


Follow @only2dhir


Spring Data MongoDB Queries


f





By [Dhiraj](#), 15 July, 2019


👁49K











Spring Data MongoDB

In this article, we will learn about [Spring data MongoDB queries](#) with different ways to create and execute [MongoDB queries](#) in a spring boot project. We will learn to create various custom dynamic queries with multiple MongoDB operators to get started. For this, we will have a spring boot MongoDB application setup and create sample examples of all our learnings.

Recommended

Spring Boot Mongoddb Crud
Spring Boot Mongoddb Configuration
Spring Hibernate Integration Example Javaconfig
Hibernate Annotations Example
Hibernate Inheritance Example
Hibernate Criteria Tutorial
Spring Data Jpa Example

Primarily, there are 4 ways with which we can query MongoDB from a spring boot application. They are:

- Auto Generated Methods with [Mongo Repositories](#)
- Using [@Query Annotation](#)
- Using Criteria Query with [MongoTemplate](#)
- Using [QueryDSL](#)

We will learn all these methods to create and run queries with examples.

To get started with the examples, you need to have a Spring Boot MongoDB configured app running on your local machine. You can follow this article for [Spring Boot MongoDB Configuration](#).

Auto Generated Methods with Mongo Repositories

To get started with the default and generic methods defined in Spring Data Repositories, we need to have a model class and a repository class that extends MongoRepository. For this example, we have an Employee model class.



```
@Document("Employee")
public class Employee {

    @Id
    private String id;
    @Indexed
    private String name;
    private String description;
    private double salary;
    private String band;
    private String position;

    //setters and getters
}
```

Now let us create some entries in our DB.

```
@Bean
CommandLineRunner init(EmployeeRepository employeeRepository){
    return args -> {
        List employees = Stream.of(1, 2, 3, 4).map(count -> {
            Employee employee = new Employee();
            employee.setName("Emp " + count);
            employee.setDescription("Emp " + 1 + " Description");
            employee.setBand("E " + count);
            employee.setPosition("POS " + count);
            employee.setSalary(4563 + (count * 10 + 45 + count));
            return employee;
        }).collect(Collectors.toList());

        employeeRepository.saveAll(employees);
    };
}
```

Now, to query upon an employee object, we can use different default methods provided by `MongoRepository` as below. Remember, these are default methods and hence no need to provide the definition in our `EmployeeRepository` class.

```
@Repository
public interface EmployeeRepository extends MongoRepository {

}
```

With just above definition, we can use below methods directly from our service class to perform queries on an `Employee` object.



```
List findAll(Sort sort);

Optional findById(String empId);

Employee insert(Employee employee);

List insert(List employees);

List findAll(Example empExample);

List findAll(Example example, Sort sort);

void delete(Employee emp);

void deleteById(String id);

void deleteAll(List emps)
```

As you can see, all the methods required for the CRUD operations are already provided by `MongoRepository`.

The `insert()` method inserts a given new entity and returns the inserted entity. But, we mostly use `save()` method for insertion and update both. In this case, if an Id is already present in the given document then the document is updated else a new document is created.

Now, let us create some examples for above methods.

Sorting with MongoRepository

As you can see the `findAll(Sort sort)` method in the above definitions that returns a list of sorted document. This is a very useful method as the sorting becomes configurable in the server side with this implementation and we don't have to introduce multiple variables to keep a record of no of a total count of documents, no of pages, etc with the methods - `getTotalPages()`, `getTotalElements()`. This method requires an instance of `Sort` class and there are 3 different methods to get an instance of it.

```
PageRequest firstPageRequest = PageRequest.of(0, 3); //0 is the page index whereas 3 is the size of each page
PageRequest secondPageRequest = PageRequest.of(0, 3, Sort.by("salary")); //returns sorted result by salary in ascending order
PageRequest thirdPageRequest = PageRequest.of(0, 3, Sort.Direction.DESC, "salary"); //returns sorted result by salary in descending order
```

```
logger.info(employeeRepository.findAll(firstPageRequest).get().toString());
logger.info(employeeRepository.findAll(secondPageRequest).getContent().toString());
logger.info(employeeRepository.findAll(thirdPageRequest).getContent().toString());
```

Query By Example in MongoRepository

match properties. Below is an example:

```
Example employeeExample = Example.of(new Employee("Emp 1"));
employeeRepository.findAll(employeeExample);

ExampleMatcher matcher = ExampleMatcher.matching()
    .withMatcher("name", startsWith().ignoreCase());

employeeRepository.findAll(Example.of(new Employee("Emp 1"), matcher));
```



Query by Object Attribute in MongoDB

All the examples above are the default and implicit methods that we get while using spring data mongo repository. To use any of the example above, you do not have to define any methods in the repository class. Now, let us see how we can define methods in the repository class to execute our custom queries to query MongoDB by our document attributes.

```
@Repository
public interface EmployeeRepository extends MongoRepository {

    List findByNameAndBand(String name, String band);

}
```

This method finds employee by employee name and band.

```
List findByNameAndBandOrderBySalary(String name, String band);
```


Above method finds employee by employee name and band and returns ordered result by salary. There are lots of keywords that are supported here. A complete list of supported keywords for query methods can be found on the [official website](#).


