

## **CS157C Final Project Report**

**Team Name:** Mongothebee

**Title of Project:** Yelp Business Reviews Search

### **Project description:**

Yelp Business Reviews Search is an application that lets you find a business and information about it. You can find a business by what type of service it gives such as food, massages, etc. If you are looking for a specific location for a business, you can find ones nearby you. You can also provide ratings for a business location. Once a business is found, you can find and read reviews of the business so you can decide if you want to visit that location.

You can appreciate NoSQL with our application because certain attributes like hours can have varying fields between different restaurants. For example, a restaurant may have fields like “monday”:”11:00-12:30” while another may not have a field for “monday” at all if it is closed on a monday.

### **Project Successes:**

This project was successfully completed due to our commitment as a team. We were able to schedule and conduct meetings to work on the project and we all were able to attend each meeting. During these meetings, we spent time trying to understand the problems we were facing and also spent some time outside of the meetings trying to fix the issues we had. Each of us provided meaningful contributions for suggestions and ideas when designing our application. Because we all stayed on the same page, we were able to develop the application that is working as we have intended.

### **Unexpected Events:**

When setting up the replica set for our application, we did not expect multiple issues. First, we did not expect how time consuming it would be to remake the replica set each time we started the lab so we had to work around it by allocating our schedules. Another thing is we did not expect to run out of storage when importing the data into the database which caused us to have to redo the replica set from scratch. Multiple times we ran into issues of the data import failing due to one of the shards failing during the imports. To combat these issues, we decided to research a little bit to set up our instances in aws so that their ip address will not expire using Amazon’s Elastic IPs, and therefore, the data would not disappear once the data imported. From there, we would then import the data.

### **Lessons Learned:**

It is safe to say that we learned a lot about how frustrating it is to set up replica sets each time and how time consuming it is so we would recommend allocating time before starting. We would also recommend prewriting the functions ahead of time like we did because we had

something to work on while we waited for data to import. We also recommend setting up the instances so that they do not expire, however that does consume more credits than usual so be aware of when you plan to finish the project because you may run out of credits before the due date if you start early and not continue.

### **DataSet:**

Dataset: <https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset>

Dataset consists of data from yelp which is a total of 9.29 GB which is changed in our wrangling method.

From the dataset, we decided to use two json files: business.json and reviews.json. From there, we decided to wrangle the data. For business.json, it consists of business information such as hours, location, the type of business, etc. For reviews.json, it consists of business' reviews information, which contains ratings and the actual review text. There was also a third type of data, user profiles, that we removed due to not being used for our application.

### **Description for Dataset Wrangling**

- a. business.json
  - Field values of each document are as such: business\_id, name, address, city, state, postal\_code, latitude, longitude, stars, review\_count, categories, hours
  - Fixed different apostrophe types to all be the same for consistent querying
  - Remove entire attributes embedded document
- b. reviews.json
  - Field values of each document are as such: review\_id, business\_id, stars, text, date
  - Removed tags: useful, funny, cool since there were not filled in any documents and were not useful for search
  - Remove users\_id because we will not be using the users data

### **Schema Description:**

Our database is split up into two collections: business and reviews.

- Business Collection: Contains all documents of businesses keyed by a unique value "business\_id".
  - We chose to use a business' postal code for the shard key and query through the shards using that key as the index.
- Reviews Collection: Contains all documents containing reviews that are connected to the businesses by "business\_id" and are keyed by a unique value "review\_id".
  - We chose to use the business\_id that is associated with the review for the shard key and query through the shards using that key as the index.

### NoSQL Configuration:

In order to host our application's database, we use AWS's EC2 instances as virtual machines in a sharded system. In total, we used 5 t2 large instances running Ubuntu version 20.04 containing 8 gb memory each. We also allocated 32 gb of storage space for each instance.

We have provided a table. The table describes our configuration set up for our aws instances, what ports were used for each replica set, and the corresponding ip addresses used for the instances. In total, we have 13 nodes: 3 for config, 3 for each shard for a total of 9, and 1 for our mongos instance. For the config and shard instances, we have a replica set set up in each with 1 primary and 2 secondaries.

Node Number	Instance	Port Number	Service	Public Ipv4 address	Public Ipv4 DNS
Node 0	config	28000	Config Server Secondary	3.229.13.209	ec2-3-229-13-209.compute-1.amazonaws.com
Node 1	config	28001	Config Server Primary	3.229.13.209	ec2-3-229-13-209.compute-1.amazonaws.com
Node 2	config	28002	Config Server Secondary	3.229.13.209	ec2-3-229-13-209.compute-1.amazonaws.com
Node 3	Shard 1	28010	Shard Server Secondary	34.236.56.135	ec2-34-236-56-135.compute-1.amazonaws.com
Node 4	Shard 1	28011	Shard Server Secondary	34.236.56.135	ec2-34-236-56-135.compute-1.amazonaws.com
Node 5	Shard 1	28012	Shard Server Primary	34.236.56.135	ec2-34-236-56-135.compute-1.amazonaws.com
Node 6	Shard 2	28020	Shard Server	52.201.31.17	ec2-52-201-3


			Secondary	0	1-170.compute-1.amazonaws.com
Node 7	Shard 2	28021	Shard Server Primary	52.201.31.170	ec2-52-201-31-170.compute-1.amazonaws.com
Node 8	Shard 2	28022	Shard Server Secondary	52.201.31.170	ec2-52-201-31-170.compute-1.amazonaws.com
Node 9	Shard 3	28030	Shard Server Primary	52.45.126.27	ec2-52-45-126-27.compute-1.amazonaws.com
Node 10	Shard 3	28031	Shard Server Secondary	52.45.126.27	ec2-52-45-126-27.compute-1.amazonaws.com
Node 11	Shard 3	28032	Shard Server Secondary	52.45.126.27	ec2-52-45-126-27.compute-1.amazonaws.com
Node 12	mongos	27017	Shard controller	54.80.176.84	ec2-54-80-176-84.compute-1.amazonaws.com

