Single-Collection Designs in MongoDB with Spring Data (Part 2)

Graeme Robinson

10 min read • Published Oct 18, 2022 • Updated Aug 12, 2024

Spring MongoDB Java

In Part 1 of this two-part series, we discussed single-collection design patterns in MongoDB and how they can be used to avoid the need for computationally expensive joins across collections. In this second part of the series, we will provide examples of how the single-collection pattern can be utilized in Java applications using Spring Data MongoDB and, in particular, how documents representing different classes but residing in the same collection can be accessed.

Accessing polymorphic single collection data using Spring Data MongoDB

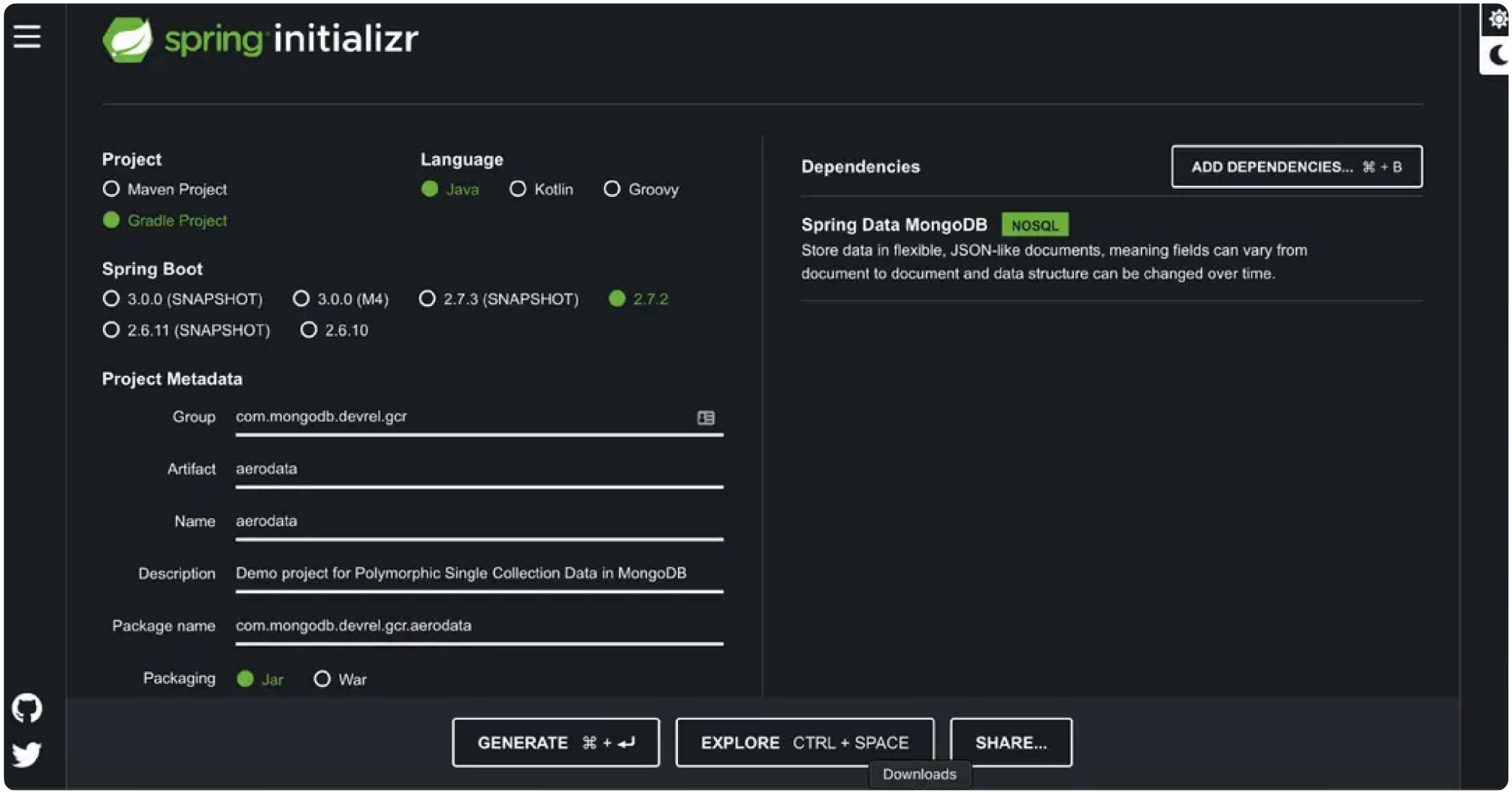
Whilst official, native idiomatic interfaces for MongoDB are available for 12 different programming languages, with community-provided interfaces available for many more, many of our customers have significant existing investment

and knowledge developing Java applications using Spring Data. A common question we are asked is how can polymorphic single-collection documents be accessed using Spring Data MongoDB?

