**MongoDb**

Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

MongoDB is an open-source document database that provides high performance, high **availability, and automatic scaling.**

The manufacturing company 10 gen has given the definition of Mongo DB:

"**Mongo DB is scalable, open source, high performance, document-oriented database**." - 10 gen

**Purpose to build MongoDB**

All the modern applications require big data, fast features development, flexible deployment and the older database systems not enough competent, so the MongoDB was obviously needed.

**Main purpose to build MongoDB:**

* Scalability
* Performance
* High Availability
* Scaling from single server deployments to large, complex multi-site architectures.

Key points of MongoDB

* Develop Faster
* Deploy Easier
* Scale Bigger

Features of MongoDB:

These are some important features of MongoDB:

**1. Support ad hoc queries**

In MongoDB, you can search by field, range query and it also supports regular expression searches.

**2. Indexing**

 Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed

**3. Replication**

MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

**4. Duplication of data**

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

**5. Load balancing**

It has an automatic load balancing configuration because of data placed in shards.

**6. Supports map reduce and aggregation tools.**

**7. Uses JavaScript instead of Procedures.**

**8. It is a schema-less database written in C++.**

**9. Provides high performance.**

**10. Stores files of any size easily without complicating your stack.**

**11. Easy to administer in the case of failures.**

**12. It also supports:**

* JSON data model with dynamic schemas
* Auto-sharding for horizontal scalability
* Built in replication for high availability

Now a day many companies using MongoDB to create new types of applications, improve performance and availability.

What is NoSQL Database

Databases can be divided in 3 types:

1. RDBMS (Relational Database Management System)
2. OLAP (Online Analytical Processing)
3. NoSQL (recently developed database)

NoSQL Database

NoSQL Database is used to refer a non-SQL or non relational database.

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

## Advantages of NoSQL

* It supports query language.
* It provides fast performance.
* It provides horizontal scalability.

Advantages of mongo DB:

MongoDB is a new and popularly used database. It is a document based, non relational database provider.

Although it is 100 times faster than the traditional database but it is early to say that it will broadly replace the traditional RDBMS. But it may be very useful in term to gain performance and scalability.

A Relational database has a typical schema design that shows number of tables and the relationship between these tables, while in MongoDB there is no concept of relationship.

## MongoDB Advantages

* **MongoDB is schema less**. It is a document database in which one collection holds different documents.
* There may be **difference between number of fields, content and size of the document** from one to other.
* **Structure of a single object is clear** in MongoDB.
* There are **no complex joins** in MongoDB.
* MongoDB provides the **facility of deep query** because it supports a powerful dynamic query on documents.
* It is very **easy to scale**.
* It **uses internal memory for storing working sets** and this is the reason of its fast access.

## Distinctive features of MongoDB

* Easy to use
* Light Weight
* Extremely faster than RDBMS

## Where MongoDB should be used

* Big and complex data
* Mobile and social infrastructure
* Content management and delivery
* User data management
* Data hub

## Performance analysis of MongoDB and RDBMS

* In relational database (RDBMS) tables are using as storing elements, while in MongoDB collection is used.
* In the RDBMS, we have multiple schema and in each schema we create tables to store data while, MongoDB is a document oriented database in which data is written in BSON format which is a JSON like format.
* MongoDB is almost 100 times faster than traditional database systems.

Schema:

A schema is a collection of database objects (as far as this hour is concerned—tables) associated with one particular database username. This username is called the schema owner, or the owner of the related group of objects. You may have one or multiple schemas in a database

Difference between MongoDB & RDBMS

Below are some of the key term differences between MongoDB and RDBMS

|  |  |  |
| --- | --- | --- |
| RDBMS | MongoDB | Difference |
| Table | Collection | In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs. |
| Row | Document | In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents. |
| Column | Field | In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields. |
| Joins | Embedded documents | In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in MongoDB. |

**Why NoSQL?**

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

## Binary JSON (BSON)

MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes. BSON extends the JSON model to provide additional data types, ordered fields, and to be efficient for encoding and decoding within different languages.

