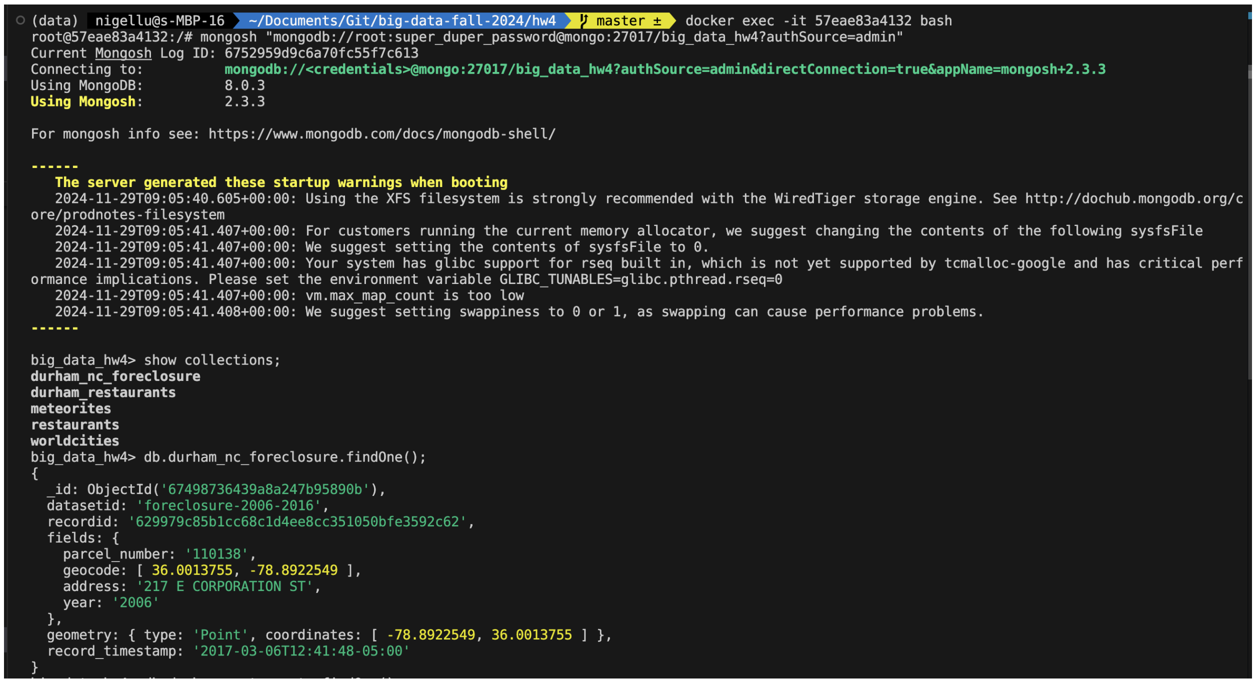
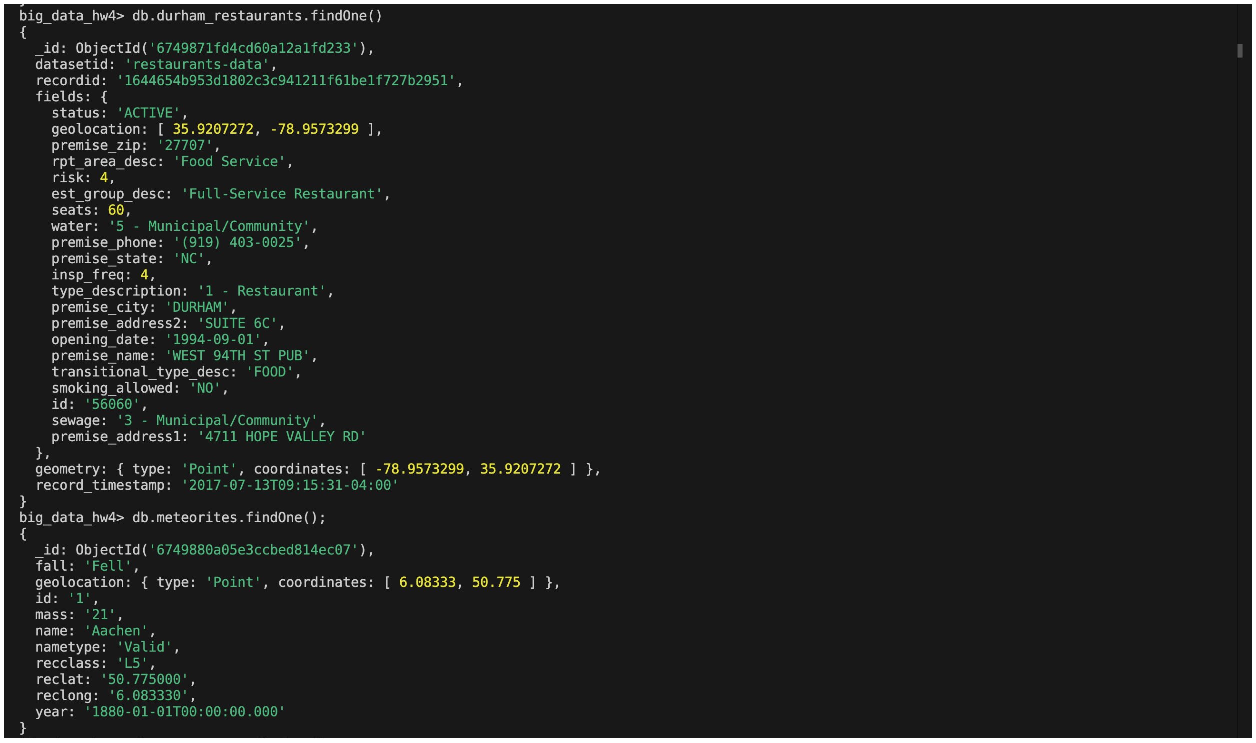
HW4

Tags	CS-GY 6513 Big Data

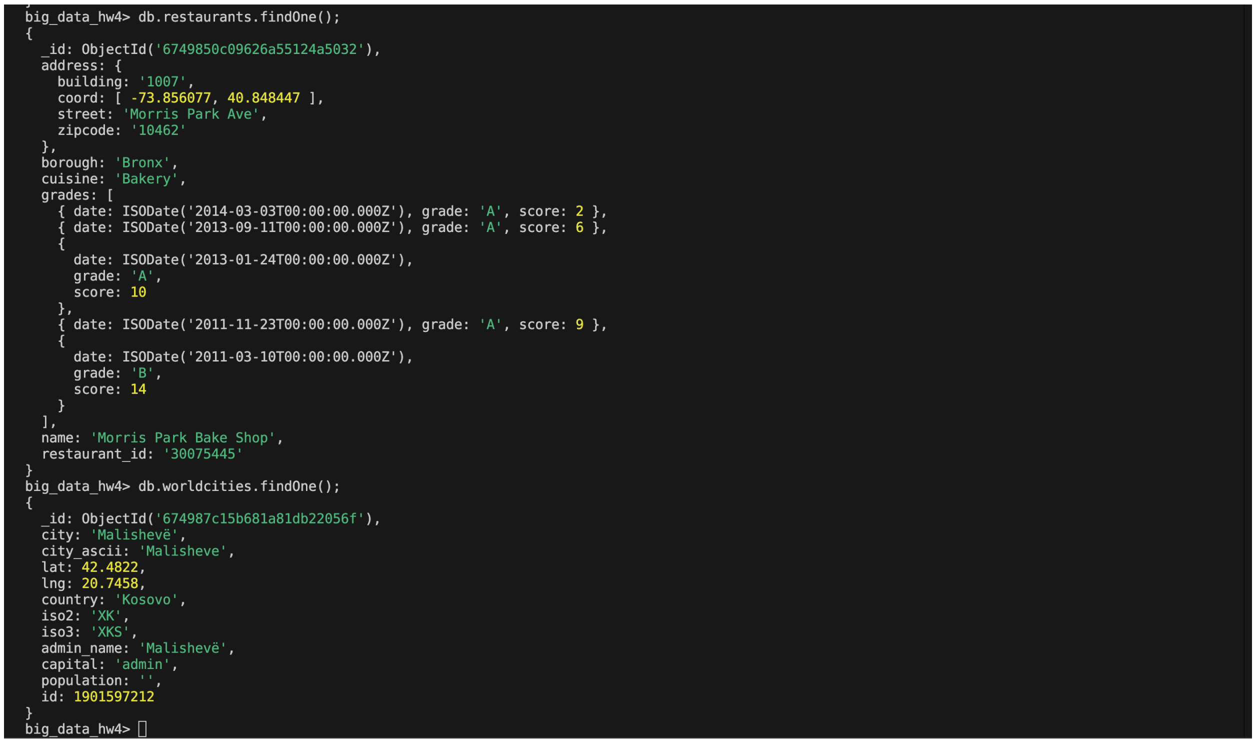
Proof of me successfully loaded the data



```
(data)  nigellu@s-MBP-16  ~/Documents/Git/big-data-fall-2024/hw4  master ±  docker exec -it 57eae83a4132 bash
root@57eae83a4132:/# mongosh "mongodb://root:super_duper_password@mongo:27017/big_data_hw4?authSource=admin"
Current Mongosh Log ID: 6752959d9c6a70fc55f7c613
Connecting to:          mongodb://<credentials>@mongo:27017/big_data_hw4?authSource=admin&directConnection=true&appName=mongosh+2.3.3
Using MongoDB:          8.0.3
Using Mongosh:          2.3.3

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-11-29T09:05:40.605+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/c
ore/prodnotes-filesystem
   2024-11-29T09:05:41.407+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
   2024-11-29T09:05:41.407+00:00: We suggest setting the contents of sysfsFile to 0.
   2024-11-29T09:05:41.407+00:00: Your system has glibc support for rseq built in, which is not yet supported by tcmalloc-google and has critical perf
ormance implications. Please set the environment variable GLIBC_TUNABLES=glibc.pthread.rseq=0
   2024-11-29T09:05:41.407+00:00: vm.max_map_count is too low
   2024-11-29T09:05:41.408+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
------

big_data_hw4> show collections;
durham_nc_foreclosure
durham_restaurants
meteorites
restaurants
worldcities
big_data_hw4> db.durham_nc_foreclosure.findOne();
{
  _id: ObjectId('67498736439a8a247b95890b'),
  datasetid: 'foreclosure-2006-2016',
  recordid: '629979c85b1cc68c1d4ee8cc351050bfe3592c62',
  fields: {
    parcel_number: '110138',
    geocode: [ 36.0013755, -78.8922549 ],
    address: '217 E CORPORATION ST',
    year: '2006'
  },
  geometry: { type: 'Point', coordinates: [ -78.8922549, 36.0013755 ] },
  record_timestamp: '2017-03-06T12:41:48-05:00'
}
```



```
big_data_hw4> db.durham_restaurants.findOne()
{
  _id: ObjectId('6749871fd4cd60a12a1fd233'),
  datasetid: 'restaurants-data',
  recordid: '1644654b953d1802c3c941211f61be1f727b2951',