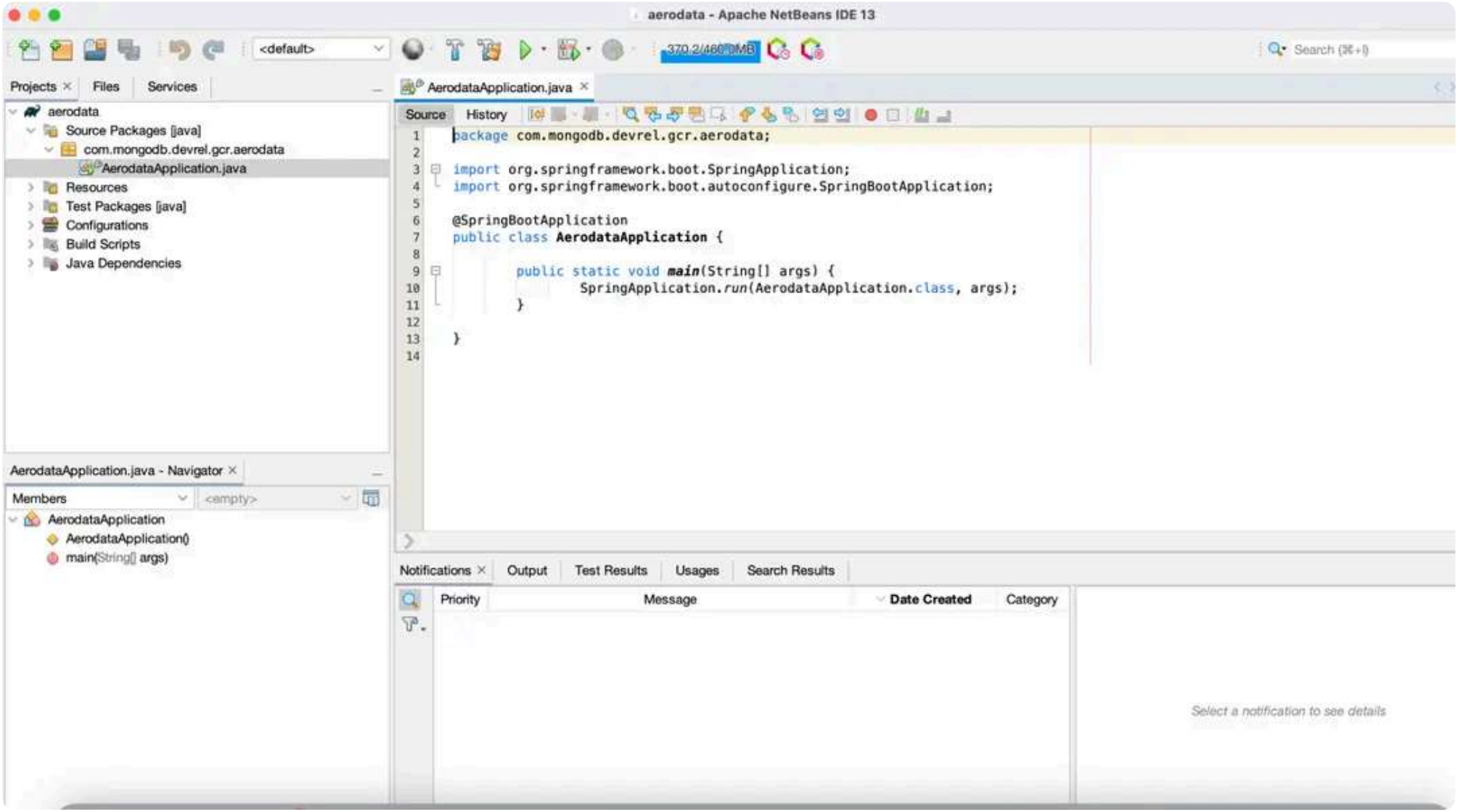
In the next few steps, I will show you how the Spring Data template model can be used to map airline, aircraft, and ADSB position report documents in a single collection named aerodata, to corresponding POJOs in a Spring application.