## Config Server rs.status():

 mongosh mongodb://3.229.13.209:28000/?directConnection=true

```
members: [
  {
    _id: 0,
    name: '3.229.13.209:28000',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6313,
    optime: { ts: Timestamp({ t: 1670012977, i: 2 }), t: Long("5") },
    optimeDate: ISODate("2022-12-02T20:29:37.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:29:37.754Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:29:37.754Z"),
    syncSourceHost: '3.229.13.209:28001',
    syncSourceId: 1,
    infoMessage: '',
    configVersion: 1,
    configTerm: 5,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '3.229.13.209:28001',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 6309,
    optime: { ts: Timestamp({ t: 1670012977, i: 1 }), t: Long("5") },
    optimeDurable: { ts: Timestamp({ t: 1670012976, i: 2 }), t: Long("5") },
    optimeDate: ISODate("2022-12-02T20:29:37.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:29:36.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:29:37.748Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:29:36.709Z"),
    lastHeartbeat: ISODate("2022-12-02T20:29:37.751Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:29:37.153Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1670006678, i: 1 }),
    electionDate: ISODate("2022-12-02T18:44:38.000Z"),
    configVersion: 1,
    configTerm: 5
  },
  {
    _id: 2,
    name: '3.229.13.209:28002',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6306,
    optime: { ts: Timestamp({ t: 1670012977, i: 1 }), t: Long("5") },
    optimeDurable: { ts: Timestamp({ t: 1670012976, i: 2 }), t: Long("5") },
    optimeDate: ISODate("2022-12-02T20:29:37.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:29:36.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:29:37.748Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:29:36.709Z"),
    lastHeartbeat: ISODate("2022-12-02T20:29:37.752Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:29:37.751Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '3.229.13.209:28001',
    syncSourceId: 1,
    infoMessage: '',
    configVersion: 1,
    configTerm: 5
  }
],
```

Shard Server 1 rs.status():

 mongosh mongodb://34.236.56.135:28010/?directConnection=true

```
members: [
  {
    _id: 0,
    name: '34.236.56.135:28010',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6412,
    optime: { ts: Timestamp({ t: 1670013091, i: 1 }), t: Long("6") },
    optimeDate: ISODate("2022-12-02T20:31:31.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:31:31.287Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:31:31.287Z"),
    syncSourceHost: '34.236.56.135:28012',
    syncSourceId: 2,
    infoMessage: '',
    configVersion: 1,
    configTerm: 6,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '34.236.56.135:28011',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6407,
    optime: { ts: Timestamp({ t: 1670013091, i: 1 }), t: Long("6") },
    optimeDurable: { ts: Timestamp({ t: 1670013091, i: 1 }), t: Long("6") },
    optimeDate: ISODate("2022-12-02T20:31:31.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:31:31.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:31:31.287Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:31:31.287Z"),
    lastHeartbeat: ISODate("2022-12-02T20:31:34.743Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:31:34.121Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '34.236.56.135:28010',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 6
  },
  {
    _id: 2,
    name: '34.236.56.135:28012',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 6404,
    optime: { ts: Timestamp({ t: 1670013091, i: 1 }), t: Long("6") },
    optimeDurable: { ts: Timestamp({ t: 1670013091, i: 1 }), t: Long("6") },
    optimeDate: ISODate("2022-12-02T20:31:31.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:31:31.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:31:31.287Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:31:31.287Z"),
    lastHeartbeat: ISODate("2022-12-02T20:31:34.743Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:31:34.121Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1670006701, i: 1 }),
    electionDate: ISODate("2022-12-02T18:45:01.000Z"),
    configVersion: 1,
    configTerm: 6
  }
],
```

## Shard Server 2 rs.status():

 mongosh mongodb://52.201.31.170:28020/?directConnection=true

```
members: [
  {
    _id: 0,
    name: '52.201.31.170:28020',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6432,
    optime: { ts: Timestamp({ t: 1670013126, i: 1 }), t: Long("7") },
    optimeDate: ISODate("2022-12-02T20:32:06.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:32:06.080Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:32:06.080Z"),
    syncSourceHost: '52.201.31.170:28022',
    syncSourceId: 2,
    infoMessage: '',
    configVersion: 1,
    configTerm: 7,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '52.201.31.170:28021',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 6425,
    optime: { ts: Timestamp({ t: 1670013126, i: 1 }), t: Long("7") },
    optimeDurable: { ts: Timestamp({ t: 1670013126, i: 1 }), t: Long("7") },
    optimeDate: ISODate("2022-12-02T20:32:06.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:32:06.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:32:06.080Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:32:06.080Z"),
    lastHeartbeat: ISODate("2022-12-02T20:32:11.247Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:32:10.185Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1670006715, i: 1 }),
    electionDate: ISODate("2022-12-02T18:45:15.000Z"),
    configVersion: 1,
    configTerm: 7
  },
  {
    _id: 2,
    name: '52.201.31.170:28022',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6422,
    optime: { ts: Timestamp({ t: 1670013126, i: 1 }), t: Long("7") },
    optimeDurable: { ts: Timestamp({ t: 1670013126, i: 1 }), t: Long("7") },
    optimeDate: ISODate("2022-12-02T20:32:06.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:32:06.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:32:06.080Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:32:06.080Z"),
    lastHeartbeat: ISODate("2022-12-02T20:32:11.247Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:32:10.924Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '52.201.31.170:28021',
    syncSourceId: 1,
    infoMessage: '',
    configVersion: 1,
    configTerm: 7
  }
],
```


## Shard Server 3 rs.status():

 mongosh mongodb://52.45.126.27:28030/?directConnection=true

```
members: [
  {
    _id: 0,
    name: '52.45.126.27:28030',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 6447,
    optime: { ts: Timestamp({ t: 1670013164, i: 1 }), t: Long("6") },
    optimeDate: ISODate("2022-12-02T20:32:44.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:32:44.788Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:32:44.788Z"),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1670006734, i: 1 }),
    electionDate: ISODate("2022-12-02T18:45:34.000Z"),
    configVersion: 1,
    configTerm: 6,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '52.45.126.27:28031',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6442,
    optime: { ts: Timestamp({ t: 1670013164, i: 1 }), t: Long("6") },
    optimeDurable: { ts: Timestamp({ t: 1670013164, i: 1 }), t: Long("6") },
    optimeDate: ISODate("2022-12-02T20:32:44.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:32:44.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:32:44.788Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:32:44.788Z"),
    lastHeartbeat: ISODate("2022-12-02T20:32:45.977Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:32:46.813Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '52.45.126.27:28030',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 6
  },
  {
    _id: 2,
    name: '52.45.126.27:28032',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6438,
    optime: { ts: Timestamp({ t: 1670013164, i: 1 }), t: Long("6") },
    optimeDurable: { ts: Timestamp({ t: 1670013164, i: 1 }), t: Long("6") },
    optimeDate: ISODate("2022-12-02T20:32:44.000Z"),
    optimeDurableDate: ISODate("2022-12-02T20:32:44.000Z"),
    lastAppliedWallTime: ISODate("2022-12-02T20:32:44.788Z"),
    lastDurableWallTime: ISODate("2022-12-02T20:32:44.788Z"),
    lastHeartbeat: ISODate("2022-12-02T20:32:45.977Z"),
    lastHeartbeatRecv: ISODate("2022-12-02T20:32:45.976Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '52.45.126.27:28030',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 6
  }
]
```



