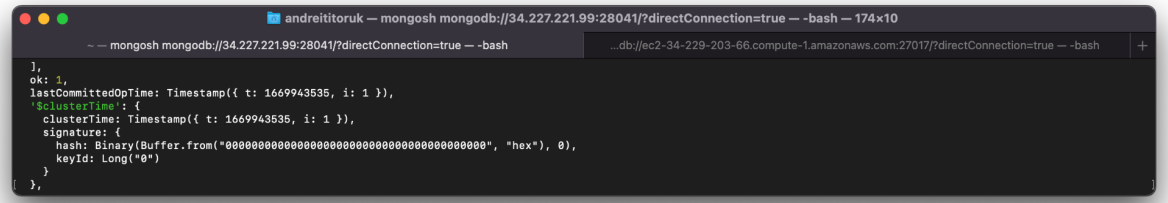
SH3 54.242.255.128 ec2-54-242-255-128.compute-1.amazonaws.com

Ports used:
node0:28041 config server PRIMARY (ports 28042 and 28043 are SECONDARY)
Node1:27017 mongos
node2:28081 Shard1 PRIMARY (ports 28082 and 28083 are SECONDARY)
Node3:29081 Shard2 PRIMARY (ports 29082 and 29083 are SECONDARY)
Node4:26001 Shard3 PRIMARY (ports 26002 and 26003 are SECONDARY)

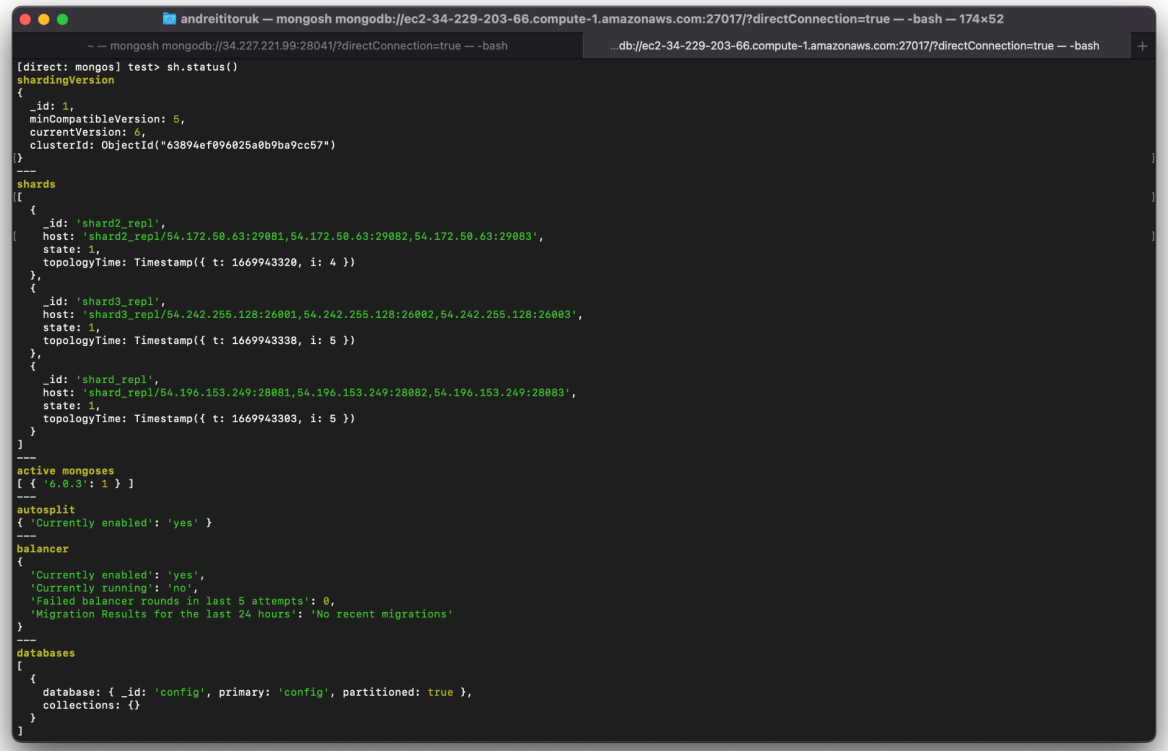
Running sh.status() in conf.


andreititoruk — mongosh mongodb://34.227.221.99:28041/?directConnection=true — -bash — 174×54
~ — mongosh mongodb://34.227.221.99:28041/?directConnection=true — -bash
...db://ec2-34-229-203-66.compute-1.amazonaws.com:27017/?directConnection=true — -bash
electionDate: ISODate("2022-12-02T01:03:43.000Z"),
configVersion: 1,
configTerm: 1,
self: true,
lastHeartbeatMessage: ''
},
{
_id: 1,
name: '34.227.221.99:28042',
health: 1,
state: 2,
stateStr: 'SECONDARY',
uptime: 523,
optime: { ts: Timestamp({ t: 1669943533, i: 1 }), t: Long("1") },
optimeDurable: { ts: Timestamp({ t: 1669943533, i: 1 }), t: Long("1") },
optimeDate: ISODate("2022-12-02T01:12:13.000Z"),
optimeDurableDate: ISODate("2022-12-02T01:12:13.000Z"),
lastAppliedWallTime: ISODate("2022-12-02T01:12:15.265Z"),
lastDurableWallTime: ISODate("2022-12-02T01:12:15.265Z"),
lastHeartbeat: ISODate("2022-12-02T01:12:14.046Z"),
lastHeartbeatRecv: ISODate("2022-12-02T01:12:15.575Z"),
pingMs: Long("0"),
lastHeartbeatMessage: '',
syncSourceHost: '34.227.221.99:28041',
syncSourceId: 0,
infoMessage: '',
configVersion: 1,
configTerm: 1
},
{
_id: 2,
name: '34.227.221.99:28043',
health: 1,
state: 2,
stateStr: 'SECONDARY',
uptime: 523,
optime: { ts: Timestamp({ t: 1669943533, i: 1 }), t: Long("1") },
optimeDurable: { ts: Timestamp({ t: 1669943533, i: 1 }), t: Long("1") },
optimeDate: ISODate("2022-12-02T01:12:13.000Z"),
optimeDurableDate: ISODate("2022-12-02T01:12:13.000Z"),
lastAppliedWallTime: ISODate("2022-12-02T01:12:15.265Z"),
lastDurableWallTime: ISODate("2022-12-02T01:12:15.265Z"),
lastHeartbeat: ISODate("2022-12-02T01:12:14.046Z"),
lastHeartbeatRecv: ISODate("2022-12-02T01:12:15.576Z"),
pingMs: Long("0"),
lastHeartbeatMessage: '',
syncSourceHost: '34.227.221.99:28041',
syncSourceId: 0,
infoMessage: '',
configVersion: 1,
configTerm: 1
}
],
ok: 1,

