S02.05 worksheet on MongoDB

The context

Trees are a huge asset in the path to resilience with regard to climate change.

The city of Paris has a strong policy to use trees to lower the temperatures during heat waves.

"La Ville s’est fixé un objectif de débitumiser 100 hectares et de planter 170 000 arbres entre 2020 et 2026. " https://www.paris.fr/pages/paris-s-adapte-au-changement-climatique-18541#plus-de-vert-moins-de-bitume-et-plus-d-arbres

see also "le plan arbres"
https://cdn.paris.fr/paris/2021/12/13/daf6cce214190a66c7919b34989cf1ed.pdf

You want to launch a new app where people can find shade when the temperature rises. Given a user location and the app finds the closest garden with the largest amount of trees.

You have an important meeting with the Paris team in charge of trees where you want to demonstrate a POC (proof of concept) of your new app.

Fortunately, there is a couple of datasets you can leverage available on the Paris open data platform

- The Paris Tree dataset that includes over 210k tress in and around Paris.
- The "Espaces verts" dataset (green spaces, aka gardens) with over 2300 areas.

The first phase of the POC, plan is to use these 2 datasets to figure out how many trees each Espace vert, green area, contains.

These are real world datasets and as such have their share of anomalies.

In today's workshop you will

- load both datasets into MongoDB collections and compare schema designs
- clean up the data by creating validation rules
- learn to use GeoJSON
- find the best schema pattern to reconcile both dataset in one.

Plan

The plan of this workshop

1. flat schema scenario for the trees dataset
    1. create database, import data directly from json file
    2. insert data, time it
    3. some queries
2. nested schema scenario for the trees dataset
    1. create schema
    2. insert data, time it
    3. some queries
3. add validation to nested schema
    1. flag dimension outliers
    2. flag duplicates geolocations using an index

4. Explore the dataset with semi-complex queries
    1. explain and add index to optimize
5. import the gardens dataset into its own collection
    1. Explore with some queries
6. Create a pipeline aggregation to Identify trees in a given area
7. build a new collection, and compare the 3 schema design scenarios
    1. gardens : nested trees
    2. gardens : reference trees
    3. gardens : nested and outlier pattern

dataset samples

- https://opendata.paris.fr/explore/dataset/les-arbres/information/
- https://opendata.paris.fr/explore/dataset/espaces_verts/information/

Here's a random sample of the trees dataset

```JavaScript
{
        "idbase":249403,
        "location_type":"Arbre",
        "domain":"Alignement",
        "arrondissement":"PARIS 20E ARRDT",
        "suppl_address":"54",
        "number":null,
        "address":"AVENUE GAMBETTA",
        "id_location":"1402008",
        "name":"Tilleul",
        "genre":"Tilia",
        "species":"tomentosa",
        "variety":null,
        "circumference":85,
        "height":10,
        "stage":"Adulte",
        "remarkable":"NON",
        "geo_point_2d":"48.86685102642415, 2.400262189227641"
    },
```

and a sample of the garden dataset

```JavaScript
    {
        "nom_de_espace_vert": "JARDINS DES CHAMPS ELYSEES - CARRE  DU THEATRE DU
        "typologie_espace_vert": "Promenades ouvertes",
        "categorie": "Jardin",
        "adresse_numero": 5.0,
        "adresse_complement": "X",
        "adresse_type_voie": "AVENUE DES",
        "adresse_libelle_voie": "CHAMPS ELYSEES",
        "code_postal": 75008.0,
        "presence_cloture": "Non",
        "annee_de_ouverture": 1928.0,
        "annee_de_changement_de_nom": 2024.0,
        "nombre_entites": 1.0,
        "ouverture_24h_24h": "Oui",
        "id_division": 104.0,
        "id_atelier_horticole": 9.0,
        "ida3d_enb": "46225",
        "site_villes": "3752",
        "id_eqpt": "4979",
        "competence": "CP",
        "geo_shape": "{\"coordinates\": [[[[2.3121917695809056, 48.8680666327570
        ...,  [2.3108033322716635, 48.86794252502758],
        ]]], \"type\": \"MultiPolygon\"}",
        "url_plan": "https://b22-pr-v1-iis01.ressources.paris.mdp/MAAP.asp?objec
        "geo_point": "48.86783322261344, 2.3111820973921144"
    },
```

Expectations and work

Each worksheet step follows

- context
- goal statement
- example of script
- create your version
- assess the results
- submit the answer

We work by default in the terminal with mongosh.