Mongos sh.status():

 mongosh mongodb://ec2-54-80-176-84.compute-1.amazonaws.com:27017/?directConnection=true

```
[direct: mongos] test> sh.status()
shardingVersion
{
  _id: 1,
  minCompatibleVersion: 5,
  currentVersion: 6,
  clusterId: ObjectId("63884c32f307df4ae771be71")
}
---
shards
[
  {
    _id: 'shard2_repl',
    host: 'shard2_repl/52.201.31.170:28020,52.201.31.170:28021,52.201.31.170:28022',
    state: 1,
    topologyTime: Timestamp({ t: 1669878874, i: 6 })
  },
  {
    _id: 'shard3_repl',
    host: 'shard3_repl/52.45.126.27:28030,52.45.126.27:28031,52.45.126.27:28032',
    state: 1,
    topologyTime: Timestamp({ t: 1669878897, i: 5 })
  },
  {
    _id: 'shard_repl',
    host: 'shard_repl/34.236.56.135:28010,34.236.56.135:28011,34.236.56.135:28012',
    state: 1,
    topologyTime: Timestamp({ t: 1669878869, i: 1 })
  }
]
---
active mongoses
[ { '6.0.3': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
---
```

mongosh mongodb://ec2-54-80-176-84.compute-1.amazonaws.com:27017/?directConnection=true

```
databases
[
  {
    database: { _id: 'config', primary: 'config', partitioned: true },
    collections: {
      'config.system.sessions': {
        shardKey: { _id: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'shard2_rep1', nChunks: 1024 } ],
        chunks: [
          'too many chunks to print, use verbose if you want to force print'
        ],
        tags: []
      }
    }
  },
  {
    database: {
      _id: 'test',
      primary: 'shard2_rep1',
      partitioned: false,
      version: {
        uuid: new UUID("c588afd4-7a9e-4ad5-8ab7-4c8f8c1f1001"),
        timestamp: Timestamp({ t: 1669878922, i: 1 }),
        lastMod: 1
      }
    },
    collections: {}
  },
  {
    database: {
      _id: 'yelpDB',
      primary: 'shard3_rep1',
      partitioned: false,
      version: {
        uuid: new UUID("b42b70b1-a3d3-4573-a181-cf13fa208143"),
        timestamp: Timestamp({ t: 1669878912, i: 1 }),
        lastMod: 1
      }
    },
    collections: {}
  }
]
```

mongosh mongodb://ec2-54-80-176-84.compute-1.amazonaws.com:27017/?directConnection=true

```
collections: {
  yelpDB.business: {
    shardKey: { postal_code: 'hashed' },
    unique: false,
    balancing: true,
    chunkMetadata: [
      { shard: 'shard2_rep1', nChunks: 2 },
      { shard: 'shard3_rep1', nChunks: 2 },
      { shard: 'shard_rep1', nChunks: 2 }
    ],
    chunks: [
      { min: { postal_code: MinKey() }, max: { postal_code: Long("-6148914691236517204") }, 'on shard': 'shard2_rep1', 'last modified': Timestamp({ t: 1, i: 0 }) },
      { min: { postal_code: Long("-6148914691236517204") }, max: { postal_code: Long("-3074457345618258602") }, 'on shard': 'shard2_rep1', 'last modified': Timestamp({ t: 1, i: 1 }) },
      { min: { postal_code: Long("-3074457345618258602") }, max: { postal_code: Long("0") }, 'on shard': 'shard3_rep1', 'last modified': Timestamp({ t: 1, i: 2 }) },
      { min: { postal_code: Long("0") }, max: { postal_code: Long("3074457345618258602") }, 'on shard': 'shard3_rep1', 'last modified': Timestamp({ t: 1, i: 3 }) },
      { min: { postal_code: Long("3074457345618258602") }, max: { postal_code: Long("6148914691236517204") }, 'on shard': 'shard_rep1', 'last modified': Timestamp({ t: 1, i: 4 }) },
      { min: { postal_code: Long("6148914691236517204") }, max: { postal_code: MaxKey() }, 'on shard': 'shard_rep1', 'last modified': Timestamp({ t: 1, i: 5 }) }
    ],
    tags: []
  },
  yelpDB.review: {
    shardKey: { business_id: 'hashed' },
    unique: false,
    balancing: true,
    chunkMetadata: [
      { shard: 'shard2_rep1', nChunks: 2 },
      { shard: 'shard3_rep1', nChunks: 2 },
      { shard: 'shard_rep1', nChunks: 2 }
    ],
    chunks: [
      { min: { business_id: MinKey() }, max: { business_id: Long("-6148914691236517204") }, 'on shard': 'shard2_rep1', 'last modified': Timestamp({ t: 1, i: 0 }) },
      { min: { business_id: Long("-6148914691236517204") }, max: { business_id: Long("-3074457345618258602") }, 'on shard': 'shard2_rep1', 'last modified': Timestamp({ t: 1, i: 1 }) },
      { min: { business_id: Long("-3074457345618258602") }, max: { business_id: Long("0") }, 'on shard': 'shard3_rep1', 'last modified': Timestamp({ t: 1, i: 2 }) },
      { min: { business_id: Long("0") }, max: { business_id: Long("3074457345618258602") }, 'on shard': 'shard3_rep1', 'last modified': Timestamp({ t: 1, i: 3 }) },
      { min: { business_id: Long("3074457345618258602") }, max: { business_id: Long("6148914691236517204") }, 'on shard': 'shard_rep1', 'last modified': Timestamp({ t: 1, i: 4 }) },
      { min: { business_id: Long("6148914691236517204") }, max: { business_id: MaxKey() }, 'on shard': 'shard_rep1', 'last modified': Timestamp({ t: 1, i: 5 }) }
    ],
    tags: []
  }
}
[direct: mongos] test>
```

## Shard Key Rationale

For each collection, we developed a shard key. For our business collection, we chose to use a field that was unique enough so that the data would be evenly distributed among the shards. As such, we chose to use the postal code for a business since each postal code covered a small amount of businesses, and this would also mean that businesses near each other would be shared on the same shard.