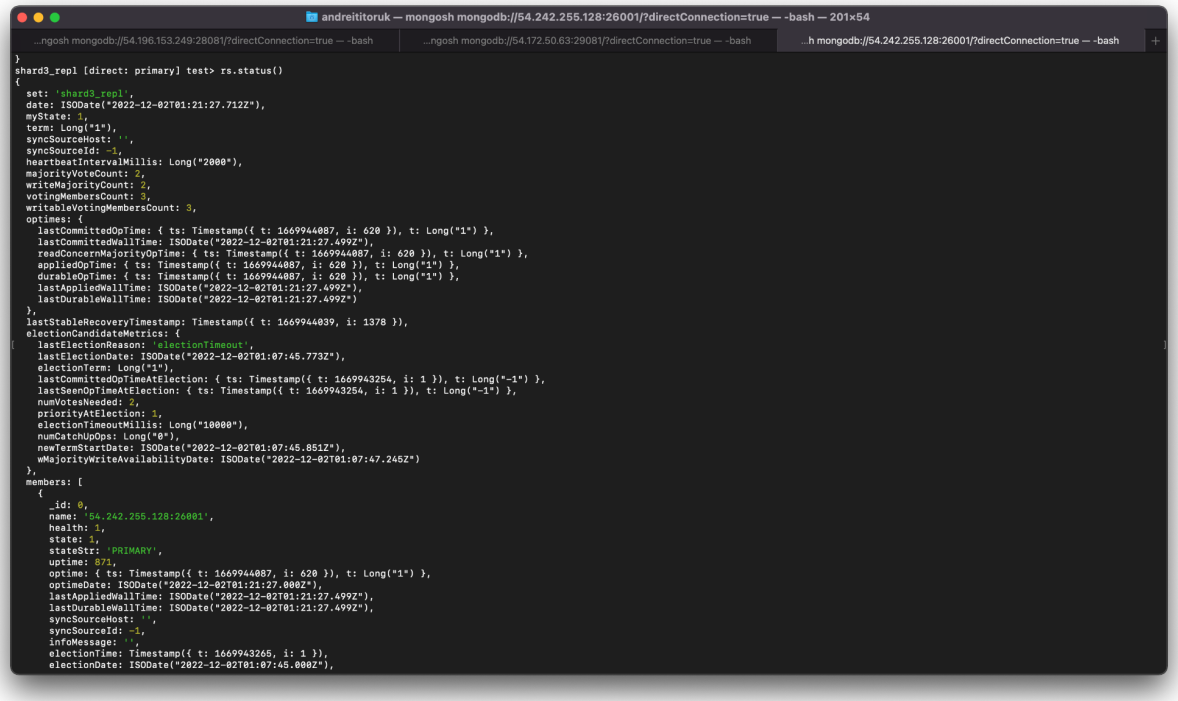


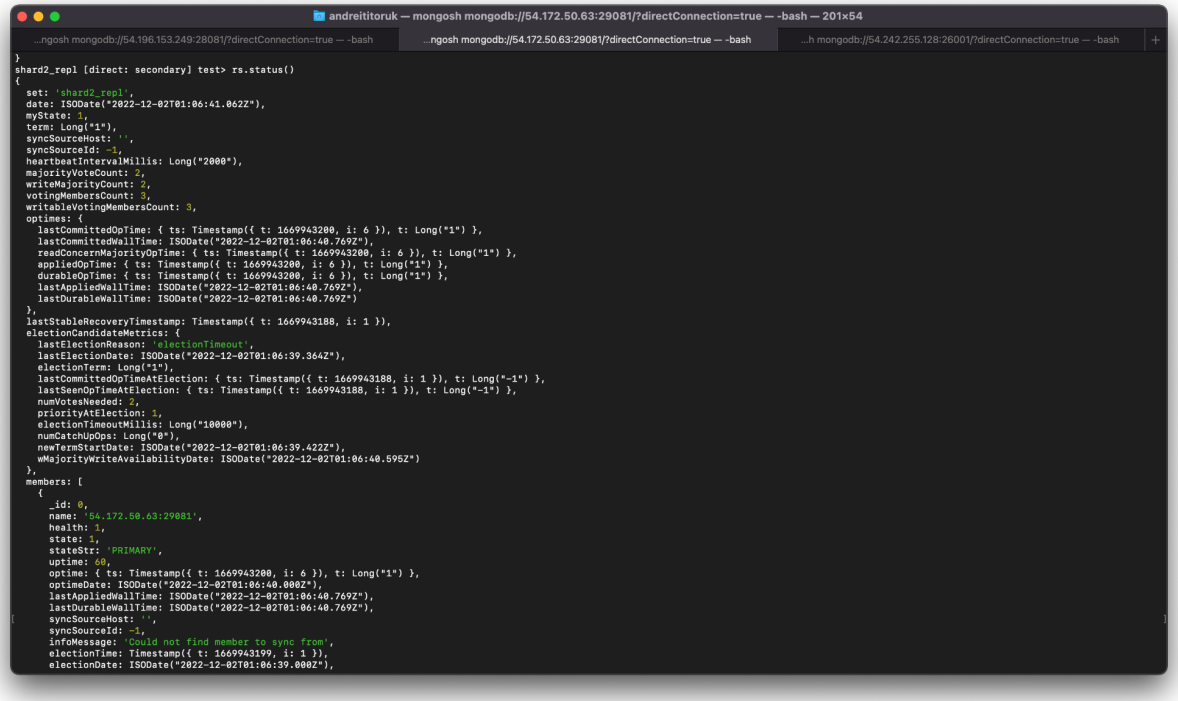


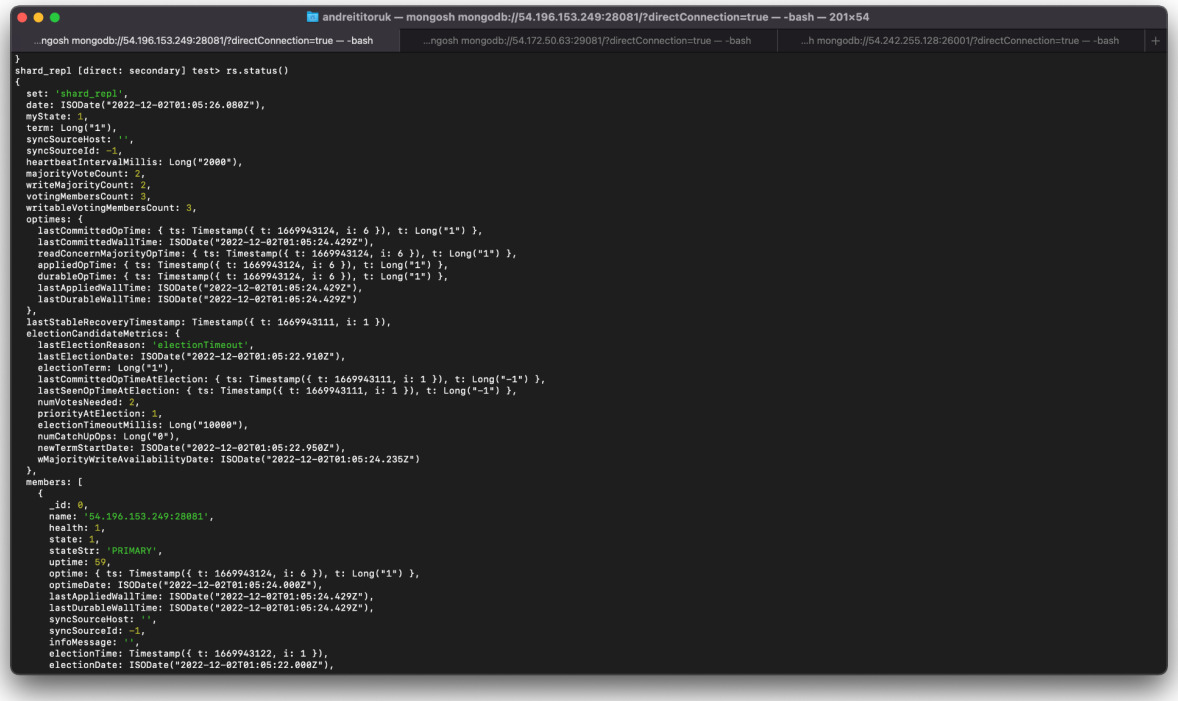
sh.status() in mongos after adding shards