The code examples that follow were created using the Netbeans IDE, but any IDE supporting Java IDE, including Eclipse and IntelliJ IDEA, can be used.

To get started, visit the Spring Initializr website and create a new Spring Boot project, adding Spring Data MongoDB as a dependency. In my example, I'm using Gradle, but you can use Maven, if you prefer.



Generate your template project, unpack it, and open it in your IDE:



Add a package to your project to store the POJO, repository class, and interface definitions. (In my project, I created a package called (com.mongodb.devrel.gcr.aerodata). For our demo, we will add four POJOs – AeroData, Airline, Aircraft, and ADSBRecord – to represent our data, with four corresponding repository interface definitions. AeroData will be an abstract base class from which the other POJOs will extend:

```
package com.mongodb.devrel.gcr.aerodata;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "aeroData")
public abstract class AeroData {

    @Id
    public String id;
    public Integer recordType;

    //Getters and Setters...

}
```

```
package com.mongodb.devrel.gcr.aerodata;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface AeroDataRepository extends MongoRepository<AeroData, String>{

}
```

```
package com.mongodb.devrel.gcr.aerodata;

import org.springframework.data.annotation.TypeAlias;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "aeroData")
@TypeAlias("AirlineData")
public class Airline extends AeroData{

```

```
    public String airlineName;
    public String country;
    public String countryISO;
    public String callsign;
    public String website;

    public Airline(String id, String airlineName, String country, String countryISO, String callsign, String website)
        this.id = id;
        this.airlineName = airlineName;
        this.country = country;
        this.countryISO = countryISO;
        this.callsign = callsign;
        this.website = website;
    }

    @Override
    public String toString() {
        return String.format(
            "Airline[id=%s, name='%s', country='%s (%s)', callsign='%s', website='%s']",
            id, airlineName, country, countryISO, callsign, website);
    }

}
```

```
package com.mongodb.devrel.gcr.aerodata;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface AirlineRepository extends MongoRepository<Airline, String>{

}
```

```
package com.mongodb.devrel.gcr.aerodata;
```

```java

import org.springframework.data.annotation.TypeAlias;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "aeroData")
@TypeAlias("AircraftData")
public class Aircraft extends AeroData{

    public String tailNumber;
    public String manufacturer;
    public String model;

    public Aircraft(String id, String tailNumber, String manufacturer, String model) {
        this.id = id;
        this.tailNumber = tailNumber;
        this.manufacturer = manufacturer;
        this.model = model;
    }

    @Override
    public String toString() {
        return String.format(
            "Aircraft[id=%s, tailNumber='%s', manufacturer='%s', model='%s']",
            id, tailNumber, manufacturer, model);
    }

}
```

```java
package com.mongodb.devrel.gcr.aerodata;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface AircraftRepository extends MongoRepository<Aircraft, String>{

}
```

```java
package com.mongodb.devrel.gcr.aerodata;

import java.util.Date;
import org.springframework.data.annotation.TypeAlias;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "aeroData")
@TypeAlias("ADSBRecord")
public class ADSBRecord extends AeroData {

    public Integer altitude;
    public Integer heading;
    public Integer speed;
    public Integer verticalSpeed;
    public Date timestamp;
    public GeoPoint geoPoint;

    public ADSBRecord(String id, Integer altitude, Integer heading, Integer speed, Integer verticalSpeed, Date timest
        this.id = id;
        this.altitude = altitude;
        this.heading = heading;
        this.speed = speed;
        this.verticalSpeed = verticalSpeed;
        this.timestamp = timestamp;
        this.geoPoint = geoPoint;
    }

    @Override
    public String toString() {
        return String.format(
            "ADSB[id=%s, altitude='%d', heading='%d', speed='%d', verticalSpeed='%d' timestamp='%tc', latitude='%f',
            id, altitude, heading, speed, verticalSpeed, timestamp, geoPoint == null ? null : geoPoint.coordinates[1]
    }
}
```

```
package com.mongodb.devrel.gcr.aerodata;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface ADSBRecordRepository extends MongoRepository<ADSBRecord, String>{

}
```