For our reviews collection, choosing a shard key was more difficult, but we settled on using the id of a business since reviews would belong to a business. Relying on a business id would allow the shards to be evenly distributed across the shards and would allow reviews belonging to the same businesses to be on the same shards.

## NoSQL Driver Description

The driver that we chose to use was pymongo to connect our python application to the MongoDB database. In order to achieve this connection, we need to use the pymongo library and can access it with an import statement.

```
import pymongo
from pprint import pprint
from bson import ObjectId
from pymongo import MongoClient
from datetime import date
import re
```

From the pymongo library, we specifically want to use the MongoClient class, which allows us to connect directly to our Mongos node in AWS using its public IPv4 DNS address. We can also then access the databases and collections directly from the MongoClient object.

```
client = MongoClient('ec2-54-80-176-84.compute-1.amazonaws.com', 27017)
yelpDB = client.yelpDB
business = yelpDB.business
reviews = yelpDB.reviews
```

In order to run the application we use `python -i .\mongothebee.py`, and this runs the program in python's interactive mode, allowing users to call each function whenever they want to.

```
PS C:\Users\gametest\Desktop> python -i .\mongothebee.py
>>> □
```

## 15 Functions:

### 1. Find all businesses that are within the given postal code

- db.business.find( {"postal\_code": "xxxxx"}, {business\_id: 1, name: 1})
- Pymongo: collection.find()

```
13 #function1 find businesses by given zip code
14 def findByZip(zip):
15     for x in business.find({"postal_code": zip}, {"name": 1, "business_id": 1, "postal_code": 1}).limit(10):
16         pprint(x)
17 #findByZip("93101")
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

gpy\launcher' '58060' '--' 'c:\Users\gametest\Desktop\mongothebee.py'

PS C:\Users\gametest\Desktop> python -i .\mongothebee.py

>>> findByZip("93101")

```
{ '_id': ObjectId('63894376224b57f83f0bc07c'),
  'business_id': '5ZU9c8V2GuREDN5KgyHFJw',
  'name': 'Santa Barbara Shellfish Company',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc009'),
  'business_id': 'noByYntDLQAra9ccqxdFdw',
  'name': 'H&M',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bbfef'),
  'business_id': 'Pns2l4eNsf08kk83dixA6A',
  'name': 'Abby Rappoport, LAC, CNQ',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc067'),
  'business_id': 'bVjnx_J1hHzob10Do5FkqQ',
  'name': 'Tinkle Belle Diaper Service',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc099'),
  'business_id': 'QZU7TcrztBb3tXaPbVckXg',
  'name': '805 Ink',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc247'),
  'business_id': 'VEbHYioBfoPiIOWntE_DBA',
  'name': 'Tienda Ho',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc256'),
  'business_id': '82sumumWp1MEq04QhiS9DQ',
  'name': 'Surreal Virtual Reality Studio',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc044'),
  'business_id': 'IDtLPgUrqrppqSLdfMhZQ',
  'name': 'Helena Avenue Bakery',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc2a2'),
  'business_id': '-ky_HDP7IMwGI-kBIZVU4A',
  'name': 'Dune Coffee Roasters - Anacapa',
  'postal_code': '93101' }
{ '_id': ObjectId('63894376224b57f83f0bc31c'),
  'business_id': '6jomGWEI4ryImrWpGuUQQQ',
  'name': 'Santa Barbara Athletic Club',
  'postal_code': '93101' }
```

c.

## 2. Find all business that are in a certain city

- db.business.find( {"city": "xxxxx"}, {business\_id: 1, name: 1})
- Pymongo: collection.find()

```
C: > Users > gametest > Desktop > mongothebee.py > ...
18
19 #function2 Find businesses in a given city
20 def findByCity(city):
21     for x in business.find({"city": city}, {"name": 1, "business_id": 1, "city": 1}).limit(10):
22         pprint(x)
23 #findByCity("Nashville")
24
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
PS C:\Users\gametest\Desktop> python -i .\mongothebee.py
>>> findByCity("Nashville")
{'_id': ObjectId('63894376224b57f83f0bc036'),
 'business_id': 'B2qqjCl_BNNnXvWwMfpFiLA',
 'city': 'Nashville',
 'name': 'Pottery Barn Kids'}
{'_id': ObjectId('63894376224b57f83f0bc090'),
 'business_id': 'G4lRmWdHdvHZAhtlSYyBw',
 'city': 'Nashville',
 'name': "Wendy's"}
{'_id': ObjectId('63894376224b57f83f0bc01e'),
 'business_id': 'lk9IwjZXqUMqqOhM774DtQ',
 'city': 'Nashville',
 'name': 'Caviar & Bananas'}
{'_id': ObjectId('63894376224b57f83f0bc278'),
 'business_id': 'oSp8B87eQ8droCGiWUERZA',
 'city': 'Nashville',
 'name': 'The Moose Music Row'}
{'_id': ObjectId('63894376224b57f83f0bc28c'),
 'business_id': 'W5tIz4s1lyRW_fOXHoGVzQ',
 'city': 'Nashville',
 'name': 'barre3 Nashville - The Gulch'}
{'_id': ObjectId('63894376224b57f83f0bc00a'),
 'business_id': 'tMkwHmWfUEXrC9ZduonpIg',
 'city': 'Nashville',
 'name': 'The Green Pheasant'}
{'_id': ObjectId('63894376224b57f83f0bc089'),
 'business_id': 'yv6kSKxDdZTsvHL25K-U2g',
 'city': 'Nashville',
 'name': 'Nash Painting'}
{'_id': ObjectId('63894376224b57f83f0bc0ff'),
 'business_id': 'rYOrSXy2I4Bcx87dhg29uA',
 'city': 'Nashville',
 'name': 'Restoration Hardware'}
{'_id': ObjectId('63894376224b57f83f0bc315'),
 'business_id': 'NrL5xcCR9Zhy8kh5mHwVwW',
 'city': 'Nashville',
 'name': 'Paige Simmons Salon'}
{'_id': ObjectId('63894376224b57f83f0bc2b5'),
 'business_id': 'ZuOtTcUKmthgvhKo7B_-sg',
 'city': 'Nashville',
 'name': 'Music City Pub Crawl'}
```

c.

### 3. Find businesses that serve a specific category

- db.business.find( {"categories" : /.\*xxxxx.\*/}, {business\_id: 1, name: 1} )
- Pymongo: collection.find()