rs.status() in shards


andreititoruk — mongosh mongodb://54.242.255.128:26001/?directConnection=true — -bash — 201×54
...ngosh mongodb://54.196.153.249:28081/?directConnection=true — -bash
...ngosh mongodb://54.172.50.63:29081/?directConnection=true — -bash
...h mongodb://54.242.255.128:26001/?directConnection=true — -bash
}
shard3_repl [direct: primary] test> rs.status()
{
set: 'shard3_repl',
date: ISODate("2022-12-02T01:21:27.712Z"),
myState: 1,
term: Long("1"),
syncSourceHost: '',
syncSourceId: -1,
heartbeatIntervalMillis: Long("2000"),
majorityVoteCount: 2,
writeMajorityCount: 2,
votingMembersCount: 3,
writableVotingMembersCount: 3,
optimes: {
lastCommittedOpTime: { ts: Timestamp({ t: 1669944087, i: 620 }), t: Long("1") },
lastCommittedWallTime: ISODate("2022-12-02T01:21:27.499Z"),
readConcernMajorityOpTime: { ts: Timestamp({ t: 1669944087, i: 620 }), t: Long("1") },
appliedOpTime: { ts: Timestamp({ t: 1669944087, i: 620 }), t: Long("1") },
durableOpTime: { ts: Timestamp({ t: 1669944087, i: 620 }), t: Long("1") },
lastAppliedWallTime: ISODate("2022-12-02T01:21:27.499Z"),
lastDurableWallTime: ISODate("2022-12-02T01:21:27.499Z")
},
lastStableRecoveryTimestamp: Timestamp({ t: 1669944039, i: 1378 }),
electionCandidateMetrics: {
lastElectionReason: 'electionTimeout',
lastElectionDate: ISODate("2022-12-02T01:07:45.773Z"),
electionTerm: Long("1"),
lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1669943254, i: 1 }), t: Long("-1") },
lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1669943254, i: 1 }), t: Long("-1") },
numVotesNeeded: 2,
priorityAtElection: 1,
electionTimeoutMillis: Long("10000"),
numCatchUpOps: Long("0"),
newTermStartDate: ISODate("2022-12-02T01:07:45.851Z"),
wMajorityWriteAvailabilityDate: ISODate("2022-12-02T01:07:47.245Z")
},
members: [
{
_id: 0,
name: '54.242.255.128:26001',
health: 1,
state: 1,
stateStr: 'PRIMARY',
uptime: 871,
optime: { ts: Timestamp({ t: 1669944087, i: 620 }), t: Long("1") },
optimeDate: ISODate("2022-12-02T01:21:27.000Z"),
lastAppliedWallTime: ISODate("2022-12-02T01:21:27.499Z"),
lastDurableWallTime: ISODate("2022-12-02T01:21:27.499Z"),
syncSourceHost: '',
syncSourceId: -1,
infoMessage: '',
electionTime: Timestamp({ t: 1669943265, i: 1 }),
electionDate: ISODate("2022-12-02T01:07:45.000Z"),


andreititoruk — mongosh mongodb://54.172.50.63:29081/?directConnection=true — -bash — 201×54
...ngosh mongodb://54.196.153.249:28081/?directConnection=true — -bash
...ngosh mongodb://54.172.50.63:29081/?directConnection=true — -bash
...h mongodb://54.242.255.128:26001/?directConnection=true — -bash
}
shard2_repl [direct: secondary] test> rs.status()
{
set: 'shard2_repl',
date: ISODate("2022-12-02T01:06:41.062Z"),
myState: 1,
term: Long("1"),
syncSourceHost: '',
syncSourceId: -1,
heartbeatIntervalMillis: Long("2000"),
majorityVoteCount: 2,
writeMajorityCount: 2,
votingMembersCount: 3,
writableVotingMembersCount: 3,
optimes: {
lastCommittedOpTime: { ts: Timestamp({ t: 1669943200, i: 6 }), t: Long("1") },
lastCommittedWallTime: ISODate("2022-12-02T01:06:40.769Z"),
readConcernMajorityOpTime: { ts: Timestamp({ t: 1669943200, i: 6 }), t: Long("1") },
appliedOpTime: { ts: Timestamp({ t: 1669943200, i: 6 }), t: Long("1") },
durableOpTime: { ts: Timestamp({ t: 1669943200, i: 6 }), t: Long("1") },
lastAppliedWallTime: ISODate("2022-12-02T01:06:40.769Z"),
lastDurableWallTime: ISODate("2022-12-02T01:06:40.769Z")
},
lastStableRecoveryTimestamp: Timestamp({ t: 1669943188, i: 1 }),
electionCandidateMetrics: {
lastElectionReason: 'electionTimeout',
lastElectionDate: ISODate("2022-12-02T01:06:39.364Z"),
electionTerm: Long("1"),
lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1669943188, i: 1 }), t: Long("-1") },
lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1669943188, i: 1 }), t: Long("-1") },
numVotesNeeded: 2,
priorityAtElection: 1,
electionTimeoutMillis: Long("10000"),
numCatchUpOps: Long("0"),
newTermStartDate: ISODate("2022-12-02T01:06:39.422Z"),
wMajorityWriteAvailabilityDate: ISODate("2022-12-02T01:06:40.595Z")
},
members: [
{
_id: 0,
name: '54.172.50.63:29081',
health: 1,
state: 1,
stateStr: 'PRIMARY',
uptime: 60,
optime: { ts: Timestamp({ t: 1669943200, i: 6 }), t: Long("1") },
optimeDate: ISODate("2022-12-02T01:06:40.000Z"),
lastAppliedWallTime: ISODate("2022-12-02T01:06:40.769Z"),
lastDurableWallTime: ISODate("2022-12-02T01:06:40.769Z"),
syncSourceHost: '',
syncSourceId: -1,
infoMessage: 'Could not find member to sync from',
electionTime: Timestamp({ t: 1669943199, i: 1 }),
electionDate: ISODate("2022-12-02T01:06:39.000Z"),



10. **Explain the rationale behind your shard key and sharding strategy selection.**

The reason we decided on using the _id as the shard key is because they each have a unique value and we could evenly distribute the book reviews across the different shards once we hashed the value. We could have considered hashing values of the same book title together, so that they would be in the same place, but it's more efficient to have the data evenly distributed for querying the data.