We'll also add a GeoPoint class to hold location information within the ADSBRecord objects:

```
package com.mongodb.devrel.gcr.aerodata;

public class GeoPoint {
    public String type;
    public Double[] coordinates;

    //Getters and Setters...
}
```

Note the annotations used in the four main POJO classes. We've used the "@Document" annotation to specify the MongoDB collection into which data for each class should be saved. In each case, we've specified the "aeroData" collection. In the Airline, Aircraft, and ADSBRecord classes, we've also used the "@TypeAlias" annotation. Spring Data will automatically add a "_class" field to each of our documents containing the Java class name of the originating object. The TypeAlias annotation allows us to override the value saved in this field and can be useful early in a project's development if it's suspected the class type may change. Finally, in the AeroData abstract class, we've used the "@id" annotation to specify the field Spring Data will use in the MongoDB _id field of our documents.

Let's go ahead and update our project to add and retrieve some data. Start by adding your MongoDB connection URI to application.properties. (A free MongoDB Atlas cluster can be created if you need one by signing up at cloud.mongodb.com.)

```
spring.data.mongodb.uri=mongodb://myusername:mypassword@abc-c0-shard-00-00.ftspj.mongodb.net:27017,abc-c0-shard-00-01.
```

Note that having unencrypted user credentials in a properties file is obviously not best practice from a security standpoint and this approach should only be used for testing and educational purposes. For more details on options for connecting to MongoDB securely, including the use of keystores and cloud identity mechanisms, refer to the MongoDB documentation.

With our connection details in place, we can now update the main application entry class. Because we are not using a view or controller, we'll set the application up as a CommandLineRunner to view output on the command line:

```
package com.mongodb.devrel.gcr.aerodata;

import java.util.Date;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AerodataApplication implements CommandLineRunner {

    @Autowired
    private AirlineRepository airlineRepo;
```

```
15
16      @Autowired
17      private AircraftRepository aircraftRepo;
18
19      @Autowired
20      private ADSBRecordRepository adsbRepo;
21
22      public static void main(String[] args) {
23          SpringApplication.run(AerodataApplication.class, args);
24      }
25
26      @Override
27      public void run(String... args) throws Exception {
28
29          // save an airline
30          airlineRepo.save(new Airline("DAL", "Delta Air Lines", "United States", "US", "DELTA", "delta.com"));
31
32          // add some aircraft aircraft
33          aircraftRepo.save(new Aircraft("DAL_a93d7c", "N695CA", "Bombardier Inc", "CL-600-2D24"));
34          aircraftRepo.save(new Aircraft("DAL_ab8379", "N8409N", "Bombardier Inc", "CL-600-2B19"));
35          aircraftRepo.save(new Aircraft("DAL_a36f7e", "N8409N", "Airbus Industrie", "A319-114"));
36
37          //Add some ADSB position reports
38          Double[] coords1 = {55.991776, -4.776722};
39          GeoPoint geoPoint = new GeoPoint(coords1);
40          adsbRepo.save(new ADSBRecord("DAL_a36f7e_1", 38825, 319, 428, 1024, new Date(1656980617041l), geoPoint));
41          Double[] coords2 = {55.994843, -4.781466};
42          geoPoint = new GeoPoint(coords2);
43          adsbRepo.save(new ADSBRecord("DAL_a36f7e_2", 38875, 319, 429, 1024, new Date(1656980618041l), geoPoint));
44          Double[] coords3 = {55.99606, -4.783344};
45          geoPoint = new GeoPoint(coords3);
46          adsbRepo.save(new ADSBRecord("DAL_a36f7e_3", 38892, 319, 428, 1024, new Date(1656980619041l), geoPoint));
47
48
49          // fetch all airlines
```

```
50          System.out.println("Airlines found with findAll():");
51          System.out.println("-------------------------------");
52          for (Airline airline : airlineRepo.findAll()) {
53              System.out.println(airline);
54          }
55          // fetch a specific airline by ICAO ID
56          System.out.println("Airline found with findById():");
57          System.out.println("-------------------------------");
58          Optional<Airline> airlineResponse = airlineRepo.findById("DAL");
59          System.out.println(airlineResponse.get());
60
61          System.out.println();
62
63          // fetch all aircraft
64          System.out.println("Aircraft found with findAll():");
65          System.out.println("-------------------------------");
66          for (Aircraft aircraft : aircraftRepo.findAll()) {
67              System.out.println(aircraft);
68          }
69          // fetch a specific aircraft by ICAO ID
70          System.out.println("Aircraft found with findById():");
71          System.out.println("-------------------------------");
72          Optional<Aircraft> aircraftResponse = aircraftRepo.findById("DAL_a36f7e");
73          System.out.println(aircraftResponse.get());
74
75          System.out.println();
76
77          // fetch all adsb records
78          System.out.println("ADSB records found with findAll():");
79          System.out.println("-------------------------------");
80          for (ADSBRecord adsb : adsbRepo.findAll()) {
81              System.out.println(adsb);
82          }
83          // fetch a specific ADSB Record by ID
84          System.out.println("ADSB Record found with findById():");
```

```
        System.out.println("--------------------------------");
        Optional<ADSBRecord> adsbResponse = adsbRepo.findById("DAL_a36f7e_1");
        System.out.println(adsbResponse.get());
        System.out.println();

    }

}
```