But if you prefer to work with Mongo Compass or a driver (python, node.js, etc ...) please do so. This work not about scripting. Feel free to use the best tool available for you.

Step 1: trees and flat schema

Let's start simple with a flat schema. Each tree document only has one level of information.

The dataset is available here

Create a new database

Let's create a new database and call it `trees_flat_db`.

```Bash
use trees_flat_db
```

and create a trees collection with

```Bash
db.createCollection("trees")
```

check that the database has been created with

```Bash
show dbs
```

Note: if you need space on your ATLAS free tier server, you cna drop a database (for instance `sample_airbnb`) with

```Bash
# connect to the database
use sample_airbnb
# drop the db
db.dropDatabase('sample_airbnb')
```

Somehow you cannot drop a database that is not the current one.

Let's load the data

We have several options . including using the `mongoimport` from the command line. `mongoimport` needs to be installed separately. It is usually the fastest option for large volumes.

Here we use the `load()` function

Let's time the operation

```JavaScript
const fs = require("fs");

// Load and parse the JSON file
const dataPath = "./trees_flat.json"
const treesData = JSON.parse(fs.readFileSync(dataPath, "utf8"));

// Insert data into the desired collection
let startTime = new Date()
db.trees.insertMany(treesData);
let endTime = new Date()
print(`Operation took ${endTime - startTime} milliseconds`)
```

Write down that time somewhere. On my mac it took milliseconds for the entire 211k+ trees.

To delete all the documents from the trees collection

```JavaScript
db.treesdb.deleteMany({})
```

To check that all the rows have been inserted correctly

```JavaScript
db.trees.stats()
```

Using mongoimport

using mongosh scripting may take too long

You can use mongoimport

make sure you load the ndjson formatted file (one document per line) and not the json file (an array of documents)

```Bash
mongoimport --uri="mongodb+srv://<username>:<password>@<cluster-url>" \
  --db <database_name> --collection trees --file ./trees_flat.ndjson --numInsert
```

```Bash
time mongoimport --uri="mongodb+srv://<username>:<password>@<cluster-url>" \
  --db <database_name> --collection trees --file ./trees_flat.ndjson --numInsert
```

play with batchSize

Queries

Explore the dataset, write the following queries

- Count trees per domain

```JavaScript
db.trees.aggregate([
  { $group: {
      _id: "$domain",
      count: { $sum: 1 }
  }},
  { $sort: { count: -1 }}
])
```

- Stats for remarkable trees

```JavaScript
db.trees.aggregate([
  { $match: {
      remarkable: "OUI",
      height: { $gt: 0 },
      circumference: { $gt: 0 }
  }},
  { $group: {
      _id: null,
      avgHeight: { $avg: "$height" },
      avgCircumference: { $avg: "$circumference" },
      count: { $sum: 1 }
  }}
])
```

- stats for non remarkable trees : field is missing

```JavaScript
db.trees.aggregate([
 { $match: {
     remarkable: { $exists: false },
     height: { $gt: 0 },
     circumference: { $gt: 0 }
 }},
 { $group: {
     _id: null,
     avgHeight: { $avg: "$height" },
     avgCircumference: { $avg: "$circumference" },
     count: { $sum: 1 }
 }}
])
```