11. **Brief description on the NoSQL driver**

Currently we are using Python as our high-level language with Pymongo as the driver. This allows us to use similar query functions such as find_one() and find() and many others in order to pull the data from our database. It is a way of communicating to our mongos router in order to query, retrieve, write and delete data, and run database commands from the client side. It is commonly used for synchronous applications, such as our own.

A connection is established between the application and the server through pymongo's MongoClient. First, we create a MongoClient object with the Mongos router input as a

parameter. Then we try pinging to the server to see if we have a connection established. As show in our try, except catch clause:

```
    # first step, download pymongo
    client = MongoClient("ec2-34-227-49-76.compute-1.amazonaws.com")
    try:
        # The ping command is cheap and does not require auth.
        client.admin.command('ping')
    except ConnectionFailure:
        print("Server not available")
```

If connection is established, we can now access our Mongos router's database and collection.

12. **For each 15 function, include the following items**
    a. One line of description for that function
    b. DDL and/or DML commands of your NoSQL choice to fulfill the function
    c. Screenshot(s) to CLEARLY show the function works.

NOTE These are the outputs from our Python application after running this main function

```
# first step, download pymongo
client = MongoClient("ec2-34-227-49-76.compute-1.amazonaws.com") # insert mongos client into quotations
try:
    # The ping command is cheap and does not require auth.
    client.admin.command('ping')
except ConnectionFailure:
    print("Server not available")
db = client.books # Replace x with the database name
print(db.list_collection_names()) # Prints collection names
```

```
Alexs-MBP:~ alexong$ /usr/local/bin/python3.8 /Users/alexong/CS157C/CS157C.py
['reviews']
```

1. Add a book review:

```python
def insertReview(_id, Title, Price, User_id, profileName, helpfulness, score, time, summary, text):
    dictionary = {
        "Id": _id,
        "Title": Title,
        "Price": Price,
        "User_id": User_id,
        "profileName": profileName,
        "review/helpfulness": helpfulness,
        "review/score": score,
        "review/time": time,
        "review/summary": summary,
        "review/text": text
        }
    rev = db.reviews.insert_one(dictionary)
    print("Document was created")

def insertIO():
    print("Inserting review")

    _id = input("Enter _id: ")

    title = input("Enter book title: ")

    price = input("Enter price: ")
    try:
        price = float(price)
    except:
        print("Error not float")
        return

    user_id = input("Enter userID: ")


    profileName = input("Enter profile name: ")

    helpfulness = "TBD"
    score = input("Enter rating: ")
    times = time.time()

    summary = input("Enter summary: ")
    text = input("Enter review: ")
    insertReview(_id, title, price, user_id, profileName, helpfulness, score, times, summary, text)
```

```
Alexs-MBP:~ alexong$ /usr/local/bin/python3.8 /Users/alexong/CS157C/CS157C.py
['reviews']
Inserting review
Enter _id: 1998
Enter book title: Ducktective
Enter price: 3.60
Enter userID: AH889
Enter profile name: Andrei T
Enter rating: 5/5
Enter summary: Ducks and mystery
Enter review: It was an amazing book about ducks
Document was created
```

2. Deleting a book review

```
def deleteReview(Id):
    rev = db.reviews.delete_one({'Id': Id})
    print(rev.raw_result)

def deleteIO():
    print("Deleting review")
    _id = input("Enter review ID: ")
    deleteReview(_id)
```

```
Deleting review
Enter review ID: 0826414346
{'n': 1, 'ok': 1.0, '$clusterTime': {'clusterTime': Timestamp(1669937274, 1), 'signature': {
'hash': b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
 'keyId': 0}}, 'operationTime': Timestamp(1669937274, 1)}
```

3. Updating a book review price

```
def updateReviewPrice(Id, price):
    rev = db.reviews.update_one({'Id': Id}, {'$set': {'Price': price}})
    print(rev.raw_result)

def updateIO():
    print("Updating review price")
    _id = input("Enter ID of review: ")

    price = input("Enter new price: ")
    try:
        price = float(price)
    except:
        print("Error not float")
        return

    updateReviewPrice(_id, price)
```

```
Updating review price
Enter ID of review: 0595344550
Enter new price: 7.00
{'nModified': 1, 'n': 1, 'ok': 1.0, '$clusterTime': {'clusterTime': Timestamp(1669937430, 1)
, 'signature': {'hash': b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00', 'keyId': 0}}, 'operationTime': Timestamp(1669937430, 1), 'updatedExisting':
 True}
```

4. Find review based on Title

```
def  all: Cursor itle):
    all = db.reviews.find({'Title': title}).limit(10)
    for document in all:
        print(document)

def findByTitleIO():
    print("Find reviews based on Title")
    title = input("Enter Title: ") # Ex: Dr. Seuss: American Icon
    findByTitle(title)
```

```
Find reviews based on Title
Enter Title: Whispers of the Wicked Saints
{'_id': ObjectId('63892ad42f74734999771137'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 7.0, 'User_id': 'A3Q12RK71N74LB', 'profileName': 'Book Reader', 're
view/helpfulness': '7/11', 'review/score': 1, 'review/time': 1117065600, 'review/summary': '
not good', 'review/text': "I bought this book because I read some glowing praise on an onlin
e library site. Unfortunately, I was deeply disappointed by page three. I always buy books i
n the hope and expectation of having an enjoyable read, not to criticise. However, this book
 is in urgent need of good editing -- though quite possibly editing alone wouldn't save it.
Examples: a bed squeaks slightly and sharply in the same sentence; a nightgown hangs freely
over her girlish figure and olive colored complexion; coffee aromas huddle; rumbling clouds
huddle (as well as the coffee?); she prepared to sip her coffee beneath the wrath of God...
cuddled within the arms of a strong breeze; (the wrath of God is a breeze?); the Columbian (
stet) coffee aroma danced beneath her sculpted tan nose; the coffee bean fragrance tangoed w
ithin her body; she placed her thick pink lips against the warm cup;... and so on, all by pa
ge three. It is quite possible that the storyline is deeply moving. I'll never know because
I can't bring myself to continue. Sorry."}
{'_id': ObjectId('63892ad42f74734999771138'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 10.95, 'User_id': 'A1E9M6APK30ZAU', 'profileName': 'V. Powell', 're
view/helpfulness': '1/2', 'review/score': 4, 'review/time': 1119571200, 'review/summary': 'H
ere is my opinion', 'review/text': 'I have to admit, I am not one to write reviews on book.
I do however like to read them. Some of the reviews are great, some are cruel, but the revie
ws on this book were absolutlely hysterical. I read this book and after watching the war of
the words I couldn\'t help, but to jump in. I am on the good side of the book. I loved how i
t kept my intersted. On the bad side I have to say, there was a typo featured on the back co
ver and the secret to liking this book is making it through the first chapter. I have to adm
it I was a little curious when I read some of the reviews. They were so mean and false that
it spiked my curiosity. I then like the last review noticed that the "Traveling librarian" w
rote a completely ignorant review on this book. My first thought was wait a minute, that was
 not the book that I read. I read my copy once again and realized her entire review was fabr
icated. Then I knew, it was some kind of game. Maybe it is someone who dislikes the author f
or personal reasons, that has posted more then one awful review, or maybe this is another wh
o is jealous, Who knows? All I know is that I liked the book, and future readers take the ad
vice of the ignorant with a grain of salt.'}
```