  fields: {
    status: 'ACTIVE',
    geolocation: [ 35.9207272, -78.9573299 ],
    premise_zip: '27707',
    rpt_area_desc: 'Food Service',
    risk: 4,
    est_group_desc: 'Full-Service Restaurant',
    seats: 60,
    water: '5 - Municipal/Community',
    premise_phone: '(919) 403-0025',
    premise_state: 'NC',
    insp_freq: 4,
    type_description: '1 - Restaurant',
    premise_city: 'DURHAM',
    premise_address2: 'SUITE 6C',
    opening_date: '1994-09-01',
    premise_name: 'WEST 94TH ST PUB',
    transitional_type_desc: 'FOOD',
    smoking_allowed: 'NO',
    id: '56060',
    sewage: '3 - Municipal/Community',
    premise_address1: '4711 HOPE VALLEY RD'
  },
  geometry: { type: 'Point', coordinates: [ -78.9573299, 35.9207272 ] },
  record_timestamp: '2017-07-13T09:15:31-04:00'
}
big_data_hw4> db.meteorites.findOne();
{
  _id: ObjectId('6749880a05e3ccbed814ec07'),
  fall: 'Fell',
  geolocation: { type: 'Point', coordinates: [ 6.08333, 50.775 ] },
  id: '1',
  mass: '21',
  name: 'Aachen',
  nametype: 'Valid',
  recclass: 'L5',
  reclat: '50.775000',
  reclong: '6.083330',
  year: '1880-01-01T00:00:00.000'
}
```



```
big_data_hw4> db.restaurants.findOne();
{
  _id: ObjectId('6749850c09626a55124a5032'),
  address: {
    building: '1007',
    coord: [ -73.856077, 40.848447 ],
    street: 'Morris Park Ave',
    zipcode: '10462'
  },
  borough: 'Bronx',
  cuisine: 'Bakery',
  grades: [
    { date: ISODate('2014-03-03T00:00:00.000Z'), grade: 'A', score: 2 },
    { date: ISODate('2013-09-11T00:00:00.000Z'), grade: 'A', score: 6 },
    {
      date: ISODate('2013-01-24T00:00:00.000Z'),
      grade: 'A',
      score: 10
    },
    { date: ISODate('2011-11-23T00:00:00.000Z'), grade: 'A', score: 9 },
    {
      date: ISODate('2011-03-10T00:00:00.000Z'),
      grade: 'B',
      score: 14
    }
  ],
  name: 'Morris Park Bake Shop',
  restaurant_id: '30075445'
}
big_data_hw4> db.worldcities.findOne();
{
  _id: ObjectId('674987c15b681a81db22056f'),
  city: 'Malishevë',
  city_ascii: 'Malisheve',
  lat: 42.4822,
  lng: 20.7458,
  country: 'Kosovo',
  iso2: 'XK',