Using @Query Annotation

@Query Annotation can be used to specify a MongoDB JSON query string. With this annotation, you can execute all the complex queries and it is also sometimes very useful to execute native MongoDB queries. We have all the freedom to take advantage of MongoDB native operators and operations.

Below is a sample example to find an employee by name using @Query annotation.

```
@Query("{\"name\" : ?0}")
Employee findEmployeeByName(String empName);
```

 By default, all the properties of Employee document will be fetched. Now, if you want to ignore some of the properties in the result, you can use filter option.

 Here, the description attribute will be ignored.



```
@Query(value = "{ 'name' : ?0 }", fields = "{ 'description' : 0 }")  
Employee findEmployeeByName(String empName);
```

And Operator

Let us see how we can use MongoDB specific operator using @Query annotation. Below is an example of AND operator.

```
@Query("{ 'name' : ?0 , 'band' : ?1 }")  
List findEmployeeByNameAndBand(String name, String band);
```

Nested Queries

Suppose you have an embedded collection. Let us say Department and Employee collections where each department can have multiple employees like our last article on [spring boot MongoDB](#). In this case, we require to execute some nested queries.

Below query returns department collection whose employee's name is empName and salary greater than salary.

```
@Query(value = "{ 'employees.name': ?0, 'employees.salary' : { $gt: ?1 } }", fields = "  
{ 'employees' : 0 }")  
Department findDepartmentByEmployeeNameAndSalary(String empName, Double salary);
```






\$lt and \$gt

These are most frequently used operators and hence let us execute some queries using these operators.

Below query returns employee collection whose salary is greater than minSalary and less than maxSalary. With comma(), it internally applies AND operator.

```
@Query("{ salary : { $lt : ?0, $gt : ?1 } }")  
List findEmployeeBySalaryRange(double maxSalary, double minSalary);
```

The result of the complex queries that you execute using @Query annotation can also be sorted in MongoRepository. Below is an example that finds employee salary in a range and returns sorted result by employee salary in descending order.



```
@Query("{salary : {$lt : ?0, $gt : ?1}}")
List findEmployeeBySalaryRangeAndSort(double maxSalary, double minSalary, Sort sort);
```

Regex using \$regex

This operator can be used to query with regular expression parameters. Below query finds employee whose name starts with "Em".

```
@Query("{ 'name' : { $regex: ?0 } }")
List findByRegex(String regexp);
```

```
employeeRepository.findByRegex("^Em")
```

Below are some of other examples which would be useful.

```
@Query("{name : {$ne : ?0}}")
List findByNameNotEqual(String countryName);

@Query("{ 'name' : null}")
List findEmployeeByNameIsNull();

@Query("{ 'name' : {$ne : null}}")
List findEmployeeByNameIsNotNull();
```

Using Criteria Query with MongoTemplate

Once, we have our MongoDB connection property defined in the application.properties file, spring automatically creates an instance of MongoTemplate and we can directly inject it in our repo class and execute criteria queries using MongoTemplate.

Below are some of the examples of using criteria query using MongoTemplate.

Save Operation

We can directly use `save()` method to perform this operation.

Update Operation



For update, we first fetch an existing document using **Is** operator in the Query and then save the document after updating the different attributes.

```
public Employee update(Employee employee){
    Query query = new Query();
    query.addCriteria(Criteria.where("id").is(employee.getId()));
    Update update = new Update();
    update.set("name", employee.getName());
    update.set("description", employee.getDescription());
    update.set("salary", employee.getSalary());
    return mongoTemplate.findAndModify(query, update, Employee.class);
}
```

Delete Operation

To perform delete operation, we use **remove()** method of MongoTemplate

```
public void deleteById(String empId) {
    Query query = new Query();
    query.addCriteria(Criteria.where("id").is(empId));
    mongoTemplate.remove(query, Employee.class);
}
```

It and gt Operator

Below is the criteria query that queries the DB for employees whose salary in between a range of max and min.

```
public List findEmployeeBySalaryRange(double minSalary, double maxSalary) {
    Query query = new Query();
    query.addCriteria(Criteria.where("salary").lt(maxSalary).gt(minSalary));
    return mongoTemplate.find(query, Employee.class);
}
```

All the other operators can be used in a similar way.