5. Find review based on the ID of the Book

```python
def findByBookID(Id):
    all = db.reviews.find({'Id': Id}).limit(10)
    for document in all:
        print(document)

def findByBookIO():
    print("Find reviews based on Book")
    bookID = input("Enter BookID: ") # Ex: 0826414346
    findByBookID(bookID)
```

```
Find reviews based on Book
Enter BookID: 0595344550
{'_id': ObjectId('63892ad42f74734999771137'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 7.0, 'User_id': 'A3Q12RK71N74LB', 'profileName': 'Book Reader', 're
view/helpfulness': '7/11', 'review/score': 1, 'review/time': 1117065600, 'review/summary': '
not good', 'review/text': "I bought this book because I read some glowing praise on an onlin
e library site. Unfortunately, I was deeply disappointed by page three. I always buy books i
n the hope and expectation of having an enjoyable read, not to criticise. However, this book
 is in urgent need of good editing -- though quite possibly editing alone wouldn't save it.
Examples: a bed squeaks slightly and sharply in the same sentence; a nightgown hangs freely
over her girlish figure and olive colored complexion; coffee aromas huddle; rumbling clouds
huddle (as well as the coffee?); she prepared to sip her coffee beneath the wrath of God...
cuddled within the arms of a strong breeze; (the wrath of God is a breeze?); the Columbian (
stet) coffee aroma danced beneath her sculpted tan nose; the coffee bean fragrance tangoed w
ithin her body; she placed her thick pink lips against the warm cup;... and so on, all by pa
ge three. It is quite possible that the storyline is deeply moving. I'll never know because
I can't bring myself to continue. Sorry."}
{'_id': ObjectId('63892ad42f74734999771138'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 10.95, 'User_id': 'A1E9M6APK30ZAU', 'profileName': 'V. Powell', 're
view/helpfulness': '1/2', 'review/score': 4, 'review/time': 1119571200, 'review/summary': 'H
ere is my opinion', 'review/text': 'I have to admit, I am not one to write reviews on book.
I do however like to read them. Some of the reviews are great, some are cruel, but the revie
ws on this book were absolutlely hysterical. I read this book and after watching the war of
the words I couldn\'t help, but to jump in. I am on the good side of the book. I loved how i
t kept my intersted. On the bad side I have to say, there was a typo featured on the back co
ver and the secret to liking this book is making it through the first chapter. I have to adm
it I was a little curious when I read some of the reviews. They were so mean and false that
it spiked my curiosity. I then like the last review noticed that the "Traveling librarian" w
rote a completely ignorant review on this book. My first thought was wait a minute, that was
 not the book that I read. I read my copy once again and realized her entire review was fabr
icated. Then I knew, it was some kind of game. Maybe it is someone who dislikes the author f
or personal reasons, that has posted more then one awful review, or maybe this is another wh
o is jealous, Who knows? All I know is that I liked the book, and future readers take the ad
vice of the ignorant with a grain of salt.'}
{'_id': ObjectId('63892ad42f74734999771139'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 10.95, 'User_id': 'AUR0VA5H0C66C', 'profileName': 'LoveToRead "Actu
ally Read Books"', 'review/helpfulness': '1/2', 'review/score': 1, 'review/time': 1119225600
, 'review/summary': 'Buyer beware', 'review/text': 'This is a self-published book, and if yo
u want to know why--read a few paragraphs! Those 5 star reviews must have been written by Ms
. Haddon\'s family and friends--or perhaps, by herself! I can\'t imagine anyone reading the
whole thing--I spent an evening with the book and a friend and we were in hysterics reading
bits and pieces of it to one another. It is most definitely bad enough to be entered into so
me kind of a "worst book" contest. I can\'t believe Amazon even sells this kind of thing. Ma
ybe I can offer them my 8th grade term paper on "To Kill a Mockingbird"--a book I am quite s
ure Ms. Haddon never heard of. Anyway, unless you are in a mood to send a book to someone as
 a joke---stay far, far away from this one!'}
```

6. Find reviews from a specific UserID:

```python
def findByUser_id(User_id):
    all = db.reviews.find({'User_id': User_id}).limit(10)
    for document in all:
        print(document)

def findByUser_id_IO():
    print("Find reviews based on User_id")
    uid = input("Enter User_id: ") # Ex: A30TK6U7DNS82R
    findByUser_id(uid)
```

```
Find reviews based on User_id
Enter User_id: AUR0VA5H0C66C
{'_id': ObjectId('63892ad42f74734999771139'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 10.95, 'User_id': 'AUR0VA5H0C66C', 'profileName': 'LoveToRead "Actu
ally Read Books"', 'review/helpfulness': '1/2', 'review/score': 1, 'review/time': 1119225600
, 'review/summary': 'Buyer beware', 'review/text': 'This is a self-published book, and if yo
u want to know why--read a few paragraphs! Those 5 star reviews must have been written by Ms
. Haddon\'s family and friends--or perhaps, by herself! I can\'t imagine anyone reading the
whole thing--I spent an evening with the book and a friend and we were in hysterics reading
bits and pieces of it to one another. It is most definitely bad enough to be entered into so
me kind of a "worst book" contest. I can\'t believe Amazon even sells this kind of thing. Ma
ybe I can offer them my 8th grade term paper on "To Kill a Mockingbird"--a book I am quite s
ure Ms. Haddon never heard of. Anyway, unless you are in a mood to send a book to someone as
 a joke---stay far, far away from this one!'}
```

7. Find reviews by profile name:

```python
def findByprofileName(profileName):
    all = db.reviews.find({'profileName': profileName}).limit(10)
    for document in all:
        print(document)

def findByprofileNameIO():
    print("Find reviews based on profile name")
    profile_name = input("Enter profile name: ") # Ex: Kevin Killian
    findByprofileName(profile_name)
```

```
Enter profile name: Andrei T
{'_id': ObjectId('6389342b85b2c7cb4b14d491'),
 'Id': 1234, 'Title': 'The duck', 'Price': 3.
4, 'User_id': 'AG123', 'profileName': 'Andrei
 T', 'review/helpfulness': 'TBD', 'review/sco
re': '4/5', 'review/time': 1669936155.247974,
 'review/summary': 'book about ducks', 'revie
w/text': 'A phenomenal book about ducks'}
{'_id': ObjectId('6389377543d0eea32f2eeaaf'),
 'Id': '1998', 'Title': 'Ducktective', 'Price
': 3.6, 'User_id': 'AH889', 'profileName': 'A
ndrei T', 'review/helpfulness': 'TBD', 'revie
w/score': '5/5', 'review/time': 1669937001.48
0811, 'review/summary': 'Ducks and mystery',
'review/text': 'It was an amazing book about
ducks'}
```