  iso3: 'XKS',
  admin_name: 'Malishevë',
  capital: 'admin',
  population: '',
  id: 1901597212
}
big_data_hw4> 
```

Below is my code for querying the results using `mongosh`. The results of the query are saved in `xl3139-hw4.json` with keys named as `<questionNumber>.<partNumber>`. For instance, part 11 of Q1 will be saved under a key `Q1.11`

```
/** @format */

// region: CONSTANTS
const DB_USER = "root";
const DB_PASSWORD = "super_duper_password";
const DB_NAME = "big_data_hw4";
const RESTAURANT_TABLE = "restaurants";
const DURHAM_RESTAURANTS_TABLE = "durham_restaurants";
const DURHAM_FORECLOSURE_TABLE = "durham_nc_foreclosure";
const METEORITES_TABLE = "meteorites";
const WORLD_CITIES_TABLE = "worldcities";
// endregion: CONSTANTS

// region: Preliminary
// * get the database intance
const db = new Mongo(`mongodb://${DB_USER}:${DB_PASSWORD}@mongo
// endregion: Preliminary

// region: Q0
/**
 * I used MongoDB's utility command line tool called `mongoimpo
 * To load normal JSON data, I do `mongoimport --uri "mongodb:/
 * To load newline-delimited JSON data, I do `mongoimport --uri
 * To load CSV data, I do `mongoimport --uri "mongodb://root:su
 *
 * Proof of data loaded:
 * After running "show collections":
 * big_data_hw4> show collections;
 * durham_nc_foreclosure
 * durham_restaurants
 * meteorites
 * restaurants
 * worldcities
 */
// endregion: Q0
// region: Q1
print("Q1\n------\n");
const restaurantCollection = db[RESTAURANT_TABLE];

// * 1. Count the number of documents in the restaurants collect
const restaurantCount = restaurantCollection.countDocuments({})
print("Q1, part 1");
print(`There are ${restaurantCount} documents in the ${RESTAURAN
print("------\n");

// * 2. Display all the documents in the collection
print("Q1, part 2");
restaurantCollection.find({}).forEach(printjson);
print("------\n");

// * 3. Display: restaurant_id, name, borough and cuisine for al
print("Q1, part 3");
restaurantCollection.find({}, { restaurant_id: 1, name: 1, borou
print("------\n");

// * 4. Display: restaurant_id, name, borough and cuisine, but e
print("Q1, part 4");