```
25 #function3 find businesses by a given genre/category
26 def findByGenre(genre):
27     for x in business.find({}, {"business_id": 1, "name": 1, "categories": 1}):
28         if genre in x["categories"]:
29             pprint(x)
30 #findByGenre("Italian")
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

```
PS C:\Users\gametest\Desktop> & 'C:\Users\gametest\AppData\Local\Programs\Python\Python310\python.exe' gpy\launcher '58598' '--' 'c:\Users\gametest\Desktop\mongothebee.py'
```

```
PS C:\Users\gametest\Desktop> python -i .\mongothebee.py
```

```
>>> findByGenre("Italian")
```

```
{'_id': ObjectId('63894376224b57f83f0bc072'),
 'business_id': 'FRYkg_JvsWU9xIXZsEZcVA',
 'categories': 'Cocktail Bars, Italian, Nightlife, Seafood, Bars, Restaurants',
 'name': 'Altamura'}
{'_id': ObjectId('63894376224b57f83f0bc0d9'),
 'business_id': 'Y6heWJJ9AmEL58fZwgi9YQ',
 'categories': 'Sports Bars, Bars, Nightlife, Italian, Restaurants, Pizza',
 'name': 'Rosati's Pizza'}
{'_id': ObjectId('63894376224b57f83f0bc146'),
 'business_id': '1P_mGUY1PyPq7_ZabrzpBw',
 'categories': 'Italian, Restaurants, Steakhouses',
 'name': 'Carmine's Steakhouse'}
{'_id': ObjectId('63894376224b57f83f0bc2f2'),
 'business_id': 'sWCCxY1-9B1FGLSveQvnHg',
 'categories': 'Restaurants, Wine Bars, Italian, Food, Sandwiches, Nightlife, '
 'Breweries, Bars, Pizza',
 'name': 'CD Roma Restaurant'}
{'_id': ObjectId('63894376224b57f83f0bc029'),
 'business_id': 'uI9XODGY_2_ieTE6xJ0myw',
 'categories': 'Restaurants, American (New), Italian',
 'name': 'Roman Forum'}
{'_id': ObjectId('63894376224b57f83f0bc0e4'),
 'business_id': 'uDQgmudYDKiB6n4vwbEeDg',
 'categories': 'Restaurants, Italian, Nightlife, Bars, Cocktail Bars, '
 'Tapas/Small Plates',
 'name': 'Chrysalis'}
```

c.

4. Find all businesses that have the same name/part of the same business chain

- db.business.find( {"name": "xxxxx"}, {business\_id: 1, name: 1} )
- Pymongo: collection.find()

```
C: > Users > gametest > Desktop > mongothebee.py > findByGenre
32 #function4 find businesses by a given name
33 def findByName(name):
34     for x in business.find({"name": name}, {"name": 1, "business_id": 1, "address": 1}).limit(10):
35         pprint(x)
36 #findByName("McDonald's")
37
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> findByName("McDonald's")
{'_id': ObjectId('63894376224b57f83f0bc1f1'),
 'address': '1119 N Tucker',
 'business_id': '9qxMhGwjGsuPUoQB3Y74Wg',
 'name': 'McDonald's'}
{'_id': ObjectId('63894376224b57f83f0bc275'),
 'address': '11110 Causeway Blvd',
 'business_id': 'A1lAqN3S0aBLoo6NqVcy4Q',
 'name': 'McDonald's'}
{'_id': ObjectId('63894376224b57f83f0bc0b0'),
 'address': '1919 S Jefferson',
 'business_id': 'yM8LlTIInbQH4FwWC97lz6w',
 'name': 'McDonald's'}
{'_id': ObjectId('63894376224b57f83f0bc283'),
 'address': '501 Franklin Mill Cir',
 'business_id': 'kLSEk3qXY1KdtFjxHJ9pww',
 'name': 'McDonald's'}
{'_id': ObjectId('63894376224b57f83f0bc2f8'),
 'address': '8907 E 116th St',
 'business_id': 'wcpL6sD6jKsB00tbdK2KEA',
 'name': 'McDonald's'}
{'_id': ObjectId('63894377224b57f83f0bc77c'),
 'address': '3165 N Hwy 67',
 'business_id': 'aqubLw9Iw03svL_uDXNJ8A',
 'name': 'McDonald's'}
{'_id': ObjectId('63894378224b57f83f0bc904'),
 'address': '906 Carlyle Ave',
 'business_id': '3bmrXhqVpB5XCg-4j6y39A',
 'name': 'McDonald's'}
{'_id': ObjectId('63894378224b57f83f0bce42'),
 'address': '321 W Trenton Ave',
 'business_id': 'y0pfoNQOyrb98aEIIdH6cnA',
 'name': 'McDonald's'}
{'_id': ObjectId('63894378224b57f83f0bceca'),
 'address': '2702 Philadelphia Pike',
 'business_id': '59fWFpyDg018TEH-Hh6jdw',
 'name': 'McDonald's'}
{'_id': ObjectId('63894378224b57f83f0bcf46'),
 'address': '133 S 69th St',
 'business_id': 'e0eKgRKbs-iAsIE0mHTrYg',
 'name': 'McDonald's'}
```

c.