Spring Boot takes care of a lot of details in the background for us, including establishing a connection to MongoDB and autowiring our repository classes. On running the application, we are:

1. Using the save method on the Airline, Aircraft, and ADSBRecord repositories respectively to add an airline, three aircraft, and three ADSB position report documents to our collection.
2. Using the findAll and findById methods on the Airline, Aircraft, and ADSBRecord repositories respectively to retrieve, in turn, all airline documents, a specific airline document, all aircraft documents, a specific aircraft document, all ADSB position report documents, and a specific ADSB position report document.

If everything is configured correctly, we should see the following output on the command line:

```
Airlines found with findAll():
-------------------------------
Airline[id=DAL, name='Delta Air Lines', country='United States (US)', callsign='DELTA', website='delta.com']
Airline[id=DAL_a93d7c, name='null', country='null (null)', callsign='null', website='null']
Airline[id=DAL_ab8379, name='null', country='null (null)', callsign='null', website='null']
Airline[id=DAL_a36f7e, name='null', country='null (null)', callsign='null', website='null']
Airline[id=DAL_a36f7e_1, name='null', country='null (null)', callsign='null', website='null']
Airline[id=DAL_a36f7e_2, name='null', country='null (null)', callsign='null', website='null']
```

```
Airline[id=DAL_a36f7e_3, name='null', country='null (null)', callsign='null', website='null']
Airline found with findById():
--------------------------------
Airline[id=DAL, name='Delta Air Lines', country='United States (US)', callsign='DELTA', website='delta.com']

Aircraft found with findAll():
--------------------------------
Aircraft[id=DAL, tailNumber='null', manufacturer='null', model='null']
Aircraft[id=DAL_a93d7c, tailNumber='N695CA', manufacturer='Bombardier Inc', model='CL-600-2D24']
Aircraft[id=DAL_ab8379, tailNumber='N8409N', manufacturer='Bombardier Inc', model='CL-600-2B19']
Aircraft[id=DAL_a36f7e, tailNumber='N8409N', manufacturer='Airbus Industrie', model='A319-114']
Aircraft[id=DAL_a36f7e_1, tailNumber='null', manufacturer='null', model='null']
Aircraft[id=DAL_a36f7e_2, tailNumber='null', manufacturer='null', model='null']
Aircraft[id=DAL_a36f7e_3, tailNumber='null', manufacturer='null', model='null']
Aircraft found with findById():
--------------------------------
Aircraft[id=DAL_a36f7e, tailNumber='N8409N', manufacturer='Airbus Industrie', model='A319-114']

ADSB records found with findAll():
--------------------------------
ADSB[id=DAL, altitude='null', heading='null', speed='null', verticalSpeed='null' timestamp='null', latitude='null', l
ADSB[id=DAL_a93d7c, altitude='null', heading='null', speed='null', verticalSpeed='null' timestamp='null', latitude='n
ADSB[id=DAL_ab8379, altitude='null', heading='null', speed='null', verticalSpeed='null' timestamp='null', latitude='n
ADSB[id=DAL_a36f7e, altitude='null', heading='null', speed='null', verticalSpeed='null' timestamp='null', latitude='n
ADSB[id=DAL_a36f7e_1, altitude='38825', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_2, altitude='38875', heading='319', speed='429', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_3, altitude='38892', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB Record found with findById():
--------------------------------
ADSB[id=DAL_a36f7e_1, altitude='38825', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
```