# MongoDB Datatypes

Following is a list of usable data types in MongoDB.

|  |  |
| --- | --- |
| **Data Types** | **Description** |
| String | String is the most commonly used datatype. It is used to store data. A string must be UTF 8 valid in mongodb. |
| Integer | Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using. |
| Boolean | This datatype is used to store boolean values. It just shows YES/NO values. |
| Double | Double datatype stores floating point values. |
| Min/Max Keys | This datatype compare a value against the lowest and highest bson elements. |
| Arrays | This datatype is used to store a list or multiple values into a single key. |
| Object | Object datatype is used for embedded documents. |
| Null | It is used to store null values. |
| Symbol | It is generally used for languages that use a specific type. |
| Date | This datatype stores the current date or time in unix time format. It makes you possible to specify your own date time by creating object of date and pass the value of date, month, year into it. |

MongoDb Structure follows:

Database -> Collection -> Documents

# **MongoDB Create Database**

**How and when to create database**

If there is no existing database, the following command is used to create a new database.

**Syntax:**

* use DATABASE\_NAME

If the database already exists, it will return the existing database.

Let' take an example to demonstrate how a database is created in MongoDB. In the following example, we are going to create a database "javatpointdb".

**See this example**

>use javatpointdb

Swithched to dbjavatpointdb

To **check the currently selected database**, use the command db:

>db

javatpointdb

To **check the database list**, use the command show dbs:

>show dbs

local 0.078GB

Here, your created database "javatpointdb" is not present in the list, **insert at least one document** into it to display database:

>db.movie.**insert**({"name":"javatpoint"})

WriteResult({ "nInserted": 1})

>show dbs

javatpointdb 0.078GB

local 0.078GB

# **MongoDB Drop Database**

The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database.

**Syntax:**

db.dropDatabase()

This syntax will delete the selected database. In the case you have not selected any database, it will delete default "test" database.

To **check the database list**, use the command show dbs:

>show dbs

javatpointdb 0.078GB

local 0.078GB

# **MongoDB Create Collection**

In MongoDB, db.createCollection(name, options) is used to create collection. But usually you don?t need to create collection. MongoDB creates collection automatically when you insert some documents. It will be explained later. First see how to create collection:

**Syntax:**

db.createCollection(**name**, options)

Here,

**Name:** is a string type, specifies the name of the collection to be created.

**Options:** is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

**How does MongoDB create collection automatically**

MongoDB creates collections automatically when you insert some documents. For example: Insert a document named seomount into a collection named SSSIT. The operation will create the collection if the collection does not currently exist.

db.SSSIT.**insert**({"name" : "seomount"})

show collections

SSSIT

**Update Operation**

In MongoDB, update() method is used to update or modify the existing documents of a collection.

**Syntax:**

db.COLLECTION\_NAME.**update**(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

**Example:**

db.student.update**({**"name"**:**"Nagendra"**},{**$set **:{**"name"**:**"SCALA"**}})**

**Remove Operation**

## Remove all documents

## Example: db.javatpoint.remove({})

## Remove all documents that match a condition

## Example:

db.student.remove**({**"name"**:**"SCALA"**})**

## Remove a single document that match a condition

If you want to remove a single document that match a specific condition, call the remove() method with justOne parameter set to true or 1.

The following example will remove a single document from the javatpoint collection where the type field is equal to programming language.

## db.javatpoint.remove( { type : "programming language" }, 1 )

## Limit And Sort

## db.student.find().limit(2)

## SQL: SELECT \* FROM userdetails LIMIT 2;

## If we want to fetch the two documents after the first document from the collection 'userdetails', the following mongodb command can be used

## db.student.find().skip(1).limit(3)

## Sort:

In MongoDB, sort() method is used to sort the documents in the collection. This method accepts a document containing list of fields along with their sorting order.

The sorting order is specified as 1 or -1.

* 1 is used for ascending order sorting.
* -1 is used for descending order sorting.

**Syntax:**

db.COLLECTION\_NAME.find().sort({**KEY**:1})

**Example:**db.student.find**()**.sort**({**"studNo"**:**1**})**

Note: By default sort() method displays the documents in ascending order. If you don't specify the sorting preference, it will display documents in ascending order.