5. Find all business which have more than 3 stars, meaning they are highly recommended to visit

- db.business.find( {"stars": {\$gt : 3}}, {business\_id: 1, name: 1} )
- Pymongo: collection.find()

```
38 #function5 find business by number of stars
39 def findByStars(stars):
40     for x in business.find({"stars": {"$gte": stars}}, {"name": 1, "business_id": 1, "stars": 1}).sort("name").limit(10):
41         pprint(x)
42 #findByStars(5)
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> findByStars(5)
{'_id': ObjectId('6389438224b57f83f0c44b6'),
 'business_id': 't7vVRlyF9_lxuL1auqVchg',
 'name': " Joe's Throwback Barber Shop",
 'stars': 5}
{'_id': ObjectId('63894379224b57f83f0bd5b6'),
 'business_id': 'BWo8s2skxQGZUgVook2HZQ',
 'name': '"No Macarena" DJ & Custom Music Service',
 'stars': 5}
{'_id': ObjectId('638943a8224b57f83f0df6d2'),
 'business_id': 'VG9Uz8YL8_-DdAY6nB_m6A',
 'name': '$155 Flat Rate Hauling Trash Removal',
 'stars': 5}
{'_id': ObjectId('6389439d224b57f83f0d7c30'),
 'business_id': '3zEedfqHtCrQEUsmPsDNsA',
 'name': '$5 Fresh Burger Stop',
 'stars': 5}
{'_id': ObjectId('638943a4224b57f83f0dc5b2'),
 'business_id': 'EGvXWSOIILCrFL68TDIA69Q',
 'name': '09 Pub',
 'stars': 5}
{'_id': ObjectId('638943a6224b57f83f0dddb2'),
 'business_id': '05xx0kketcfczIINbe016g',
 'name': '1 Hour Jewelry and Watch Repair',
 'stars': 5}
{'_id': ObjectId('6389439b224b57f83f0d5f8a'),
 'business_id': 'FOu4RijHs3ro-Bkok6EfYA',
 'name': '1 Solution AV',
 'stars': 5}
{'_id': ObjectId('6389437f224b57f83f0c23df'),
 'business_id': 'WmLZanQpFG4qpFT7j3RXbg',
 'name': '1 Stop Mini Market',
 'stars': 5}
{'_id': ObjectId('63894389224b57f83f0c934b'),
 'business_id': 'Ac_q18b7jUom4CybvKjMUQ',
 'name': '1 Stop Sushi Express',
 'stars': 5}
{'_id': ObjectId('6389437f224b57f83f0c250a'),
 'business_id': 'BZAVgg5ybeX_a4w-Fw0mbA',
 'name': '1-800-GOT-JUNK? Central Coast',
 'stars': 5}
```

C.



## 6. Finding the address of a specific business

- db.business.find({"business\_id": "xxxxx"}, {business\_id: 1, name: 1, address: 1, city: 1, state: 1, postal\_code: 1});
- Pymongo: collection.find()

```
44 #function6 find the address of a specific business
45 def findAddress(businessID):
46     for x in business.find({"business_id": businessID}, {"name": 1, "address": 1, "business_id": 1}):
47         print(x["name"] + " address: " + x["address"] + ", id: " + x["business_id"])
48 #findAddress("CAIIPqKY0pS0yQ8fKGIDPg")
49
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> findAddress("CAIIPqKY0pS0yQ8fKGIDPg")
Åse Yoga Studio and Tea Room address: 4054 Ridge Ave, id: CAIIPqKY0pS0yQ8fKGIDPg
```

c.

## 7. Making a business entry

- db.business.insertOne(business\_id: xxxx, name: xxxx, address: xxxx, city: xxx, state: xxx, postal\_code:xxxx, latitude:xxxx, longitude:xxx, stars:xxx, review\_count:xxxx, categories:xxx, hours:xxx);
- Pymongo: collection.find(), collection.insert\_one()

```
50 #function7 Make a new business
51 def makeBusiness(id, name, address, city, state, postal_code, lat, long, stars, categories, hours):
52     business = {"business_id": id, "name": name, "address": address, "city": city, "state": state, "postal_code": postal_code, "latitude": lat, "longitude": long,
53               "stars": stars, "review_count": 0, "categories": categories, "hours": hours}
54     business.insert_one(business)
55     x = business.find_one({"business_id": id})
56     pprint(x)
57 #makeBusiness("test", "TEST BUSINESS", "1 ROAD 1", "San Jose", "CA", "11111", "1", "2", "5", "testing, Italian", {"monday": "1:00-2:00"})
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> makeBusiness("test", "TEST BUSINESS", "1 ROAD 1", "San Jose", "CA", "11111", "1", "2", "-1", "testing, Italian", {"monday": "1:00-2:00"})
{'_id': ObjectId('63894dacbab2f6e6f4668258'),
 'address': '1 ROAD 1',
 'business_id': 'test',
 'categories': 'testing, Italian',
 'city': 'San Jose',
 'hours': {'monday': '1:00-2:00'},
 'latitude': '1',
 'longitude': '2',
 'name': 'TEST BUSINESS',
 'postal_code': '11111',
 'review_count': 0,
 'stars': '-1',
 'state': 'CA'}
```

c.

**8. All of a certain business' average rating in stars**

- a. `db.business.find({"name": "xxxxx"}, {business_id: 1} );`
- b. Pymongo: `collection.find()`

```
59 #function8 all of a certain restaurant's average rating in stars
60 def avgReviews(businessName):
61     count = 0
62     stars = 0
63     for x in business.find({"name": businessName}, {"stars": 1}):
64         count = count + 1
65         stars = stars + x["stars"]
66     avg = stars/count
67     print(businessName + " average stars: " + str(avg))
68 #avgReviews("McDonald's")
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> avgReviews("McDonald's")
McDonald's average stars: 1.8634423897581793
```

c.

**9. Check what time a certain business is open**

- a. `db.business.find({"business_id": "xxxxx"}, {business_id: 1, name: 1, hours: 1});`
- b. Pymongo: `collection.find()`

```
70 #function9 find a business's hours
71 def businessHours(businessID):
72     for x in business.find({"business_id": businessID}, {"name": 1, "hours": 1, "business_id": 1}):
73         pprint(x)
74 #businessHours("CAIIPqKY0pS0yQ8fKG1DPg")
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> businessHours("CAIIPqKY0pS0yQ8fKG1DPg")
{'_id': ObjectId('6389438e224b57f83f0ccb56'),
 'business_id': 'CAIIPqKY0pS0yQ8fKG1DPg',
 'hours': {'Friday': '18:0-19:30',
           'Saturday': '8:0-22:0',
           'Sunday': '9:0-10:30',
           'Wednesday': '18:0-19:30'},
 'name': 'Åse Yoga Studio and Tea Room'}
```

c.