restaurantCollection.find({}, { restaurant_id: 1, name: 1, borou
print("------\n");

// * 5. Display: restaurant_id, name, borough and zip code, excl
print("Q1, part 5");
restaurantCollection
  .find({}, { restaurant_id: 1, name: 1, borough: 1, "address.zi
  .forEach(printjson);
print("------\n");

// * 6. Display all the restaurants in the Bronx
print("Q1, part 6");
restaurantCollection.find({ borough: "Bronx" }).forEach(printjso
print("------\n");

// * 7. Display the first 5 restaurants in the Bronx
print("Q1, part 7");
restaurantCollection.find({ borough: "Bronx" }).limit(5).forEac
print("------\n");

// * 8. Display the second 5 restaurants in the Bronx (skip the
print("Q1, part 8");
restaurantCollection.find({ borough: "Bronx" }).skip(5).limit(5
print("------\n");

// * 9. Find the restaurants with any score more than 85
print("Q1, part 9");
restaurantCollection.find({ "grades.score": { $gt: 85 } });
print("------\n");

// * 10. Find the restaurants that achieved score, more than 80
print("Q1, part 10");
restaurantCollection.find({ "grades.score": { $gt: 80, $lt: 100
print("------\n");

// * 11. Find the restaurants which locate in longitude value le
print("Q1, part 11");
restaurantCollection.find({ "address.coord.0": { $lt: -95.75416
print("------\n");

// * 12. Find the restaurants that do not prepare any cuisine o
// * and their grade score more than 70 and longitude less than
print("Q1, part 12");
restaurantCollection.find({
  $and: [
    { cuisine: { $ne: "American " } },
    { "grades.score": { $gt: 70 } },
    { "address.coord.0": { $lt: -65.754168 } },
  ],
});
print("------\n");

// * 13. Find the restaurants which do not prepare any cuisine o
// * and achieved a score more than 70
// * and located in the longitude less than -65.754168
// * (without using $and operator).
print("Q1, part 13");
restaurantCollection.find({
  cuisine: { $ne: "American " },
  "grades.score": { $gt: 70 },
  "address.coord.0": { $lt: -65.754168 },
}); // * here since we are operating on different fields, Mongo
print("------\n");

// * 14. Find the restaurants which do not prepare any cuisine o