**Projections**

Projections means selecting only necessary data rather than selecting whole of the data of document,

db.student.find**({},{**"name"**:**1**})**

**{**

"\_id"**:**ObjectId**(**"5c9daaf2f9211a39d316cf18"**),**

"name"**:**"Nag"

**}**

db.student.find**({},{**"name"**:**1**,**"age"**:**2**})**

**{**

"\_id"**:**ObjectId**(**"5c9daaf2f9211a39d316cf18"**),**

"name"**:**"Nag"**,**

"age"**:**"36"

**}**

**Removing the \_id element:**

db.student.find**({},{**"name"**:**1**,**"age"**:**2**,**"\_id"**:**0**})**

**{**

"name"**:**"Nag"**,**

"age"**:**"36"

**}**

// ----------------------------------------------

**{**

"name"**:**"indra"**,**

"age"**:**"37"

**}**

**Index**

Index is used to fetching records fast mannerin case millions of records

Creating Index for uniq field:

db.student.ensureIndex**({**"studNo"**:**1**})**

db.student.ensureIndex**({**"studNo"**:**1, “age”: -1**})**

db.student.dropIndex**({**"studNo"**:**1**})**

## The ensureIndex() Method

To create an index you need to use ensureIndex() method of MongoDB.

### Syntax

The basic syntax of ensureIndex() method is as follows().

>db.COLLECTION\_NAME.ensureIndex({KEY:1})

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

**Aggregation**

Now suppose we have a School database and have a Student Collection as below

Hide   Copy Code

db.Student.insert({StudentName : "Vijay",Section : "A",Marks:70,Subject:["Hindi","English","Math"]})

db.Student.insert({StudentName : "Gaurav",Section : "A",Marks:90,Subject:["English"]})

db.Student.insert({StudentName : "Ajay",Section : "A",Marks:70,Subject:["Math"]})

db.Student.insert({StudentName : "Ankur",Section : "B",Marks:10,Subject:["Hindi"]})

db.Student.insert({StudentName : "Sunil",Section : "B",Marks:70,Subject:["Math"]})

db.Student.insert({StudentName : "Preeti",Section : "C",Marks:80,Subject:["Hindi","English"]})

db.Student.insert({StudentName : "Anuj",Section : "C",Marks:50,Subject:["English"]})

db.Student.insert({StudentName : "Palka",Section : "D",Marks:40,Subject:["Math"]})

db.Student.insert({StudentName : "Soniya",Section : "D",Marks:20,Subject:["English","Math"]})

We will see different stages and how they works on this Student Collection.So Let's ready for some good stuff

**$match**

**$match** is similar to **Where**in SQL. In SQL we use Where to filter the data and same is here.If we need to pass only a subset of our data in next stage of Aggregation Pipeline then we use **$match**.**$match**filters the data and pass the matching data to the next stage of Pipeline.

**Example 1 :**Suppose we want to filter data based on **Section A** in Student Collection then we will use **$match**as below

Hide   Copy Code

db.Student.aggregate

(

[

{

"$match":

{

"Section":"A"

}

}

]

)

This will filter the data according to our $match and will pass only 3 rows to next Stage of pipeline where **Section is A.**

**Result**



**Example 2 :**Suppose if want to find out all the records where **Section is A** and **Marks**is greater then **80**

Hide   Copy Code

db.Student.aggregate (

[

   {

       $match:

       {

           $and:[{Section:'A'},{Marks: {"$gt" :80}}]

       }

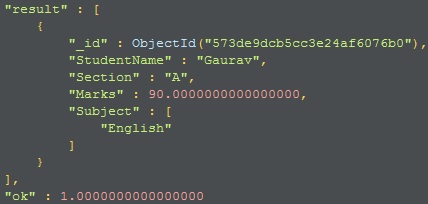
    }

 ]

)

This will give us one record

**Result**



**NOTE : There can be more than one $match in Aggregate Function.**

**$project** :

We can compare this clause with **SELECT**in SQL. We can select certain fields, rename Fields from documents though **$project**. In short **$project** reshape the documents by adding/removing or renaming the documents for the next stage of pipeline. In $project we use 1 or true if we want to include the Field and 0 or false if we want to exclude a particular field.

**Example 1 :** In the below query we want only StudentName,Section and Marks from student collection then we will use the below query

Hide   Copy Code

db.Student.aggregate

(

 [

  {

       "$project":{StudentName : 1,Section:1,Marks:1}

  }

 ]

)

**Example 2 :** Now if we want to find out **StudentName**,**Section**and **Marks**from Student Collection where **Section**is **'A'** than we will use $**project**and $**match**both

Hide   Copy Code

db.Student.aggregate

(

 [

  {

     "$match":

     {

        "Section":"A"

     }

  },

  {

       "$project":