## 10. Find businesses containing a certain range of reviews

- db.business.find( {"review\_count": {\$gte : x, \$lte : x}, {business\_id: 1, name: 1} )
- Pymongo: collection.find()

```
76 #function10 find businesses with certain number / range of reviews
77 def reviewRange(low, high):
78     for x in business.find({"review_count": {"$gte" : low, "$lte" : high}}, {"name": 1, "business_id": 1, "review_count": 1}).limit(10):
79         pprint(x)
80 #reviewRange(300, 500)
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> reviewRange(300, 500)
{'_id': ObjectId('63894376224b57f83f0bc056'),
 'business_id': 'TLZ3-eDPLHuzfsw04ad6Ug',
 'name': 'Mahony's Po-Boys & Seafood',
 'review_count': 382}
{'_id': ObjectId('63894376224b57f83f0bc2c6'),
 'business_id': '18eWJFJbXyR9j_5xfcRLYA',
 'name': 'Siam Elephant',
 'review_count': 460}
{'_id': ObjectId('63894376224b57f83f0bc046'),
 'business_id': 'VNjyv0gfor2g8lbnUpTnKg',
 'name': 'Copper Vine',
 'review_count': 350}
{'_id': ObjectId('63894377224b57f83f0bc454'),
 'business_id': 'MuKoTR56s6eIHET2wUkgJA',
 'name': 'Grace Meat + Three',
 'review_count': 470}
{'_id': ObjectId('63894377224b57f83f0bc606'),
 'business_id': 'bXf6TpgHOLiwyMNBYA00CQ',
 'name': 'The Little Lamb',
 'review_count': 307}
{'_id': ObjectId('63894377224b57f83f0bc610'),
 'business_id': 'Y3ZC017N1_I_Ms1JmsmwZA',
 'name': 'Taqueria Pico De Gallo',
 'review_count': 397}
{'_id': ObjectId('63894378224b57f83f0bc82a'),
 'business_id': '2Q1R20hBbAQ581vK_r7NHA',
 'name': 'Ooka Restaurant',
 'review_count': 394}
{'_id': ObjectId('63894378224b57f83f0bc88d'),
 'business_id': 'D_sIoGDRaLJv0DAIc46MMA',
 'name': 'Ruby Slipper Cafe',
 'review_count': 347}
{'_id': ObjectId('63894377224b57f83f0bc652'),
 'business_id': 'ARgidfLSXPHj7uhK-Y7hwA',
 'name': 'Kampai Sushi Bar',
 'review_count': 328}
{'_id': ObjectId('63894378224b57f83f0bc92c'),
 'business_id': 'hkh5iyDeGLNhu-JUQFRE1Q',
 'name': 'Royal Sonesta New Orleans',
 'review_count': 492}
```

C.

## 11. Find all reviews for a specific business

- `db.business.find({"name": "xxxxx"}, {business_id: 1} );`
- `db.reviews.find( {"business_id": "xxxxx"}, {text: 1});`
- Pymongo: `collection.find()`

```
82 #function11 find all reviews of a business
83 def allReviews(businessName):
84     for x in business.find({"name": businessName}).limit(10):
85         for y in reviews.find({"business_id": x["business_id"]}, {"text": 1}).limit(1):
86             pprint(y)
87 #allReviews("McDonald's")
88
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> allReviews("McDonald's")
{'_id': ObjectId('63892de7224b57f83fa14094'),
 'text': "This McDonald's is as terrible as they come. They have no care, mess "
        'up every order, take at least 10mins to answer you in drive thru at '
        'slow points in the day. Stay away smdh!'}
{'_id': ObjectId('63892df7224b57f83fa2282e'),
 'text': "I have lived in this area for four years. I've been to this location "
        'four times. Customer service lacks, rude cashiers and drive thru '
        'people. The food also lacks in quality: taste and preparation- not '
        'visually aesthetic...even the yogurt cones aren't made with "
        'love...drive a little further and go to the Mcds on 301 in front of '
        '"Progressive...Now that's service"}
{'_id': ObjectId('63892de9224b57f83fa159ac'),
 'text': "If you can't come to work with pride and energy and enthusiasm then "
        'don't work at a McDonald's where people are in a hurry to get "
        'somewhere else.....so disappointed in the behavior of humans!'}
{'_id': ObjectId('63892dea224b57f83fa171bd'),
 'text': "I ordered the meal deal with his six nuggets and fries and BBQ sauce "
        'and a large sweet. Receive my order and I touch my fries and they '
        'was cold. Is be young guy in the window to your have any hot fries '
        'so I gave him my the cold fries and he hollered back to the person '
        'that was cooking. They said yes we have hot fries. So he went back '
        'to get them and when he was walking to the window he just hand me '
        'the hot fries in this hands. He should have put them in a bag or '
        'wrapped a napkin around them so I got another order of fries.'}
{'_id': ObjectId('63892dfe224b57f83fa2949c'),
 'text': "Only came here so my three year old could burn off some energy. The "
        'children's play center was disgusting-food and trash was all over "
        'the place.'}
{'_id': ObjectId('63892e15224b57f83fa3d0ee'),
 'text': "One of the worst McDonalds around. Employees do not care. They give "
        'you incomplete orders. Sometimes don't even speak a word to you. "
        'It's horrible"}
{'_id': ObjectId('63892e20224b57f83fa47c02'),
 'text': "Meh I know it's McDonald 's, but of all the ones this is my least "
        'fav. Their drive thru is slow and they frequently mess up the '
        'orders. Their burgers always seem really salty. They got a new '
        'manager so I think things are improving. It's great they have a play "
        'place and redbox though!'}
{'_id': ObjectId('63892de5224b57f83fa11fbc'),
 'text': "I've eaten here and gotten food via the drive-thru many times, and "
        'the food has always been fine. It's standard McDonald's food, so I "
        'won't other with a description. I happen to love their fries and "
        'like their cheeseburgers and a couple of the breakfast sandwiches. \n'
        '\n'
```

d.