// * and achieved a grade point 'A'
// * and not in the borough of Brooklyn
// * sorted by cuisine in descending order.
print("Q1, part 14");
db.restaurants
  .find({
    cuisine: { $ne: "American " },
    "grades.grade": "A",
    borough: { $ne: "Brooklyn" },
  })
  .sort({ cuisine: -1 });
print("------\n");

// * 15. Find the restaurant Id, name, borough, and cuisine for
// * which contain 'Wil' as the first three letters of its name
print("Q1, part 15");
restaurantCollection.find(
  {
    name: /^Wil/,
  },
  {
    restaurant_id: 1,
    name: 1,
    borough: 1,
    cuisine: 1,
    _id: 0,
  },
);
print("------\n");

// * 16. Find the restaurant Id, name, borough, and cuisine for
// * which contain 'ces' as the last three letters of its name.
print("Q1, part 16");
restaurantCollection.find({ name: /ces$/ }, { restaurant_id: 1,
print("------\n");

// * 17. Find the restaurant Id, name, borough, and cuisine for
// * which contain 'Reg' as three letters somewhere in its name
print("Q1, part 17");
restaurantCollection.find({ name: /Reg/ }, { restaurant_id: 1,
print("------\n");

// * 18. Find the restaurants which belong to the borough Bronx
// * and prepare either American or Chinese dishes.
print("Q1, part 18");
restaurantCollection.find({
  borough: "Bronx",
  $or: [{ cuisine: "American " }, { cuisine: "Chinese" }],
});
print("------\n");

// * 19. Find the restaurant Id, name, borough, and cuisine for
// * which belong to the boroughs of Staten Island or Queens or
print("Q1, part 19");
restaurantCollection.find(
  { borough: { $in: ["Staten Island", "Queens", "Bronx", "Brook
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0 },
);
print("------\n");

// * 20. Find the restaurant Id, name, borough, and cuisine for
// * which are not belonging to the borough Staten Island or Que
print("Q1, part 20");
restaurantCollection.find(
  { borough: { $nin: ["Staten Island", "Queens", "Bronx", "Brool
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0 },
);
print("------\n");

// * 21. Find the restaurant Id, name, borough, and cuisine for
// * which achieved a score below 10.