As you can see, our data has been successfully added to the MongoDB collection, and we are able to retrieve the data. However, there is a problem. The findAll methods of each of the repository objects are returning a result for

every document in our collection, not just the documents of the class type associated with each repository. As a result, we are seeing seven documents being returned for each record type – airline, aircraft, and ADSB – when we would expect to see only one airline, three aircraft, and three ADSB position reports. Note this issue is common across all the "All" repository methods – findAll, deleteAll, and notifyAll. A call to the deleteAll method on the airline repository would result in all documents in the collection being deleted, not just airline documents.

To address this, we have two options: We could override the standard Spring Boot repository findAll (and deleteAll/notifyAll) methods to factor in the class associated with each calling repository class, or we could extend the repository interface definitions to include methods to specifically retrieve only documents of the corresponding class. In our exercise, we'll concentrate on the later approach by updating our repository interface definitions:

```
1  package com.mongodb.devrel.gcr.aerodata;
2
3  import java.util.List;
4  import java.util.Optional;
5  import org.springframework.data.mongodb.repository.MongoRepository;
6  import org.springframework.data.mongodb.repository.Query;
7
8  public interface AirlineRepository extends MongoRepository<Airline, String>{
9
10     @Query("{_class: \"AirlineData\"}")
11     List<Airline> findAllAirlines();
12
13     @Query(value="{_id: /^?0/, _class: \"AirlineData\"}", sort = "{_id: 1}")
14     Optional<Airline> findAirlineByIcaoAddr(String icaoAddr);
15
16 }
```

```
1  package com.mongodb.devrel.gcr.aerodata;
```

```
2
3  import java.util.List;
4  import org.springframework.data.mongodb.repository.MongoRepository;
5  import org.springframework.data.mongodb.repository.Query;
6
7  public interface AircraftRepository extends MongoRepository<Aircraft, String>{
8
9      @Query("{_class: \"AircraftData\"}")
10     List<Aircraft> findAllAircraft();
11
12     @Query("{_id: /^?0/, _class: \"AircraftData\"}")
13     List<Aircraft> findAircraftDataByIcaoAddr(String icaoAddr);
14
15 }
```

```
1  package com.mongodb.devrel.gcr.aerodata;
2
3  import java.util.List;
4  import org.springframework.data.mongodb.repository.MongoRepository;
5  import org.springframework.data.mongodb.repository.Query;
6
7  public interface ADSBRecordRepository extends MongoRepository<ADSBRecord, String>{
8
9      @Query(value="{_class: \"ADSBRecord\"}",sort="{_id: 1}")
10     List<ADSBRecord> findAllADSBRecords();
11
12     @Query(value="{_id: /^?0/, _class: \"ADSBRecord\"}", sort = "{_id: 1}")
13     List<ADSBRecord> findADSBDataByIcaoAddr(String icaoAddr);
14
15 }
```

In each interface, we've added two new function definitions – one to return all documents of the relevant type, and one to allow documents to be returned when searching by ICAO address. Using the @Query annotation, we are able to

format the queries as needed.