## 12. Find all reviews that are more than 3 stars for a certain business location

- `db.business.find({"business_id": "xxxxx", "stars": {"$gt": 3}});`
- Pymongo: `collection.find()`

```
C: > Users > gametest > Desktop > mongothebee.py > ...
98 #function12 find businesses with good ratings (stars greater than 3)
99 def goodRatings(businessID):
100     for x in reviews.find({"business_id": businessID, "stars": {"$gt": 3}}, {"business_id": 1, "stars": 1, "text": 1}):
101         pprint(x)
102 #goodRatings("CAIIPqKY0pS0yQ8fkG1DPg")
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> goodRatings("CAIIPqKY0pS0yQ8fkG1DPg")
{'_id': ObjectId('63893161224b57f83fd31255'),
 'business_id': 'CAIIPqKY0pS0yQ8fkG1DPg',
 'stars': 5,
 'text': 'My bridal party hosted my shower at Ase and it exceeded '
        'expectations. The owner and host were so professional and welcoming. '
        'The food was healthy and delicious. The space was perfect for the '
        '"small tea party I'd dreamed of. The tea party sets they have were so "
        'cute and everyone raved about the black mango tea that was served! '
        'They even honored me with a gift that was so full of life as I head '
        'into this new season of my life. I look forward to taking some yoga '
        'classes here in the future. Great place for parties and events.')}
{'_id': ObjectId('63893174224b57f83fd42a9d'),
 'business_id': 'CAIIPqKY0pS0yQ8fkG1DPg',
 'stars': 5,
 'text': 'Great experience! Very welcoming and calming atmosphere. Enjoy the '
        'practice every time I attend!!!'}
{'_id': ObjectId('63893182224b57f83fd4f1bf'),
 'business_id': 'CAIIPqKY0pS0yQ8fkG1DPg',
 'stars': 5,
 'text': 'Every single lesson leaves you feeling relaxed and refreshed. I also '
        'enjoy learning about various ways to keep my mind, body and spirit '
        '"whole. Ase Yoga is more than stretching, it's complete spiritual "
        'rejuvenation. I definitely recommend this studio!'}
{'_id': ObjectId('63893192224b57f83fd5db9c'),
 'business_id': 'CAIIPqKY0pS0yQ8fkG1DPg',
 'stars': 5,
 'text': 'I absolutely love Asé Yoga! The classes are intimate and typically '
        'have 3-5 people. I've been several times over the years and there is '
        'always a welcoming energy in spite of my inconsistencies. Hunter is '
        'super friendly; and Dr. Alston is a well of knowledge and does an '
        'excellent job of articulating how various asanas impact the body. If '
        'you want yoga classes that really center your breath and mindfulness '
        '(as opposed to solely a workout), this is the space for you.')}
{'_id': ObjectId('63893193224b57f83fd5efde'),
 'business_id': 'CAIIPqKY0pS0yQ8fkG1DPg',
 'stars': 5,
 'text': "If you're looking for a yoga class that offers not just relaxation "
        'and strength building, but a class that offers accommodation, Ase '
        'Yoga is it! As a student at Ase Yoga for a little more than a year '
        '"now, it's my opinion that not only does Dr. Alston offer practical, "
        'historical and applicable information, but she provides an fun, '
        'nurturing environment that is inclusive, that meets you where you '
        'are, physically. Whether you're wheelchair-bound, whether you're "
        'dealing with some degree of obesity, whether you're between 40 to 60 "
        '(or older), OR COMPLETELY NEW TO YOGA, Dr. Alston can, has and does '
```

C.

### 13. Insert new review for a business

- db.reviews.insertOne({review\_id: xxxxx, business\_id: xxxxx, stars: 4, text: "xxxxx", date: "xxxxx"});
- Pymongo: collection.insert\_one()

```
112 #function13 create a review for a given business
113 def writeReview(reviewID, businessID, stars, text):
114     review = {"review_id": reviewID, "business_id": businessID, "stars": stars, "text": text, "date": str(date.today())}
115     reviews.insert_one(review)
116     getReview(reviewID)
117 #writeReview("TESTREVIEWID2", "k6ti2dkJD_6xGzxjQQ_FnA", 5, "test review")
118
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> writeReview("TESTREVIEWID2", "k6ti2dkJD_6xGzxjQQ_FnA", 5, "test review")
{'_id': ObjectId('638a5ef5d5f36cd9f1e1f170'),
 'business_id': 'k6ti2dkJD_6xGzxjQQ_FnA',
 'date': '2022-12-02',
 'review_id': 'TESTREVIEWID2',
 'stars': 5,
 'text': 'test review'}
```

c.

### 14. Change the text of a review

- db.reviews.update({"review\_id": xxxxx}, {"\$set: {"text": "xxxxx"}});
- Pymongo: collections.find\_one(), collection.update\_one()

```
89 #function14 change text of review
90 def changeReview(reviewID, newText):
91     x = reviews.find_one({"review_id": reviewID}, {"business_id": 1})
92     query = {"review_id": reviewID, "business_id": x["business_id"]}
93     newValue = {"$set": {"text": newText}}
94     reviews.update_one(query, newValue, upsert=True)
95     getReview(reviewID)
96 #changeReview("TESTREVIEWID2", "changed")
97
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> changeReview("TESTREVIEWID2", "changed")
{'_id': ObjectId('638a5ef5d5f36cd9f1e1f170'),
 'business_id': 'k6ti2dkJD_6xGzxjQQ_FnA',
 'date': '2022-12-02',
 'review_id': 'TESTREVIEWID2',
 'stars': 5,
 'text': 'changed'}
```

c.

### 15. Delete a review for a business

- a. `db.reviews.deleteOne( {review_id: "xxxxx"})`;
- b. Pymongo: `collection.find_one()`, `collection.delete_one()`;

```
104 #function15 delete a review for a given business
105 def deleteReview(reviewID):
106     x = reviews.find_one({"review_id": reviewID}, {"business_id": 1})
107     query = {"review_id": reviewID, "business_id": x["business_id"]}
108     reviews.delete_one(query)
109     getReview(reviewID)
110 #deleteReview("TESTREVIEWID2")
111
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
>>> deleteReview("TESTREVIEWID2")
None
```

- c.