Searched up the review I inserted earlier

8. Find reviews lower than certain price

```
def findBooksUnderPrice(priceMax):
    all = db.reviews.find({ 'Price': {'$lte': priceMax} } ).limit(10)
    for document in all:
        print(document)

def findByPriceIO():
    print("Find reviews lower than price of: ")
    priceMax = input("Enter price: ") # Ex: 10
    findBooksUnderPrice(priceMax)
```

```
Enter price: 5.00
{'_id': ObjectId('63892ad42f74734999771129'), 'Id': '1882931173', 'Title': 'Its Only Art If
Its Well Hung!', 'Price': '0.0', 'User_id': 'AVCGYZL8FQQTD', 'profileName': 'Jim of Oz "jim-
of-oz"', 'review/helpfulness': '7/7', 'review/score': 4, 'review/time': 940636800, 'review/s
ummary': 'Nice collection of Julie Strain images', 'review/text': "This is only for Julie St
rain fans. It's a collection of her photos -- about 80 pages worth with a nice section of pa
intings by Olivia.If you're looking for heavy literary content, this isn't the place to find
 it -- there's only about 2 pages with text and everything else is photos.Bottom line: if yo
u only want one book, the Six Foot One ... is probably a better choice, however, if you like
 Julie like I like Julie, you won't go wrong on this one either."}
{'_id': ObjectId('63892ad42f74734999771165'), 'Id': '0854968350', 'Title': "Muslim Women's C
hoices: Religious Belief and Social Reality (Cross Cultural Perspectives on Women)", 'Price'
: '0.0', 'User_id': 'A2OP3N8OXCQEZB', 'profileName': 'Lyta', 'review/helpfulness': '6/8', 'r
eview/score': 5, 'review/time': 1121385600, 'review/summary': 'Excellent', 'review/text': "T
his is an excellent book. It offers the opinions of a diverse sample of real Muslim Women wi
th very different attitudes. It will not appeal to those who don't want to listen or who wan
t an anti-islamic feminist political tract. The value of this is that you hear the opinions,
 educated and not, of the real people involved.Most of the time, the real voices are ignored
 and what is presented in the name of feminism is non-muslim american female TV personalitie
s yacking endlessly on about the lives and experiences of people they really don't know and
don't want to know.What must be understood above all else is that if you wish to change a so
ciety or a culture, you must at the very least understand how the people in that culture who
 disagree with you think. That is what this book does so well. I may not agree at all with t
he opinion of every woman in the book, but I at least understand how they think after finish
ing it.What many will never understand is that Islamic Society has changed and is still chan
ging. But it can only change by consensus from within. No amount of cultural imperialism fro
m western countries is going to change anything for the better.There is also more than a lit
tle hypocracy. For all the lectures about the veil from American Women, few of them stop to
think about why they are themselves forced into certain choices. A veil isn't great, but its
 a whole lot more confortable than the pantyhose that many Western women must wear effective
ly as a condition of employment. One wonders at a certain point if its the veil itself they
object to or that as constrasted to their money-driven appearance decisons, the veil has a m
eaning beyond the secular."}
```

Price is 0.0 since i put a default value for Books w/out prices

9. Find books by a keyword (ie: Duck)

```
def findBookByKeyword(keyword):
    all = db.reviews.find( { '$text': { '$search': keyword } } ).limit(10)
    #all = db.users.find({"Title" : /.*keyword.*/});.limit(10)
    for document in all:
        print(document)

def findBookByKeywordIO():
    print("Find book review by keyword: ")
    keyword = input("Enter keyword: ") # Ex: duck
    findBookByKeyword(keyword)
```

```
Find book review by keyword:
Enter keyword: duck
{'_id': ObjectId('638951f7e23c3cdc0b50be6e'), 'Id': '0689845065', 'Title': 'Giggle, Giggle,
Quack', 'Price': 12.23, 'User_id': 'AMX0PJKV4PPNJ', 'profileName': 'E. R. Bird "Ramseelbird"
', 'review/helpfulness': '1/1', 'review/score': 4, 'review/time': 1109376000, 'review/summar
y': "Duck's triumphant return", 'review/text': 'Riding on the heels of the ever popular, "Cl
ick Clack Moo", Doreen Cronin, and partner in crime Betsy Lewin, attempted to once more to h
arness the magic that made their earlier creation so beloved. All your favorite characters a
re back. Farmer Brown. His eloquent cows. His pigs. His chickens. And of course, the Duck. D
uck firmly establishes himself to be as quick with the pencil as the cows once were with the
ir typewriters. It is in "Giggle, Giggle, Quack" that he first truly comes into his own.Some
where along the line, Farmer Brown figured out that the craftiest animal in his barnyard was
 that no good sneaky-peek Duck. So when the farmer leaves on vacation, his tells his brother
 in no uncertain terms to "keep an eye on Duck. He\'s trouble". And sure enough, with the di
scovery of a wayward pencil, Duck starts taking farm matters into his own hands/beak. He fig
ures out that Farmer Brown left Bob, his bro, copious notes regarding the running of the far
m. It doesn\'t take long for Duck to deftly swap the farmer\'s notes for his own livestock f
riendly ones. Before you know it, the farm animals are munching pizza, getting lovely spa-li
ke baths, and enjoying "movie night" on Farmer Brown\'s couch. When their absent landlord fi
nally catches on via the telephone, Duck presents his babysitter with the note, "It\'s for y
ou, Bob!". The last image is of a Hawaiian-shirt wearing Farmer Bob pelting it quicktime bac
k to the farm once again to restore some semblance of order."Giggle, Giggle, Quack", doesn\'
t have the snappy ending that "Click Clack Moo" had, admittedly. And the phrase, "giggle, gi
ggle, oink" (or whatever animal noise being used at the time), isn\'t really going to compar
e with the satisfying sounds of "click, clack, moo". Still, there\'s a lot to enjoy here. Th
e animals take a special pleasure in outwitting the dullard humans around them. Kids reading
 the book will enjoy pinpointing the exact moment that Farmer Brown\'s pencil slips from his
 back pocket and onto the ground. And Duck\'s discovery of it is well accompanied by surrept
itious glances exchanged between the cows. For kids who\'ve already grown to love these char
acters, "Giggle Giggle Quack" provides more of the same silly subversive fun. For kids who\'
ve never encountered the series before, they\'ll probably still find something to enjoy here
. Though it\'s not the best written of the Cronin/Lewin combo, it\'s still a fine n\' dandy
way to spend a minute or two\'s worth of bedtime reading.'}
{'_id': ObjectId('638951f7e23c3cdc0b50be52'), 'Id': '0689845065', 'Title': 'Giggle, Giggle,
Quack', 'Price': 12.23, 'User_id': 'Unknown', 'profileName': 'Unknown', 'review/helpfulness'
```