Pageable Using MongoTemplate

We already discussed the different out of the box methods which can be used to create an instance of Pageable. Once, those instances are created, they can be used with MongoTemplate to query the DB.



```
public Page findEmployeeByPage() {
    Pageable pageRequest = PageRequest.of(0, 3, Sort.Direction.DESC, "salary");
    Query query = new Query();
    query.with(pageRequest);
    List list = mongoTemplate.find(query, Employee.class);
    return PageableExecutionUtils.getPage(
        list,
        pageRequest,
        () -> mongoTemplate.count(query, Employee.class));
}
```

Projections with MongoTemplate

As we saw in the above examples in our MongoRepository implementation, we used fields attribute to project only the columns which we are interested in. The same result can be achieved with MongoTemplate too and this is achieved with the **include** and **exclude** operators. Below is an example:

```
public List findSalary(){
    Query query = new Query();
    query.fields().exclude("position").exclude("band");
    query.addCriteria(Criteria.where("salary").lt(898888).gt(113));
    return mongoTemplate.find(query, Employee.class);
}
```

In the above example, the position and band will default to its default value of null.

Aggregation with Mongotemplate

MongoDB provides multiple native operators to perform aggregation \$group, \$order, \$sort, etc. In spring data mongo, we can perform these operations with different aggregate functions. In Spring Data, we perform these aggregation with mainly 3 classes – Aggregation, AggregationOperation and AggregationResults.

[MongoDB's aggregation](#) framework is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.

Below is an example of aggregation using different aggregate operations:


```
GroupOperation groupByBand = group("band")
    .sum("salary").as("salary");

//$sort
SortOperation sort = sort(new Sort(Sort.Direction.DESC, "salary"));
```



```
Aggregation aggregation = newAggregation(
    groupByBand, salaryMatch, sort);

AggregationResults result = mongoTemplate.aggregate(
    aggregation, "result", Employee.class);
```

Using QueryDSL

MongoDB repository support integrates with the QueryDSL project which provides a means to perform type-safe queries in Java. Once the Query DSL is configured in the project, we can have Code completion in intelligence in IDE and we can easily write queries with almost no syntactically invalid queries.

We will discuss the details about setting up this Query DSL in our next article as we are only looking into ways of executing queries in this article. The different maven dependencies required for this can be downloaded from [here](#) and [here](#).

Below are some of the queries example using DSL.

```
QEmployee qEmployee = new QEmployee("employee");
Predicate predicate = qEmployee.name.eq("Emp 1");
employeeRepository.findAll(predicate);

employeeRepository.findAll(qEmployee.salary.between(1234, 787654));

employeeRepository.findAll(qEmployee.name.contains("p"), new PageRequest(0, 2, Sort.Direction.ASC, "salary"));
```

QEmployee is a class that is generated (via the Java annotation post-processing tool) which is a Predicate that allows you to write type-safe queries. Notice that there are no strings in the query other than the query parameters that we supply.

QEmployee.java

f

t

p

in

t

```
public static final QEmployee employee = new QEmployee("employee");

public final StringPath band = createString("band");

public final StringPath description = createString("description");

public final StringPath id = createString("id");

public final StringPath name = createString("name");

public final StringPath position = createString("position");

public final NumberPath salary = createNumber("salary", Double.class);

public QEmployee(String variable) {
    super(Employee.class, forVariable(variable));
}

public QEmployee(Path path) {
    super(path.getType(), path.getMetadata());
}

public QEmployee(PathMetadata metadata) {
    super(Employee.class, metadata);
}



}
```

Conclusion

In this tutorial, we learned about spring data queries and learn different ways to create and execute MongoDB queries with spring data using MongoRepository, @Query annotation, criteria query with mongo template and Query DSL.


If You Appreciate This, You Can Consider:

Donate

Like us at:  or follow us at 

Share this article on social media or with your teammates.

About The Author



A technology savvy professional with an exceptional capacity to analyze, solve problems and multi-task. Technical expertise in highly scalable distributed systems, self-healing systems, and service-oriented architecture. Technical Skills: Java/J2EE, Spring, Hibernate, Reactive Programming, Microservices, Hystrix, Rest APIs, Java 8, Kafka, Kibana, Elasticsearch, etc.

Further Reading on Spring Boot

- 1. [Spring Boot MongoDB Crud](#)
- 2. [Spring Boot MongoDB Configuration](#)
- 3. [Spring Hibernate Integration Example Javaconfig](#)

6. Hibernate Criteria Tutorial

7. Spring Data Jpa Example

f

p

in

t

ALSO ON DEVGLAN

9 months ago • 2 comments

Uploading and Downloading Files ...

9 months ago • 4 comments

Spring Boot Websocket Example

a ye

Ty

Tu

Be

What do you think?

1 Response

👍 Upvote

😂 Funny

❤️ Love

😮 Surprised

😡 Angry

😞 Sad

Comments

Community

🔒 Privacy Policy

1

Login ▾

❤️ Recommend

🐦 Tweet

📌 Share

Sort by Best ▾