     {

         StudentName : 1,Section:1,Marks:1

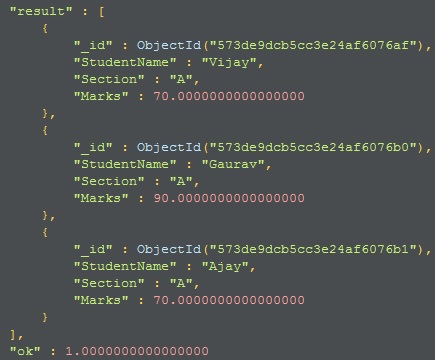
     }

  }

 ]

)

**Result**



**NOTE : \_id will be visible by default, if we don't want the \_id field in result then we need to remove it explicitly as below**

Hide   Copy Code

"$project":{StudentName : 1,Section:1,Marks:1,\_id:0}

**$group**

MongoDB use **$group** to group the documents by some specified expression.$group is similar to Group clause in SQL. Group in SQL is not possible without any Aggregate Function and the same is here. We can not group in MongoDB without Aggregate Functions. let's understand with an example

**Example 1** : Suppose we want to find out Total Marks group by Section then we will use $group as below

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         }

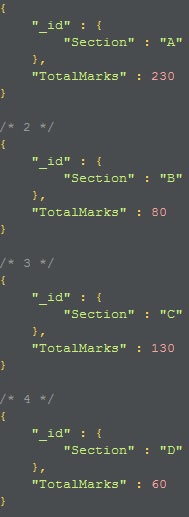
      }

   }

])

In this query **\_id**Field is mandatory. In **\_id** we pass the field on which we want to group the documents.This will give us below result

**Result**



**Example 2 :**If we want to fetch Total Marks for only Section 'A' then we can pass a **$match** also.

Hide   Copy Code

db.Student.aggregate ([

   {

       "$match":{Section :'A'}

   },

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         }

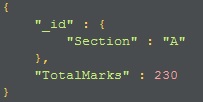
      }

   }

])

This will Sum the Total Marks of Section 'A' only.

**Result**



**Example 3** :

Suppose we want to fetch the count of students in each section and Total marks and average marks as well

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

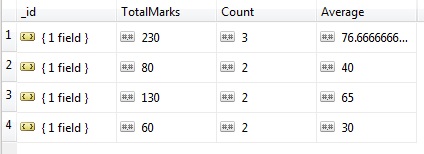
         "Average" : {"$avg" : "$Marks"}

      }

   }

])

**Result**



**Example 4 :**If we want to rename the column Names in above query(Section to SectionName and TotalMarls to Total)  then we can use **$project** along with **$group** as below

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

         "Average" : {"$avg" : "$Marks"}

      }

   },

   {

       "$project" :

       {

           "SectionName" : "$\_id.Section",

           "Total" : "$TotalMarks"

       }

   }

])

**$sort**

**$sort** is similar to  orderby clause in SQL server. In MongoDB we have **$sort** for this. **$sort** will sort the documents in either ascending or descending order as below. MongoDB uses 1 for ascending and -1 for descending

**Example 1 :**If we want to  sort the result in descending order by SectionName then we can use $sort

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

         "Average" : {"$avg" : "$Marks"}

      }

   },

   {

       "$project" :

       {

           "SectionName" : "$\_id.Section",

           "Total" : "$TotalMarks"

       }

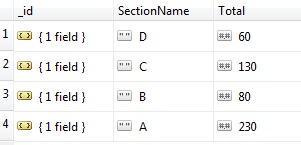
   },

   {

       "$sort":{"SectionName":-1}

   }

])



**https://www.codeproject.com/Articles/1096142/MongoDB-Tutorial-Day-Aggregation**

**Sql Aggregation functions:**

|  |
| --- |
| select dept max(sal) as MaxSalary from employee group By dept;  select dept min(sal) as MinSalary from employee group By dept;  select dept avg(sal) as AvearageSalary from employee group By dept;  select city sum(sal) as TotalSalary from employee group by city;  select city, gender, sum(sal) as TotalSalary count(ID) as TotalEmployess from employee  group by city, gender  order by city;  **Filter the rows with having cluse**  select city, gender, sum(sal) as TotalSalary count(ID) as TotalEmployess from employee  group by city, gender  having gender = 'male'  SELECT Student, SUM(score) AS total FROM Marks GROUP BY Student  HAVING total > 70  **Where cluse b/w having:**  **Where** caluse can be used with select,insert and update statements.  **Having** clause can only be used with the select statement.  **HAVING** specifies a search condition for a group or an aggregate function used in SELECT statement.  **HAVING**: is used to check conditions *after* the aggregation takes place. **WHERE**: is used to check conditions *before* the aggregation takes place. |