- top 5 names of trees

```javascript
db.trees.aggregate([
 { $group: {
   _id: "$name",
   count: { $sum: 1 }
 }},
 { $sort: { count: -1 }},
 { $limit: 5 }
])
```

now let's focus on the Platane which is the most common tree in Paris,

What makes these remarkable tree

check the avg heaight and circ for Pllatanses

- Stats for Platane trees by remarkable status
- check also for Marronier

```javascript
db.trees.aggregate([
 { $match: { name: "Platane" }},
 { $group: {
   _id: { remarkable: { $ifNull: ["$remarkable", "NOT_SET"] }},
   count: { $sum: 1 },
   avgHeight: { $avg: "$height" },
   maxHeight: { $max: "$height" },
   avgCircumference: { $avg: "$circumference" }
 }},
 { $sort: { "_id.remarkable": 1 }}
])
```

So can you conclude what makes these trees remarkable ?

just for the sake of it ... add a field that calculates the age of the tree

the rule is very roughly

age in years = Circumference / π / 1cm

```javascript
db.trees.aggregate([
 { $addFields: {
   age: { $divide: ["$circumference", 3.14159] }
 }}
])
```

and see the stats

```JavaScript
db.trees.aggregate([
  { $addFields: {
    age: { $divide: ["$circumference", 3.14159] }
  }},
  { $match: {
    circumference: { $gt: 0 }
  }},
  { $group: {
    _id: null,
    avgAge: { $avg: "$age" },
    minAge: { $min: "$age" },
    maxAge: { $max: "$age" }
  }}
])
```

you can use $set to create the field age and make it permanent

```JavaScript
db.trees.updateMany(
 { circumference: { $gt: 0 }},
 [{
   $set: {
     age: { $divide: ["$circumference", 3.14159] }
   }
 }]
)
```

Looking at the max age ... Houston we have a problem

As we've seen some trees have anomalies in their measurements. Notably the max height is 2524 meters and the max circumference is 80105 cm. Most probably a human error.

Note: it's possible to visually check if a tree has the dimensions indicated in the dataset. Just search for the latitutde and longitude on google maps and check out the street photos. For instance this tree is supposed to have `circumference: 1650` but the photo shows only normal sized Platanes}.

Validator

Before we import the data let's make sure that only trees with somewhat *normal* measures will get in the collection.

We can create the collection before importing the data with the function `db.createCollection(name, options)`.

The syntax for create collection can be found here

Note the `validator: <document>,` line. This is what we use to specify validation rules.

> The validator option takes a `document` that specifies the validation rules or expressions.

Check out the example given here

You can also specify what happens when a document is not valid.

- validationAction: What happens when a document fails validation
  - `error` (default): Rejects invalid documents
  - `warn`: Allows invalid documents but logs a warning message

Your task is to write a validator that requires

- `dimensions.height < 50`
- `dimensions.circumference < 500`
- required `name` and `location.geo_point_2d`

JavaScript

```
db.createCollection("trees", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name","geo_point_2d", "height", "circumference"],
      properties: {
        name: {
          bsonType: "string",
          description: "Name must be a string and is required"
        },
        height: {
          bsonType: "number",
          minimum: 0,
          maximum: 50,
          description: "Height must be between 0 and 50"
        },
        circumference: {
          bsonType: "number",
          minimum: 0,
          maximum: 500,
          description: "Circumference must be between 0 and 500"
        }
      }
    },
    validationAction: "error",
});
```

before you import the data with `mongoimport` or with `db.trees.insertMany()`. Specify which action on validation you'd like.

- test both error and warn validationAction
- for each validationAction count the documents in the collection
- what do you observe ?

- don't forget to drop the collection between each run
- what happens if you don't drop the collection between each run ?

uniques

Besides crazy heights and insane circumferences, the dataset also holds geolocation duplicates.

Write the aggregation pipeline that finds all the trees that share the same geolocation.

You should find 22 trees that have the same geolocation

The best way to avoid duplicates is to add a unique index

- drop the collection and recreate it with the validator
- add a unique index with

```JavaScript
db.collection.createIndex({ "fieldName": 1 }, { unique: true })
```

check with `db.trees.getIndexes()`

Now reimport the data

you should see messages like

```Bash
2024-12-08T12:12:32.561+0100    continuing through error: E11000 duplicate key e
```

and if you rerun the aggregate pipeline for duplicates geolocations it should return nothing.