With our function definitions in place, we can now update the main application class:

```
1  package com.mongodb.devrel.gcr.aerodata;
2
3  import java.util.Date;
4  import java.util.Optional;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.boot.CommandLineRunner;
7  import org.springframework.boot.SpringApplication;
8  import org.springframework.boot.autoconfigure.SpringBootApplication;
9
10 @SpringBootApplication
11 public class AerodataApplication implements CommandLineRunner {
12
13     @Autowired
14     private AirlineRepository airlineRepo;
15
16     @Autowired
17     private AircraftRepository aircraftRepo;
18
19     @Autowired
20     private ADSBRecordRepository adsbRepo;
21
22     public static void main(String[] args) {
23         SpringApplication.run(AerodataApplication.class, args);
24     }
25
26     @Override
27     public void run(String... args) throws Exception {
28
29         //Delete any records from a previous run;
30         airlineRepo.deleteAll();
```

```
31
32          // save an airline
33          airlineRepo.save(new Airline("DAL", "Delta Air Lines", "United States", "US", "DELTA", "delta.com"));
34
35          // add some aircraft aircraft
36          aircraftRepo.save(new Aircraft("DAL_a93d7c", "N695CA", "Bombardier Inc", "CL-600-2D24"));
37          aircraftRepo.save(new Aircraft("DAL_ab8379", "N8409N", "Bombardier Inc", "CL-600-2B19"));
38          aircraftRepo.save(new Aircraft("DAL_a36f7e", "N8409N", "Airbus Industrie", "A319-114"));
39
40          //Add some ADSB position reports
41          Double[] coords1 = {-4.776722, 55.991776};
42          GeoPoint geoPoint = new GeoPoint(coords1);
43          adsbRepo.save(new ADSBRecord("DAL_a36f7e_1", 38825, 319, 428, 1024, new Date(16569806170411), geoPoint));
44          Double[] coords2 = {-4.781466, 55.994843};
45          geoPoint = new GeoPoint(coords2);
46          adsbRepo.save(new ADSBRecord("DAL_a36f7e_2", 38875, 319, 429, 1024, new Date(16569806180411), geoPoint));
47          Double[] coords3 = {-4.783344, 55.99606};
48          geoPoint = new GeoPoint(coords3);
49          adsbRepo.save(new ADSBRecord("DAL_a36f7e_3", 38892, 319, 428, 1024, new Date(16569806190411), geoPoint));
50
51
52          // fetch all airlines
53          System.out.println("Airlines found with findAllAirlines():");
54          System.out.println("-------------------------------");
55          for (Airline airline : airlineRepo.findAllAirlines()) {
56              System.out.println(airline);
57          }
58          System.out.println();
59          // fetch a specific airline by ICAO ID
60          System.out.println("Airlines found with findAirlineByIcaoAddr(\"DAL\"):");
61          System.out.println("-------------------------------");
62          Optional<Airline> airlineResponse = airlineRepo.findAirlineByIcaoAddr("DAL");
63          System.out.println(airlineResponse.get());
64
65          System.out.println();
```

```
66
67          // fetch all aircraft
68          System.out.println("Aircraft found with findAllAircraft():");
69          System.out.println("-------------------------------");
70          for (Aircraft aircraft : aircraftRepo.findAllAircraft()) {
71              System.out.println(aircraft);
72          }
73          System.out.println();
74          // fetch Aircraft Documents specific to airline "DAL"
75          System.out.println("Aircraft found with findAircraftDataByIcaoAddr(\"DAL\"):");
76          System.out.println("-------------------------------");
77          for (Aircraft aircraft : aircraftRepo.findAircraftDataByIcaoAddr("DAL")) {
78              System.out.println(aircraft);
79          }
80
81          System.out.println();
82
83          // fetch Aircraft Documents specific to aircraft "a36f7e"
84          System.out.println("Aircraft found with findAircraftDataByIcaoAddr(\"DAL_a36f7e\"):");
85          System.out.println("-------------------------------");
86          for (Aircraft aircraft : aircraftRepo.findAircraftDataByIcaoAddr("DAL_a36f7e")) {
87              System.out.println(aircraft);
88          }
89
90          System.out.println();
91
92          // fetch all adsb records
93          System.out.println("ADSB records found with findAllADSBRecords():");
94          System.out.println("-------------------------------");
95          for (ADSBRecord adsb : adsbRepo.findAllADSBRecords()) {
96              System.out.println(adsb);
97          }
98          System.out.println();
99          // fetch ADSB Documents specific to airline "DAL"
100         System.out.println("ADSB Documents found with findADSBDataByIcaoAddr(\"DAL\"):");
```

```
        System.out.println("--------------------------------");
        for (ADSBRecord adsb : adsbRepo.findADSBDataByIcaoAddr("DAL")) {
            System.out.println(adsb);
        }

        System.out.println();

        // fetch ADSB Documents specific to aircraft "a36f7e"
        System.out.println("ADSB Documents found with findADSBDataByIcaoAddr(\"DAL_a36f7e\"):");
        System.out.println("--------------------------------");
        for (ADSBRecord adsb : adsbRepo.findADSBDataByIcaoAddr("DAL_a36f7e")) {
            System.out.println(adsb);
        }
    }
}
```

Note that as well as the revised search calls, we also added a call to deleteAll on the airline repository to remove data added by prior runs of the application.