print("Q1, part 21");
restaurantCollection.find(
  { "grades.score": { $lt: 10 } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0 },
);
print("------\n");

// * 22. Find the restaurant Id, name, borough, and cuisine for
// * which prepared dishes except 'American' and 'Chinese'
// * or whose name begins with the letter 'Wil'.
print("Q1, part 22");
restaurantCollection.find(
  {
    $or: [{ cuisine: { $nin: ["American ", "Chinese"] } }, { nar
  },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0 },
);
print("------\n");

// * 23. Find the restaurant Id, name, and grades for those rest
// * which achieved a grade of "A"
// * and scored 11 on an ISODate "2014-08-11T00:00:00Z" among ma
print("Q1, part 23");
restaurantCollection.find(
  {
    grades: {
      $elemMatch: {
        grade: "A",
        score: 11,
        date: ISODate("2014-08-11T00:00:00Z"),
      },
    },
  },
  { restaurant_id: 1, name: 1, grades: 1, _id: 0 },
); // * here we use $elemMatch to match multiple conditions for
print("------\n");

// * 24. Find the restaurant Id, name, and grades for those rest
// * where the 2nd element of the grades array contains
// * a grade of "A", a score of 9, and an ISODate "2014-08-11T0
print("Q1, part 24");
restaurantCollection.find(
  {
    "grades.1.grade": "A",
    "grades.1.score": 9,
    "grades.1.date": ISODate("2014-08-11T00:00:00Z"),
  },
  { restaurant_id: 1, name: 1, grades: 1, _id: 0 },
);
print("------\n");

// * 25. Find the restaurant Id, name, address, and geographical
// * where the 2nd element of the coordinates contains
// * a value more than 42 and up to 52.
print("Q1, part 25");
restaurantCollection.find(
  {
    "address.coord.1": { $gt: 42, $lte: 52 },
  },
  { restaurant_id: 1, name: 1, address: 1, _id: 0 },
);
print("------\n");
// endregion: Q1

// region: Q2
const durhamRestaurantsCollection = db[DURHAM_RESTAURANTS_TABLE]
const durhamForeclosuresCollection = db[DURHAM_FORECLOSURE_TABLE

print("Q2\n------\n");
// * get the target restaurants, where
// * Rpt_Area_Desc="restaurants" (i.e., Food Service)
// * Seats >= 100
const targetRestaurants = durhamRestaurantsCollection
  .find({
    "fields.rpt_area_desc": "Food Service",
    "fields.seats": { $gte: 100 },
  })
  .toArray();

// * find the min-max longitude and latitude to determine the po
let minLongitude = 180;
let maxLongitude = -180;
let minLatitude = 90;
let maxLatitude = -90;
targetRestaurants.forEach((restaurant) => {
  if (!restaurant.geometry) return;
  minLongitude = Math.min(restaurant.geometry.coordinates[0], mi
  maxLongitude = Math.max(restaurant.geometry.coordinates[0], ma
  minLatitude = Math.min(restaurant.geometry.coordinates[1], mir
  maxLatitude = Math.max(restaurant.geometry.coordinates[1], max
});

// * create a polygon object based on MongoDB's documentation
```

```
const targetPolygon = {
  type: "Polygon",
  coordinates: [
    [
      [minLongitude, minLatitude],
      [maxLongitude, minLatitude],
      [maxLongitude, maxLatitude],
      [minLongitude, maxLatitude],
      [minLongitude, minLatitude],
    ],
  ],
};

// * filter and count the foreclosures within our target polygo
const countForeclosures = durhamForeclosuresCollection
  .find({
    geometry: {
      $geoWithin: {
        $geometry: targetPolygon,
      },
    },
  })
  .count();
print(`Number of foreclosures within the target polygon is ${co
// endregion: Q2
```