DATA PRE-PROCESSING

The csv with the decoded ship location messages (nari_dynamic) was imported into MongoDB Compass, 1 million records were used. The collection was named dynamic_locs. Below is the “preprocessing” done on the specific collection of the database, through the MongoSH GUI.

STEP 1

Initially, UNIX epochs timestamps were converted to Dates as follows:

```
var new_docs= db.dynamic_locations.aggregate({$set: {"timestamp": {$toDate: {$multiply: ['$t', 1000]}}}}).toArray()

db.dynamic_locs.insertMany(new_docs)

db.dynamic_locs.updateMany({}, {$unset: {t:""}})
```

STEP 2

Set the latitude and longitude as location and then deleted the 'lon' and 'lat' fields:

```
db.dynamic_locs.updateMany({},[{"$set":{"location":{type:"Point",coordinates:["$lon", "$lat"] } }} ] )

db.dynamic_locs.updateMany({},{"$unset":{ lon: 1, lat: 1 }})
```

STEP 3

A new field named mmsi_country_code was created which contains the first three digits of the field sourcemmsi and which correspond to the code of the country to which the ship belongs.

```
var new_docs1= db.dynamic_locations.aggregate({"$set": {"country_code": { $substr: ["$sourcemmsi", 0, 3] }}}).toArray()

db.dynamic_locs.insertMany(new_docs1)

db.dynamic_locs.updateMany({}, [ { $set: { mmsi_country_codes: { $toInt: "$country_code" } } }, { $unset: "country_code" }])
```

The nari_static.csv file containing static and travel-related information was also inserted into the database, after some fields were first deleted using python. Then it was necessary to convert the timestamps that were in UNIX epochs format to Dates. This collection was named static_data.

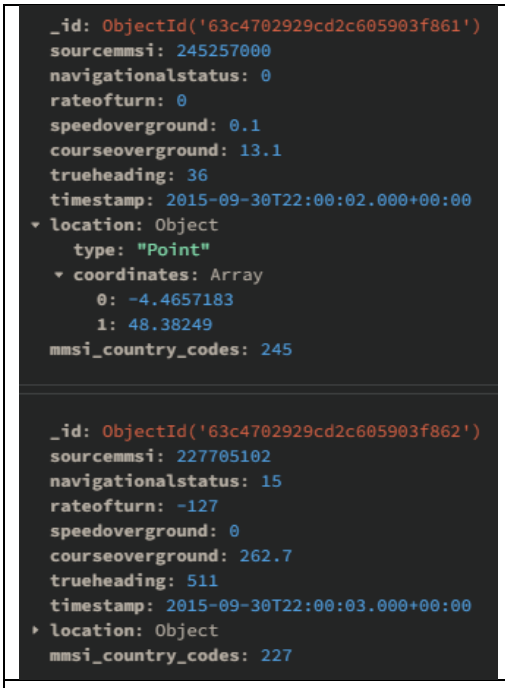
```
var new_docs1= db.static_data.aggregate({$set: {"timestamp": {$toDate: {$multiply: ['$t', 1000]}}}}).toArray()
```

db.static_data.insertMany(new_docs01)

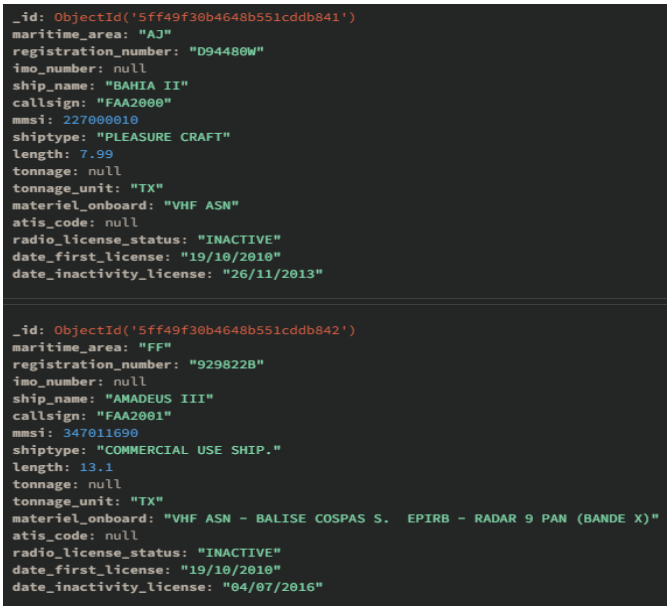
db.static_data.updateMany({}, {$unset: {t:""}})