With the updates in place, when we run the application, we should now see the expected output:

```
Airlines found with findAllAirlines():
-------------------------------
Airline[id=DAL, name='Delta Air Lines', country='United States (US)', callsign='DELTA', website='delta.com']

Airlines found with findAirlineByIcaoAddr("DAL"):
-------------------------------
Airline[id=DAL, name='Delta Air Lines', country='United States (US)', callsign='DELTA', website='delta.com']

Aircraft found with findAllAircraft():
-------------------------------
Aircraft[id=DAL_a93d7c, tailNumber='N695CA', manufacturer='Bombardier Inc', model='CL-600-2D24']
Aircraft[id=DAL_ab8379, tailNumber='N8409N', manufacturer='Bombardier Inc', model='CL-600-2B19']
```

```
Aircraft[id=DAL_a36f7e, tailNumber='N8409N', manufacturer='Airbus Industrie', model='A319-114']

Aircraft found with findAircraftDataByIcaoAddr("DAL"):
------------------------------
Aircraft[id=DAL_a36f7e, tailNumber='N8409N', manufacturer='Airbus Industrie', model='A319-114']
Aircraft[id=DAL_a93d7c, tailNumber='N695CA', manufacturer='Bombardier Inc', model='CL-600-2D24']
Aircraft[id=DAL_ab8379, tailNumber='N8409N', manufacturer='Bombardier Inc', model='CL-600-2B19']

Aircraft found with findAircraftDataByIcaoAddr("DAL_a36f7e"):
------------------------------
Aircraft[id=DAL_a36f7e, tailNumber='N8409N', manufacturer='Airbus Industrie', model='A319-114']

ADSB records found with findAllADSBRecords():
------------------------------
ADSB[id=DAL_a36f7e_1, altitude='38825', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_2, altitude='38875', heading='319', speed='429', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_3, altitude='38892', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:

ADSB Documents found with findADSBDataByIcaoAddr("DAL"):
------------------------------
ADSB[id=DAL_a36f7e_1, altitude='38825', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_2, altitude='38875', heading='319', speed='429', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_3, altitude='38892', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:

ADSB Documents found with findADSBDataByIcaoAddr("DAL_a36f7e"):
------------------------------
ADSB[id=DAL_a36f7e_1, altitude='38825', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_2, altitude='38875', heading='319', speed='429', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
ADSB[id=DAL_a36f7e_3, altitude='38892', heading='319', speed='428', verticalSpeed='1024' timestamp='Tue Jul 05 01:23:
```

In this two-part post, we have seen how polymorphic single-collection designs in MongoDB can provide all the query flexibility of normalized relational designs, whilst simultaneously avoiding anti-patterns such as unbounded arrays and

unnecessary joins. This makes the resulting collections highly performant from a search standpoint and amenable to horizontal scaling. We have also shown how we can work with these designs using Spring Data MongoDB.

The example source code used in this series is available in Github.

Rate this tutorial

Related

TUTORIAL

Implementing Bulk Writes using Spring Boot for MongoDB

Mar 22, 2023 | 3 min read

TUTORIAL

Building invincible applications with Temporal and MongoDB

Jan 27, 2025 | 26 min read

ARTICLE

How to Optimize Java Performance With Virtual Threads, Reactive Programming, and MongoDB

Aug 29, 2024 | 5 min read

QUICKSTART

Building Quarkus Application with MongoDB and Panache

Dec 03, 2024 | 5 min read

Request a Tutorial

English

About

Careers

Investor Relations

Legal

GitHub

Security Information

Trust Center

Connect with Us

Support

Contact Us

Customer Portal

Atlas Status

Customer Support

Manage Cookies

Deployment Options

MongoDB Atlas

Data Basics

Vector Databases

Enterprise Advanced

Community Edition

NoSQL Databases

Document Databases

RAG Database

ACID Transactions

MERN Stack

MEAN Stack