Break : load the whole dataset

Load the whole dataset use the ndjson forrmat (rm --jsonArray)

```Bash
mongoimport --uri="mongodb+srv://<>@skatai.w932a.mongodb.net" \
  --db trees_flat_db --collection trees --file ./trees_flat.ndjson --verbose=100
```

takes approx. 15mn

```Bash
2024-12-08T12:44:27.147+0100    209895 document(s) imported successfully. 1444 d
```

Once that is done you should clone teh collection so that if anything breaks you do not have to reload the whole dataset

Cloning the collection

You can do so with

```JavaScript
db.trees.aggregate([
   { $match: {} },  // This matches all documents
   { $out: "trees_backup" }  // Creates a new collection named trees_backup
])
```

GeoJSON

next

- convert string to geojson
- find your address lat and long
- find number nearest trees given location
- do you have a remarkable tree ?

---

conversions

The geolocation in the database is just a string with `<latitude>,<longitude>`.

Let's convert it to GeoJSON format.

see geoJSON in MongoDB for reference

This will allow us to

- find trees near a location

for instance the trees within 100 meters of the Eiffel tower

```JavaScript
// Find how many trees are within 100 meters of the Eiffel Tower
db.trees.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [2.294694, 48.858093]  // Eiffel Tower coordinates
      },
      $maxDistance: 100  // in meters
    }
  }
}).count()
```

- Find trees and calculate their distance from a particular location

```javascript
db.trees.aggregate([
  {
    $geoNear: {
      near: {
        type: "Point",
        coordinates: [2.294694, 48.858093]
      },
      distanceField: "distance",
      spherical: true,
      query: { "genre": "Tilia" }  // Only Linden trees
    }
  }
])
```

or

- "Neighborhood Biodiversity": Count different species within circular areas (using $geoWithin with $centerSphere)
- "Tree Clusters": Find areas with high tree density (using $geoNear with aggregation)

Convert the geo*point*2d string into a Point

But first we must convert the geo*point*2d string 'lat, long' into a geoJSON point.

Note: contrary to common speech where latitude comes before longitude, the geoJSON points have the longtitude first and the latitude second

Build an aggregation pipeline that

- splits the `geo_point_2d` field into an array of 2 numbers into a temporary field `temp`
- then converts the array into a Point

1st step : split the string into an array of floats

Since we haven't had much practice with that type of aggregation pipelines let's try to write the first step given this description

> The `$addFields` stage adds new fields to documents without modifying the existing fields. In this case, we're creating a new field called "coordinates". Think of it like adding a new column to a spreadsheet, but one that's calculated from existing data. Let's look at what's happening inside, working from the innermost operation outward: $split: ["$location_string", ", "]
>
> - This takes our input string like "48.86685102642415, 2.400262189227641"
> - The $split operator cuts this string wherever it finds ", " (comma followed by space)
> - The result is an array: ["48.86685102642415", "2.400262189227641"]
>
> The `$map` operator is like a transformation assembly line: - input: Takes the array we just created from splitting - as: "coord": For each piece of the array, temporarily names it "coord" (the $$ in $$coord is how we reference this temporary variable) - in: { $toDouble: "$$coord" }: This is the

> transformation applied to each piece
>
> - $toDouble converts each string number into an actual numeric value:

2nd step convert the array of floats into a Point

The $addFields stage here creates a new field called "location_geojson" that follows the GeoJSON specification.

The $arrayElemAt operator is like reaching into a specific position in an array. Think of it like reaching into a box of two numbers and pulling out either the first or second number. The syntax is:

First argument: the array we're looking at ("$coordinates") Second argument: which position we want (0 for first, 1 for second)

covert gardens geolocation

https://claude.ai/chat/eee56da3-bb07-4437-b71f-419059dbf80d

transform point to GeoJSON

https://chatgpt.com/c/6753fcbe-20c8-800e-a730-a4be1208201c

Solution

It's best to test the pipeline over a few documents

add `{ $limit: 5 },` at the **beginning** of your pipeline.

```Bash
db.trees.aggregate([
  { $limit: 1000 },  // Get just 5 documents to work with
  {
    $addFields: {
      // Split the string into an array of [latitude, longitude]
      coordinates: {
        $map: {
          input: { $split: ["$geo_point_2d", ", "] },
          as: "coord",
          in: { $toDouble: "$$coord" } // Convert to numbers
        }
      }
    }
  },
  {
    $addFields: {
      // Create a GeoJSON object with reordered coordinates [longitude, latitude
      location_geojson: {
        type: "Point",
        coordinates: [
          { $arrayElemAt: ["$coordinates", 1] }, // longitude
          { $arrayElemAt: ["$coordinates", 0] }  // latitude
        ]
      }
    }
  },
  {
    $unset: "coordinates" // Optionally remove the intermediate array field
  },
  {
    $merge: {     // Or use $out if you want to create a new collection
      into: "trees",
      whenMatched: "merge",
      whenNotMatched: "discard"
    }
  }
]);
```

check that the right number of documents now have the `location_geojson` field

```JavaScript
db.trees.countDocuments({ location_geojson : {$exists : true}   } )
```

add a

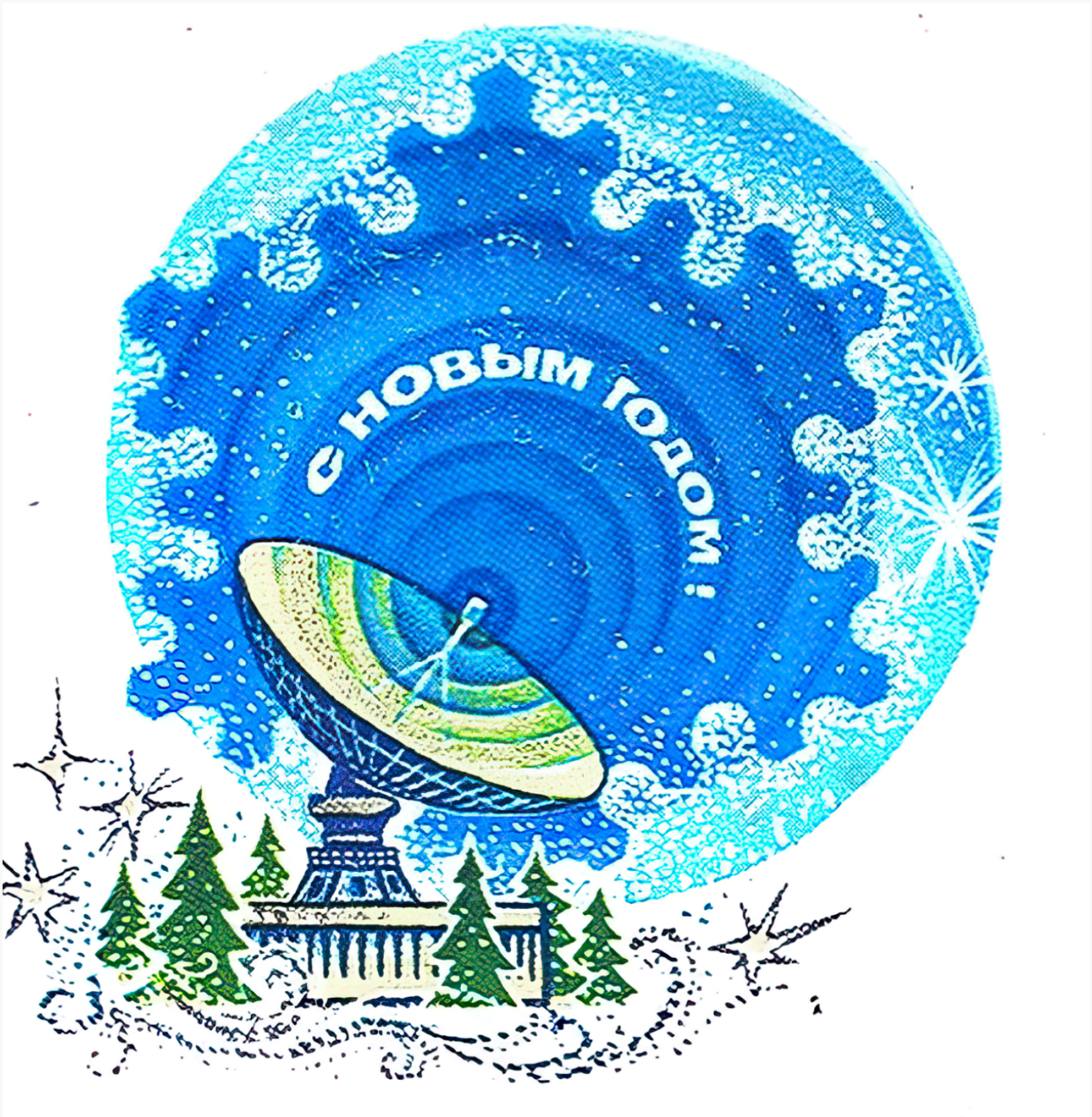
You need to create a geospatial index first. Add this before running the query:

db.trees.createIndex({ location_geojson: "2dsphere" })

query

Now let's take an address

you can find the related longitude and latitude using google maps. just click right on the red pin



you get for instance : 48.82124804336634, 2.363645912660704

Let's find the number of trees around this address, in a radius of 50m

```JavaScript
// Find how many trees are within 100 meters of Tricotin
db.trees.find({
  location_geojson: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [2.363645912660704, 48.82124804336634]  // Tricotin coordir
      },
      $maxDistance: 100  // in meters
    }
  }
}).count()
```

What about Epita ? with coordinates 48.815820795886815, 2.362806368481762

```JavaScript
// Find how many trees are within 100 meters of Tricotin
db.trees.find({
  location_geojson: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [2.362806368481762, 48.815820795886815]  // Epita coordinat
      },
      $maxDistance: 100  // in meters
    }
  }
}).count()
```

only 11 trees!

What about your own address ?

Can you list the number of trees for each name ?

write the query that returns the number of trees by name around a 100m radius of uyour address

Just add a group by $name after teh first geo filter

```JavaScript
  {
   $group: {
     _id: "$name",
     count: { $sum: 1 }
   }
 },
```

```javascript
db.trees.aggregate([
 {
   $geoNear: {
     near: {
       type: "Point",
        // coordinates: [2.362806368481762, 48.815820795886815] // Epita
        // coordinates: [2.363645912660704, 48.82124804336634]  // Tricotin coor
        coordinates: [2.3368673391467576, 48.841417646345064]
    //    coordinates: [2.336245066633228, 48.84691090730081] // jardin du luxem
    //    coordinates: [2.348377228006662, 48.87940121485902]
     },
     distanceField: "distance",
     maxDistance: 100,
     spherical: true,
     key: "location_geojson"
   }
 },
 {
   $group: {
     _id: "$name",
     count: { $sum: 1 }
   }
 },
 {
   $sort: { count: -1 }
 }
]);
```

db.trees.aggregate([{ $geoNear: { near: { type: "Point", coordinates: [2.3368673391467576, 48.841417646345064] }, distanceField: "distance", maxDistance: 100, spherical: true, key: "location_geojson" } }]);

db.trees.find({ location_geojson: { $near: { $geometry: { type: "Point", coordinates: [2.3368673391467576, 48.841417646345064] }, $maxDistance: 100 // in meters } } }).count()

48.841417646345064, 2.3368673391467576

Gardens

Next, - load the gardens dataset into a gardens collection - transform the geo_shape into a compatible geoJSON array of coordinates

- for a given garden find all the trees and their names
- what are the grdents with the highest number of trees ?
- what number fo trees could be considered as a threshold between normal gardens and ones with a large numbe rof trees

Next,

create a new collection of gardens and trees

for each garden add a nested collection of documents with all the trees in the garden

Does that exceed the 16mb limit for some gardens

Apply the outlier schema to gardens and trees using the threshold you have found before.