**commands:**

db.dropDatabase**()// dorp database**

db.createCollection**(**"myCollection"**) // create collection / table**

db.myCollections.insert**({**"name"**:**"Nag"**}) // create collection with document(row)**

db.myCollection.drop**() // drop collection**

show collections // show collections / tables

**insert document with collection:**

db.student.insert**(**

**{**"studNo"**:**"1"**,**

"name"**:**"Nagendra"**,**

"age"**:**"35"**,**

**})**

**Insert multiple document:**

|  |
| --- |
| db.student.insert**([**  **{**"studNo"**:**"1"**,**  "name"**:**"nag"**,**  "age"**:**"36"**,**  **},**  **{**"studNo"**:**"2"**,**  "name"**:**"nagendra"**,**  "age"**:**"37"**,**  **},**  **{**"studNo"**:**"3"**,**  "name"**:**"nm"**,**  "age"**:**"36"**,**  **},**  **{**"studNo"**:**"4"**,**  "name"**:**"mn"**,**  "age"**:**"37"**,**  **}**  **]**  **)** |

**Find query:**

|  |
| --- |
| db.student.find**()**.pretty**()**  **Output:**  **{**  "\_id"**:**ObjectId**(**"5c9daa64f9211a39d316cf17"**),**  "studNo"**:**"1"**,**  "name"**:**"Nagendra"**,**  "age"**:**"35"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9daaf2f9211a39d316cf18"**),**  "studNo"**:**"2"**,**  "name"**:**"Nag"**,**  "age"**:**"36"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9daaf2f9211a39d316cf19"**),**  "studNo"**:**"3"**,**  "name"**:**"indra"**,**  "age"**:**"37"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9dab2df9211a39d316cf1a"**),**  "studNo"**:**"10"**,**  "name"**:**"nm"**,**  "age"**:**"36"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9dab2df9211a39d316cf1b"**),**  "studNo"**:**"5"**,**  "name"**:**"mn"**,**  "age"**:**"37"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9dab51f9211a39d316cf1c"**),**  "studNo"**:**"10"**,**  "name"**:**"nm"**,**  "age"**:**"36"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9dab51f9211a39d316cf1d"**),**  "studNo"**:**"5"**,**  "name"**:**"mn"**,**  "age"**:**"37"  **}**  // ----------------------------------------------  **{**  "\_id"**:**ObjectId**(**"5c9dab98f9211a39d316cf1e"**),**  "studNo"**:**"1"**,**  "name"**:**"nag"**,**  "age"**:**"36"  }  ……..  db.student.findOne**() // find the first document/row**  **output:**  **{**  "\_id"**:**ObjectId**(**"5c9daa64f9211a39d316cf17"**),**  "studNo"**:**"1"**,**  "name"**:**"Nagendra"**,**  "age"**:**"35"  **}**  db.student.find**({**"studNo"**:**"2"**}) // based student no**  like sql:  select \* from student where studNo= “2”  db.student.find**({**"name"**:**"nagendra"**}) // based on name**  like sql:  select \* from student where name= “nagendra”  db.student.find**({**"studNo"**:{**$gt**:**"2"**}}) // select the data based age > 2**  like sql:  select \* from student where studNo**> 4**  db.student.find**({**"studNo"**:{**$gte**:**"2"**}})// select the data based age >= 2**  like sql:  select \* from student where studNo**>= 2**  db.student.find**({**"studNo"**:{**$lt**:**"4"**}})// select the data based age < 2**  like sql:  select \* from student where studNo**< 4**  db.student.find**({**"studNo"**:{**$lte**:**"4"**}}) // select the data based age <= 2**  like sql:  select \* from student where studNo**<= 4**  db.student.find**({**"studNo"**:{**$ne **:**"4"**}}) // select the data != 4**  like sql:  select \* from student where studNo **!= 4**  **AND operator:**  db.student.find**({**"name"**:**"indra"**,**"age"**:**"37"**}) // select data with and condition**  like sql:  select \* from student where name=”indra” and age = 37;  **OR Operator:**  db.student.find**({** $or**:[{**"name"**:**"indra"**},{**"age"**:**"37"**}]})**  like sql:  select \* from student where name=”indra” or age = 37;  **Both and/or operator:**  db.student.find**({**"name"**:**"mn"**,** $or**:[{**"age"**:**"36"**},{**"age"**:**"37"**}]})**  like sql:  select \* from student where name=” indra” or age = 36 or age = 37; |