Anfr.csv was then imported which contains a list of vessels recorded by ANFR. No further processing was required. This collection was called vessels. Finally, the csv file entitled MMSI Country Codes was also used, which was saved in the collection named countries.

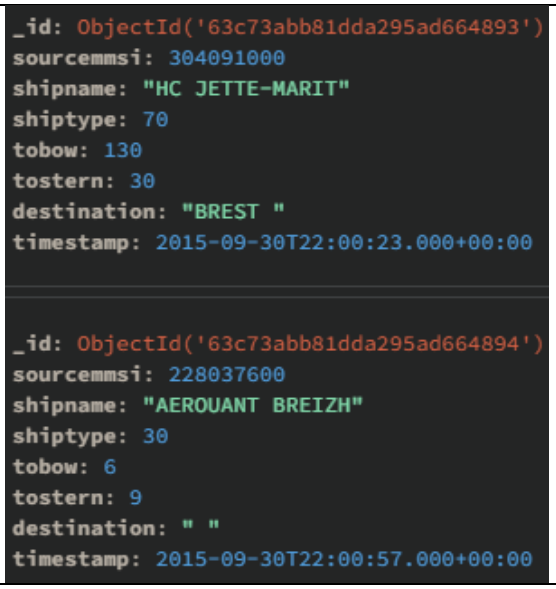
The following screenshots show samples from the documents of each collection.



dynamic_locs collection



Collection of vessels



static_data collection

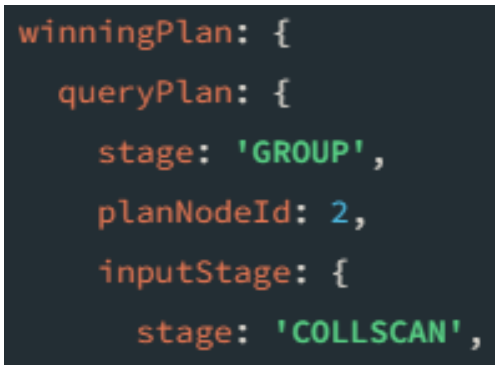
Countries collection

QUERIES AND INDEXES

Relational query on collection vessels: Are you looking for the sea area (maritime_area) which contains most pleasure boats (shiptype: “PLEASURE CRAFT”) and which have a length between 5 and 6 meters (length between 5 and 6)

```
db.vessels.explain(“executionStats).aggregate([{$match: {$and: [{shiptype: "PLEASURE CRAFT"}, {length: {$gt: 5, $lt: 6}}]}}, {$group: {_id: "$maritime_area", count: { $sum: 1 }}}, {$sort: {count: -1}}, {$limit: 1}])
```

Runtime, keys and files that were examined



The index used

The specific query was executed five times and the average execution time of the query was calculated (mean_executionTimeMillis: 108.2ms).

Then three different indexes were created:

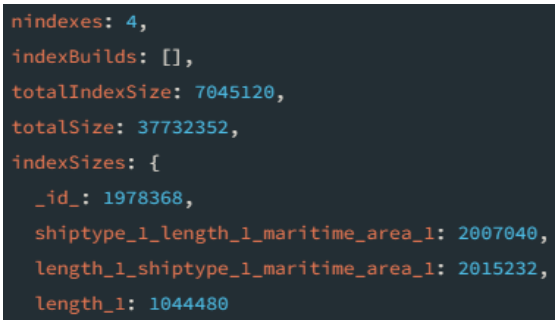
```
db.vessels.createIndex({shiptype: 1, length: 1, maritime_area:1})
```

```
db.vessels.createIndex({length: 1, shiptype: 1, maritime_area:1})
```

```
db.vessels.createIndex({length: 1})
```

as shown by the command

```
db.vessels.stats()
```



The screenshot shows the number of indexes as well as the size of each index in bytes.