10. Top 10 cheapest books (if it has 10 books); Sorted by price in descending order:

```python
# Cheaapest book reviewed by certain user
def findCheapestBook(User_id):
    pipe = [
        {"$match": {"User_id": User_id}},
        {"$sort": {"price": 1}},
        {"$limit": 10}
    ]
    all = db.reviews.aggregate(pipe)
    for doc in all:
        print(doc)

def cheapBookIO():
    print("Get top 10 cheapest book by reviewer")
    u_id = input("Enter User_id: ")
    findCheapestBook(u_id)
```

```
Get top 10 cheapest book by reviewer
Enter User_id: A123
{'_id': ObjectId('63893f536341bc98c4d90021'), 'Id': '323455', 'Title': 'Disgusting Book', 'P
rice': 7.79, 'User_id': 'A123', 'profileName': 'Andrei T', 'review/helpfulness': 'TBD', 'rev
iew/score': '2/5', 'review/time': 1669939017.5211, 'review/summary': 'PogChamp', 'review/tex
t': 'This is a bad book'}
{'_id': ObjectId('63893f1521e138806d2fb5b9'), 'Id': '3234', 'Title': 'Bad Book', 'Price': 9.
99, 'User_id': 'A123', 'profileName': 'Andrei T', 'review/helpfulness': 'TBD', 'review/score
': '4/5', 'review/time': 1669938957.948229, 'review/summary': 'adwi', 'review/text': 'ADWodw
iiadwowio'}
```

11. Find reviews based on scores (ie: amount of 4 star scores, 5 star scores, etc.)

```
def findAmountOfScoresForBook(Title):
    pipe = [
        {"$match": {"Title": Title}},
        {'$group': {'_id': '$review/score', 'count': {'$sum': 1}}}
    ]
    all = db.reviews.aggregate(pipe)
    for doc in all:
        print(doc)

def findAmountScoresBookIO():
    print("Find amount of scores per book")
    title = input("Input Title: ")
    findAmountOfScoresForBook(title)
```

```
Find amount of scores per book
Input Title: Dr. Seuss: American Icon
{'_id': 4, 'count': 4}
{'_id': 5, 'count': 5}
```

```python
def findByScore(score):
    all = db.reviews.find({'review/score': score}).limit(10)
    for doc in all:
        print(doc)

def findByScoreIO():
    print("Find review based on score")
    score = input("Enter score: ") # 4.6
    try:
        score = int(score)
    except:
        print("Error not int")
        return

    findByScore(score)
```

```
Find review based on score
Enter score: 3
{'_id': ObjectId('63892ad42f74734999771171'), 'Id': '1858683092', 'Title': 'Mensa Number Puz
zles (Mensa Word Games for Kids)', 'Price': '0.0', 'User_id': 'A1AYN4J7T43M11', 'profileName
': '"dirtpile"', 'review/helpfulness': '4/4', 'review/score': 3, 'review/time': 981417600, '
review/summary': 'Made me wish I was Einstein.', 'review/text': "Not much I can say about th
is book... It's full of VVVery difficult number puzzles... at least for me. Okay, so I don't
 get 'A's for my grades at school, but I think that even you might find it a little difficul
t. The way I see it, you will need lots of paper, patience and time, say 30 minutes or so to
 solve even the easiest of these.Why? Because basically, you are asked how many combinations
 can be formed from such and such. Example: How many ways get 31241 by adding prime numbers
only? This is not really what the problems are like in the book, they are far more complicat
ed than that.For me, it's just too much. I can't even do 1 single problem. I can't even look
 up the answers because uses some sort of a legend that I can't understand. But if you have
an above average intelligence or are mathematically gifted, you will probably find some obsc
ure short cut to the problem a la Gauss. For the rest, forget it.Word of warning: these are
purely number problems. I can't tell if logic will help or even plays a part in this book, b
ut the problems are NUMBER problems."}
{'_id': ObjectId('63892ad42f74734999771175'), 'Id': '0792391810', 'Title': 'Vector Quantizat
ion and Signal Compression (The Springer International Series in Engineering and Computer Sc
ience)', 'Price': 76.94, 'User_id': 'A30DX2BO4Y4NLU', 'profileName': 'Moosh', 'review/helpfu
lness': '3/5', 'review/score': 3, 'review/time': 1113609600, 'review/summary': 'Comprehensiv
e but marred by poor printing', 'review/text': 'This book appears to be a "print on demand"
style book which manifests itself, in this case, as a poor quality hardcover. Some of the te
xt is laughably bad, the few images near the end look like they are snapshots from a black a
nd white TV screen, but most bothersome is the "muddy" look of the text. The price seems a b
it steep for such a poor quality print. That said, the actual content is very comprehensive.
 One of the authors (Gray) is co-creator of the now commonplace LBG (Linde Buzo Gray) VQ alg
orithm. Fortunately, the quality of the content shines through the terrible printing.'}
{'_id': ObjectId('63892ad42f7473499977117a'), 'Id': '0974289108', 'Title': 'The Ultimate Gui
de to Law School Admission: Insider Secrets for Getting a "Big Envelope" with Your Acceptanc
e to Law School!', 'Price': 14.95, 'User_id': 'A1KZ0RDJZQSY4O', 'profileName': 'sayock', 're
view/helpfulness': '27/29', 'review/score': 3, 'review/time': 1090368000, 'review/summary':
```