**Spring Boot with Mongo dB operations**

|  |
| --- |
| **application.properties:**  spring.data.mongodb.host=localhost  spring.data.mongodb.port=27017  spring.data.mongodb.database=app1  **Model class**  @Document(collection = "person")  public class Person {  @Id  private String personId;  private String name;  private long age;  private List<String> favoriteBooks;  private Date dateOfBirth;    ----------  ----------  }  **Data Access Layer**  public interface PersonDAL {  Person savePerson(Person person);  List<Person> getAllPerson();  List<Person> getAllPersonPaginated(  int pageNumber, int pageSize);  Person findOneByName(String name);  List<Person> findByName(String name);  List<Person> findByBirthDateAfter(Date date);  List<Person> findByAgeRange(int lowerBound, int upperBound);  List<Person> findByFavoriteBooks(String favoriteBook);  void updateMultiplePersonAge();  Person updateOnePerson(Person person);  void deletePerson(Person person);  }  @Repository  **public** **class** PersonDALImpl **implements** PersonDAL {  @Autowired  **private** MongoTemplate mongoTemplate;    @Override  **public** Person savePerson(Person person) {  mongoTemplate.save(person);  **return** person;  }    @Override  **public** List<Person> getAllPerson() {  **return** mongoTemplate.findAll(Person.**class**);  }    @Override  **public** List<Person> getAllPersonPaginated(**int** pageNumber, **int** pageSize) {  Query query = **new** Query();  query.skip(pageNumber \* pageSize);  query.limit(pageSize);  **return** mongoTemplate.find(query, Person.**class**);  }    @Override  **public** Person findOneByName(String name) {  Query query = **new** Query();  query.addCriteria(Criteria.where("name").is(name));  **return** mongoTemplate.findOne(query, Person.**class**);  }  @Override  **public** List<Person> findByName(String name) {  Query query = **new** Query();  query.addCriteria(Criteria.where("name").is(name));  **return** mongoTemplate.find(query, Person.**class**);  }    @Override  **public** List<Person> findByBirthDateAfter(Date date) {  Query query = **new** Query();  query.addCriteria(Criteria.where("dateOfBirth").gt(date));  **return** mongoTemplate.find(query, Person.**class**);  }    @Override  **public** List<Person> findByAgeRange(**int** lowerBound, **int** upperBound) {  Query query = **new** Query();  query.addCriteria(Criteria.where("age").gt(lowerBound)  .andOperator(Criteria.where("age").lt(upperBound)));  **return** mongoTemplate.find(query, Person.**class**);  }    @Override  **public** List<Person> findByFavoriteBooks(String favoriteBook) {  Query query = **new** Query();  query.addCriteria(Criteria.where("favoriteBooks").in(favoriteBook));  **return** mongoTemplate.find(query, Person.**class**);  }    @Override  **public** **void** updateMultiplePersonAge() {    Query query = **new** Query();  query.addCriteria(Criteria.where("name").is("Markus"));  Update update = **new** Update();  update.set("name", "Nick");  User user = mongoTemplate.findAndModify(query, update, Person.**class**);    mongoTemplate.findAndModify(query, update, Person.**class**);;  }    @Override  **public** Person updateOnePerson(Person person) {  mongoTemplate.save(person);  **return** person;  }    @Override  **public** **void** deletePerson(Person person) {  mongoTemplate.remove(person);  }  }  **Configuration**  @Configuration  **public** **class** SpringMongoDBConfiguration {  @Autowired  **private** MongoDbProperties mongoDbProperties;  @Bean  **public** MongoDbFactory getMongoDbFactory() {  MongoClient client = **new** MongoClient(mongoDbProperties.getHost(), mongoDbProperties.getPort());  SimpleMongoDbFactory simpleMongoDbFactory = **new** SimpleMongoDbFactory(client, mongoDbProperties.getDataBase());  **return** simpleMongoDbFactory;  }    @Bean  **public** MongoTemplate getMongoTemplate() {  MongoTemplate mongoTemplate = **new** MongoTemplate(getMongoDbFactory());  **return** mongoTemplate;  }  }  **Reference**: <http://appsdeveloperblog.com/spring-boot-and-mongotemplate-tutorial-with-mongodb/>  Aggregation:  <https://xpadro.com/2016/04/data-aggregation-with-spring-data-mongodb-and-spring-boot.html> |