The query was run again to record the new execution time, as well as to find which index is being used.

```
executionStats: {
  executionSuccess: true,
  nReturned: 49,
  executionTimeMillis: 9,
  totalKeysExamined: 6630,
  totalDocsExamined: 0,
```

Runtime, keys and files that were examined

```
winningPlan: {
  queryPlan: {
    stage: 'GROUP',
    planNodeId: 3,
    inputStage: {
      stage: 'PROJECTION_COVERED',
      planNodeId: 2,
      transformBy: {
        maritime_area: true,
        _id: false
      },
      inputStage: {
        stage: 'IXSCAN',
        planNodeId: 1,
        keyPattern: {
          shiptype: 1,
          length: 1,
          maritime_area: 1
        },
        indexName: 'shiptype_1_length_1_maritime_area_1'
```

The index used

Time was reduced tomean_executionTimeMillis: 9msand the index used was {shiptype: 1, length: 1, maritime_area:1}. However, the size of the index must also be taken into account.

<u>Spatial query on collection dynamic_locs:</u> Searching for ship codes (sourcemmsi) that have been found at any time within the marine natural park of Iroise, which is located west of the French city of Brest.

The following query uses the coordinates that define the specific polygon.

```
db.dynamic_locs.aggregate([
  {$match: {
    location: {
      $geoWithin: {
        $geometry: {
          type: "Polygon" ,
          coordinates: [[
            [-5.1759338,48.448333],
            [-5.1759338,48.448333],
            [-5.0008392,48.2955285],
            [-4.8353577,48.302837],
            [-4.8319244,48.3772333],
            [-5.072937,48.5097813],
            [-5.1759338,48.448333]
          ]]
        }
      }
    }
  }},
  {$group: {
    _id: "$sourcemmsi"
  }},
  {$group: {
    _id: null,
    sourcemmsi: {$addToSet: "$_id"}
  }},
  {$project: {_id: 0, sourcemmsi: 1}}
])
```

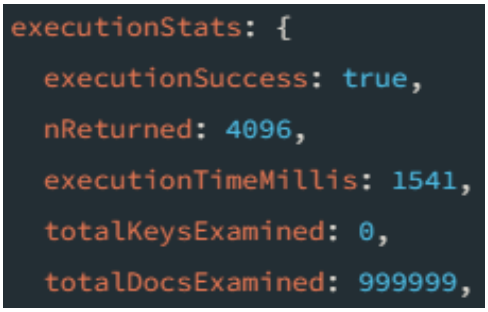
Spatial question


```
{
  sourcemmsi: [
    228183600,
    227536950,
    226338000,
    228022900,
    227008170,
    228211900,
    228813000,
    227820000,
    228017700,
    226263000,
    228064900,
    226084000,
    228336000,
    245257000,
    227114300,
    227003050,
    245334000,
    227002330,
    226318000,
    227312180
  ]
}
```

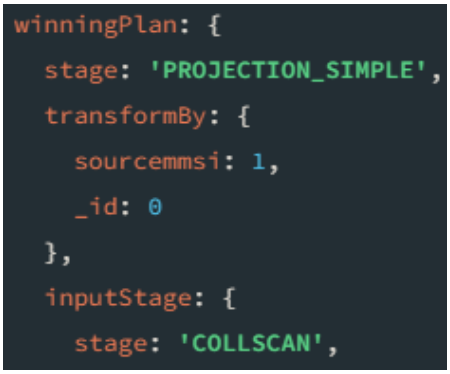

Result

db.dynamic_locs.aggregate([{$match: {location: {$geoWithin: {$geometry: {type: "Polygon" , "coordinates":[[[-5.1759338,48.448333],[-5.1759338,48.448333],[-5.0008392,48.2955285],[-4.8353577,48.302837],[-4.8319244,48.3772333],[-5.072937,48.5097813],[-5.1759338,48.448333]]]}}}}},{$group: {_id: "$sourcemmsi"}},{$group: {_id: null,sourcemmsi: {$addToSet: "$_id"}}},{$project: {_id: 0, sourcemmsi: 1}}])

Data from the execution of the spatial query, before index creation.