12. Find the Amount of Score for a specific book:

13. Find books within a price range

```
def findPriceBetweenRange(min, max):
    all = db.reviews.find({'Price': {'$gte': min, '$lte': max}}).limit(10)
    for doc in all:
        print(doc)

def priceRangeIO():
    print("Find a book within a price range")
    priceMin = input("Minimum price: ")
    priceMax = input("Maximum price: ")
    try:
        priceMin = float(priceMin)
        priceMax = float(priceMax)
    except:
        print("Error not float")
        return
    findPriceBetweenRange(priceMin, priceMax)
```

```
Find a book within a price range
Minimum price: 3.00
Maximum price: 6.00
{'_id': ObjectId('63892ad42f7473499977127b'), 'Id': 'B000FZEKVA', 'Title': "Communicating wi
th Orcas - The Whales' Perspective", 'Price': 4.3, 'User_id': 'A33LNJBNL8KCU6', 'profileName
': 'Paul Nelson', 'review/helpfulness': '12/13', 'review/score': 4, 'review/time': 104284800
0, 'review/summary': 'Orca Wisdom as told by Mary Getten', 'review/text': 'Whales in the dre
ams of children, whale spirituality, whale sex and whale governance are all things we\'ll no
t likely get from scientific researchers for decades, perhaps a century or two. That is one
reason the new book by Mary Getten, a Naturalist specializing in the wildlife of the San Jua
n Islands and a practicing animal communicator is so brilliant. This is not for folks who th
ink that actual communication with whales (or other critters) is impossible. Those folks are
 probably the same people who get angry at their dogs for not doing what they are told.Mary
herself was once a skeptic, as she relates in "Communicating with Orcas: The Whales\' Perspe
ctive". At her first animal communication workshop, facilitated by Penelope Smith, Mary was
given the task to communicate with Pirouette, Penelope\'s cockatiel:"I knew nothing about bi
rds. Mammals were my passion, and I had nointerest in delicate little avians. Truthfully, I
was a little frightened of them, yet here I was face to face with a cockatiel and it was tim
e for us to talk.I looked into her bright white face, admiring the crest of yellow feathers
on her forehead and the orange dot on her cheek. Drinking in her features, I relaxed and tho
ught, this won\'t be too tough. She seems nice enough. So I began to think about what to say
 to her.This is where it all went downhill. I didn\'t have a clue what to say to a bird. You
 would have though that I\'d spent my entire life locked into a closet. Not a single questio
n appeared. Time ticked by and I knew I had to so something, so I kept repeating: "Hello Pir
ouette. Hello Pirouette. Hello Pirouette."Finally, I stopped for a moment to think of someth
ing else to say, and clear as a bell I heard Pirouette say: "Can\'t you say anything but hel
lo?"Humor, backed with good science and historical tidbits going back to Aristotle and Pliny
 The Elder balance the presentation for those who are a little reticent to believe in this m
ode of communication. (This skepticism persists even in the wake of breakthrough research by
 famed scientist Rupert Sheldrake.) Mary\'s writing is clear and accessible and the stories
of whale sex are quite funny. I mean laugh-out-loud, drool on the PAGE funny.Yet, the skepti
cism that Mary had when first learning to communicate with animals is the book\'s biggest we
akness, in my view. I was fascinated by the notion of Ruffles, the male elder of the Souther
n Puget Sound resident community\'s J Pod, visiting the dreams of schoolchildren who had bee
n on whale-watching cruises. The notion that our species, the lowly Homo sapiens, is at the
beginning of a new wave of consciousness that will see remarkable innovations in bodies and
senses, or that of the whale councils, the ability to live in past lives (though whales don\
't call them that) and their description of the universal life force as "The Prime Cause," a
re all subjects that fascinate me, but the book only teases with these concepts. Perhaps a s
equel is called for.However, these are pretty far-out concepts for the average reader and wh
en it comes down to it, Mary has covered the basics in a thorough manner with a wicked wit.
If you have the least bit of interest in Orcas, you will be able to read this book in a week
end.May we all be able to live in the moment like these whales have for thousands of years.
Their intelligence is much needed in these days when our allegedly more intelligent species,
 can\'t get beyond such insanity as war and environmental degradation.Paul Nelson'}
{'_id': ObjectId('63892ad42f7473499977127c'), 'Id': 'B000FZEKVA', 'Title': "Communicating wi
th Orcas - The Whales' Perspective", 'Price': 4.3, 'User_id': 'A2LI0ZEY24NV0H', 'profileName
': 'K. Jones "kjay12"', 'review/helpfulness': '6/6', 'review/score': 5, 'review/time': 10411
20000, 'review/summary': 'Open your mind and your heart!', 'review/text': "This book is one
of the most profound and wonderful books I have ever had the pleasure of reading! It's a bea
utiful true story that tells of two animal communicators, Mary and her good friend Raphaela,
 and their conversations with Granny. Granny is a 92 (est.) year-old orca who lives in the P
acific Northwest with her family, J Pod. During Mary and Raphaela's time conversing with Gra
nny, they question her on many aspects of orca life from finding food to orca spirituality.
Granny relays a very important message to the human race in this book. A message that MUST N
OT be overlooked! This book should be read and emabraced by all. Granny's words certainly op
ened my eyes and made me see things in a different light. Open your mind and your heart. Rea
d this book!"}
```

14. Most helpful reviews from a specific user (sorted in descending order):

```
def findProfileNameHelpful(profile_name):
    pipe = [
        {"$match": {"profileName": profile_name}},
        {"$sort": {"review/helpfulness": -1}},
        {"$limit": 10}
    ]
    all = db.reviews.aggregate(pipe)
    for doc in all:
        print(doc)

def profileHelpfulIO():
    print("Top 10 most helpful reviews by user")
    profileName = input("Enter profileName of Reviewer: ")
    findProfileNameHelpful(profileName)
```

```
Enter profileName of Reviewer: J. serratos
{'_id': ObjectId('63892d6f2f7473499980de7e'), 'Id': 'B000724VK6', 'Title': 'Harriet Tubman:
The Moses of her people', 'Price': '0.0', 'User_id': 'A29I6N5F8P386Y', 'profileName': 'J. se
rratos', 'review/helpfulness': '1/2', 'review/score': 4, 'review/time': 1110067200, 'review/
summary': 'The moses of her people', 'review/text': 'The book made me realize that enyone ca
n do anything they wish. I wish it would have gone a little deeper into the life of harriet
tubman, however I thought it was still a good book.'}
{'_id': ObjectId('63892ad42f7473499977114a'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 10.95, 'User_id': 'A29I6N5F8P386Y', 'profileName': 'J. serratos', '
review/helpfulness': '0/0', 'review/score': 5, 'review/time': 1111017600, 'review/summary':
"The best book I've read in years !!", 'review/text': 'This is a wonderful book that will ke
ep you guessing. It is Rare to find a book that will make you laugh and cry at the same time
 and still leave you wondering what will happen next. What was even more amazing about it, i
s the fact that it is written in such a poetic way. This is a rare find and I look forward t
o the next book written by Veronica Haddon.'}
```

15. Find highest scores in descending order (high to low) by a user:

```python
def findIdRating(User_id):
    pipe = [
        {"$match": {"User_id": User_id}},
        {"$sort": {"review/score": -1}},
        {"$limit": 10}
    ]
    all = db.reviews.aggregate(pipe)
    for doc in all:
        print(doc)

def IdScoreIO():
    print("Top 10 biggest scored books by user")
    User_id = input("Enter id of Reviewer: ")
    findIdRating(User_id)
```

```
Top 10 biggest scored books by user
Enter id of Reviewer: A29I6N5F8P386Y
{'_id': ObjectId('63892ad42f7473499977114a'), 'Id': '0595344550', 'Title': 'Whispers of the
Wicked Saints', 'Price': 10.95, 'User_id': 'A29I6N5F8P386Y', 'profileName': 'J. serratos', '
review/helpfulness': '0/0', 'review/score': 5, 'review/time': 1111017600, 'review/summary':
"The best book I've read in years !!", 'review/text': 'This is a wonderful book that will ke
ep you guessing. It is Rare to find a book that will make you laugh and cry at the same time
 and still leave you wondering what will happen next. What was even more amazing about it, i
s the fact that it is written in such a poetic way. This is a rare find and I look forward t
o the next book written by Veronica Haddon.'}
{'_id': ObjectId('63892d6f2f7473499980de7e'), 'Id': 'B000724VK6', 'Title': 'Harriet Tubman:
The Moses of her people', 'Price': '0.0', 'User_id': 'A29I6N5F8P386Y', 'profileName': 'J. se
rratos', 'review/helpfulness': '1/2', 'review/score': 4, 'review/time': 1110067200, 'review/
summary': 'The moses of her people', 'review/text': 'The book made me realize that enyone ca
n do anything they wish. I wish it would have gone a little deeper into the life of harriet
tubman, however I thought it was still a good book.'}
```