Runtime, keys and files that were examined



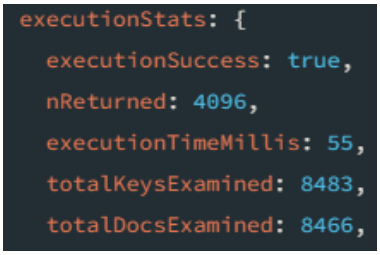
The index used

In the five executions of the query, the average execution time was:
mean_executionTimeMillis: 1558.2ms

2dsphere index was then created which is a type of index in MongoDB, used to support queries that use MongoDB's geospatial feature. It allows MongoDB to efficiently find documents located within a specified geographic area by using spherical geometry to represent points on a sphere (Earth) rather than a flat plane.

```
db.dynamic_locs.createIndex({"location":"2dsphere"})
```

Data from the execution of the spatial query, after creating the index.



In the screenshot on the right, it looks like the index used was 2dsphere, while in the photo above run time are shown, as well as the keys and files examined

The average execution time at five executions were:
mean_executionTimeMillis: 46.8ms

Spatio-temporal query in dynamic_locs collection: Searching for ship codes (sourcemmsi) which were found within the protected area of the Iroise park (NATURA 2000), in a specific time range.

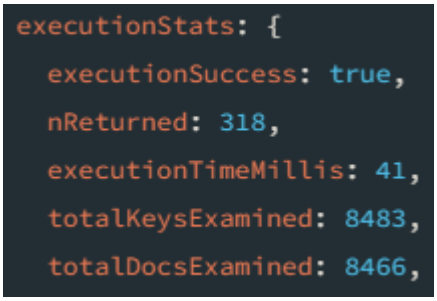
```
db.dynamic_locs.aggregate([
  {$match: {
    location: {
      $geoWithin: {
        $geometry: {
          type: "Polygon" ,
          coordinates: [[
            [-5.1759338,48.448333],
            [-5.1759338,48.448333],
            [-5.0008392,48.2955285],
            [-4.8353577,48.302837],
            [-4.8319244,48.3772333],
            [-5.072937,48.5097813],
            [-5.1759338,48.448333]
          ]]
        }
      }
    },
    timestamp: {
      $gte: ISODate("2015-09-30T22:00:02.000+00:00"),
      $lte: ISODate("2015-10-01T22:00:02.000+00:00")
    }
  }},
  {$group: {
    _id: "$sourcemmsi"
  }},
  {$group: {
    _id: null,
    sourcemmsi: {$addToSet: "$_id"}
  }},
  {$project: {_id: 0, sourcemmsi: 1}}
])
{
  sourcemmsi: [
    245257000,
    227008170,
    228017700,
```

```
executionStats: {
  executionSuccess: true,
  nReturned: 318,
  executionTimeMillis: 655,
  totalKeysExamined: 0,
  totalDocsExamined: 999999,
```

```
winningPlan: {
  stage: 'PROJECTION_SIMPLE',
  transformBy: {
    sourcemmsi: 1,
    _id: 0
  },
  inputStage: {
    stage: 'COLLSCAN',
```

In the screenshots above, the output of the query is first shown, and then its execution time, etc., as well as the Index used. Has not the index is still created. (5 runs,mean_executionTimeMillis: 657.4ms)

Data from the execution of the spatiotemporal query, after index creation.
`db.dynamic_locs.createIndex({"location":"2dsphere"})`



Runtime, keys and files that were examined

The index used

mean_executionTimeMillis: 43ms